Butterfly Monitoring and Analyses

Reto Schmucki

2024-09-24

Table of contents

Preface

This project aims to collate documentation of methods used to analyze and work with Butterfly Monitoring Schemes (BMS) data. We focus on methods used to analyze count data associated with Pollard transects that represent the core of the eBMS database.

NOTE - this book project is a work in progress that only started.

Reto Schmucki, July 2024

Part I

Counting Butterflies

Butterfly Monitoring

Content under construction

Part II

From Counts to Flight Curves

Data Analyses and Methods

Biodiversity monitoring is essential to assess and understand the status and trends of species and ecosystems. Through active monitoring, we can integrate new information and update the knowledge needed for decision-making. Analyzing monitoring data can range from simple data exploration to the development of complex statistical models. A thorough understanding of both the data and the methods is fundamental to the selection and application of appropriate methods. Such an understanding will also provide important insights into the results enhancing our comprehension and helping us to communicate the results effectively to decision-makers.

Data Generation Processes

To develop robust monitoring methods and analyze the incoming data appropriately, it is important to understand the processes that generate the data and how the monitoring protocol affects the nature and structure of the data. For example, the life cycle of butterflies affects the number of adult individuals that can be observed in a given location at a given time. The seasonality of the emergence process creates a temporal pattern in the observed and recorded data that must be accounted for in the analysis. Systematic variation may also result from differences in sampling effort (e.g. the area sampled, the time spent recording or the experience of the recorders).

Understanding the influence of both biological and sampling components on the data generation process is crucial for the design and development of methods that can filter out the component of interest while accounting for systematic structures in the data. This also helps in assessing whether the monitoring program accurately captures and reflects the population of interest or whether the sample is biased and unrepresentative. When designing a monitoring program, it is important to identify the population to be monitored and understand how the sampling protocol may affect the representativeness of the data and introduce potential bias.

Data Simulation

To better understand the influence of species biology and sampling protocols on data generation, we will use data simulation approaches. Data simulation involves generating random data sets based on defined rules and known parameters. This technique is useful not only to illustrate the outcome of ecological and sampling processes but also to test methods and improve our understanding of statistical models and their potential failures.

When carefully designed, data simulation is a powerful tool for testing and validating methods and performing sensitivity analyses to assess their robustness to violations of underlying assumptions. Simulated data sets allow for exploring systematic patterns in the data and evaluating the behavior of models to identify their strengths and limitations.

In the following sections, we will use data simulations to illustrate and explore the different components of butterfly monitoring data generated under different scenarios. Through these simulations, we aim to 1) gain a deeper understanding of the data structure resulting from the ecological and sampling processes involved in butterfly monitoring schemes, 2) demonstrate and compare different modeling approaches and 3) understand the information that can be derived from these approaches.

Butterfly Counts

Simulation of Butterfly Counts

We will use simulated data to demonstrate and evaluate methods for calculating butterfly abundance indices, population trends and multi-species indicators such as the European Grassland Butterfly Indicator. To achieve this, we need to generate realistic data sets with known parameters. Data simulation will allow us to apply and test the methods on data generated under different scenarios. This approach will enable rigorous sensitivity analysis and provide crucial insights into the methods and a deeper understanding of their performance and limitations.

```
if(!require("data.table")) install.packages("data.table")
if(!require("ggplot2")) install.packages("ggplot2")
if(!require("devtools")) install.packages("devtools")
if(!require("rbms")) devtools::install_github("RetoSchmucki/rbms")

flc_col <- '#ff8c00'
cnt_col <- '#008b8b'
missing_col <- '#8b0000'
GAM_col <- '#483d8b'
```

Simulation Tool for Butterflies and Other Phenologies

Because butterflies' life cycle is strongly structured in time, with species-specific phenologies, we must account for this ecological process when simulating individual counts recorded across an entire season. The temporal pattern in the number of adult butterflies (imago) is determined by their emergence rate, the timing of the emergence and the life span of the adult. These parameters will often result in the number of adult individuals increasing over a certain period, up to a peak after when their number starts to decline. If a species can produce more than one generation per season, the number will display additional waves of emergence, each with its respective start, peak and decline periods. For each generation, the hump-shaped temporal pattern in the number of adults can often be described by a function that has a mean (center), variance (width) and a certain level of skewness (asymmetry). When merged, the individual patterns can become hidden under the pattern resulting from the cumulative effect of partly overlapping generations.

To simulate butterfly count data with such phenological patterns, we will use the function `timeseries_sim()` from the R package `butterflyGamSims` developed by Collin Edwards (see Edwards et al. 2023). This package is freely available on GitHub and will allow us to generate realistic data sets under scenarios with various levels of complexity.

We illustrate how the `timeseries_sim()` function works with a simple case where we simulate butterfly counts for a univoltine species with a Gaussian pattern (i.e. one generation with Normal distribution), where the peak abundance is observed at day 175 with a standard deviation of 15 days.

```
set.seed(13276)

if(!require("butterflyGamSims")) devtools::install_github("cbedwards/butterflyGamSims")

btfl_data <- timeseries_sim(nsims=1,
                year = c(2023),
                doy.samples = seq(from=1, to=365, by=1),
                abund.type = "exp",
                activity.type = "gauss",
                sample.type = "pois",
                sim.parms = list(growth.rate = 0,
                                 init.size = 500,
                                 act.mean = 175,
                                 act.sd = 15)
                )
```

The object produced by the `timeseries_sim()` function contains 1) a `data.frame` NAME$timeseries with the time series and 2) a `data.frame` NAME$parms with the parameters used for the simulation. In the parameters, you will find the population growth rate (`growth.rate`), the initial population size (`init. size`) measure in number of individuals expected over the season, the peak of the activity curve (`act.mean`) measured in days and the width of the activity curve (`act.sd`) that is measured in standard deviation.

> **Note**
>
> Note that not all sampling parameters used for the simulation are included in the parms object; the `activity.type` (the distribution function used to define the activity curve), the `sample.type` (the sampling process used to sample random counts along the activity curve) and the `abund.type` (the type of the growth rate, deterministic or with a log-normal process error) are missing.

Simulate Data Sampling Process

With the `timeseries_sim()` function above, we simulated a regular time series of butterfly counts where the actual number of active butterfly of each day are draw from a Poisson distribution with a given expectation defined by the activity curve along a day-of-year vector j following the probability density of a normal distribution (Gaussian) with mean equal to the peak day μ (`act.mean`) and a standard deviation α (`act.sd`). Because the integral of the probability density distribution (area under the curve) sum to 1, we can multiply the density by the abundance to retrieve a vector of expected abundance for each day-of-year, λ_j.

$$\lambda_j = abundance * \frac{1}{\alpha\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{(j-\mu)}{\alpha}\right)^2\right)$$

From the activity curve, we can use the `rpois()` function in R to draw a random value from a Poisson distribution, y_i representing the count for day j, where the mean is specified by expected value λ at day j.

$$y_j = rpois(\lambda_j)$$

From this simulation, we have generated the ecological process for the butterfly counts, for a given abundance distributed over a specific phenology defined by a Gaussian curve with a peak (mean) and a breath (standard deviation).

```
library(knitr)
kable(btfl_data$timeseries[c(1:3,160:163,250:253),])
```

Table 1: Butterfly Simmulation Data

	years	doy	count	act	abund.true	onset.true	median.true	end.true	fp.true	sim.id
1	2023	1	0	0.0000000	500	155.8	175	194.2	38.4	1
2	2023	2	0	0.0000000	500	155.8	175	194.2	38.4	1
3	2023	3	0	0.0000000	500	155.8	175	194.2	38.4	1
160	2023	160	9	8.0656908	500	155.8	175	194.2	38.4	1
161	2023	161	11	8.6025942	500	155.8	175	194.2	38.4	1
162	2023	162	10	9.1345489	500	155.8	175	194.2	38.4	1
163	2023	163	5	9.6563851	500	155.8	175	194.2	38.4	1
250	2023	250	0	0.0000496	500	155.8	175	194.2	38.4	1
251	2023	251	0	0.0000354	500	155.8	175	194.2	38.4	1
252	2023	252	0	0.0000252	500	155.8	175	194.2	38.4	1
253	2023	253	0	0.0000179	500	155.8	175	194.2	38.4	1

The additional structure resulting from the monitoring protocol (observation process) can now be added to the simulated time series and replicate a specific protocol. In this first case, we will simulate a protocol with weekly visits and include some missing counts for weeks when the minimal monitoring conditions were not met or the observer was absent. For this, we will write some new R functions. The first function will define the start and end of the monitoring season and sample one monitoring day per week over the season. Then we will write functions to simulate a certain level of missing weekly visits within the season. The likelihood of missing weeks tends to be higher at the beginning and the end of the season and lower in the middle. Let's start with the first function that defines the monitoring season and resamples one day of the simulated time series every week. We will name the function `sim2bms()` as it aligns the simulated time series to the protocol of a specific Butterfly Monitoring Scheme (BMS). The function needs the time series, and additional arguments to define the year that we want to extract `yearKeep`, if the time series must be resampled weekly `weeklySample`, which day should be used for the weekly resampling `weekdayKeep` (this can be a vector of days, e.g. c(2,3,4,5), or a specific day), and finally the monitoring season `monitoringSeason` which correspond to a vector of months, e.g c(4,5,6,7,8,9) represent a season starting in April and ending in September. Note that this function will also add some new variables such as the date, the ISO week number and the day of the week.

```
# data: Time series resulting from the simulation generated for 365 days
# weeklySample: TRUE or FALSE; should the daily count in the time series be resampled weekly?
# weekdayKeep: vector of days c(1,2,..., 7) to be sampled from for the weekly count. If the vector
# contains c(2,3,4), the sampling process will be restricted to Tuesday, Wednesday or Thursday.
# monitoringSeason: vector of months that define the monitoring season (e.g. April to September is c(

sim2bms <- function(data, yearKeep = NULL, weeklySample = FALSE, weekdayKeep = NULL, monitoringSeason
```

```
                btfl_ts <- data.table::data.table(data)[, site_id := paste0("site_",sim.id)]
                if(!is.null(yearKeep)){
                      btfl_ts <- btfl_ts[years %in% yearKeep, ]
                }
                btfl_ts[ , date := as.Date(doy, origin = paste0(years, "-01-01"))-1]
                btfl_ts[ , week := isoweek(date)]
                btfl_ts[month(date) != 1 | week < 50, weekday := rowid(week), by = .(site_id, years)]
                if(isTRUE(weeklySample)){
                      if(!is.null(weekdayKeep)){
                            btfl_ts <- btfl_ts[weekday %in% weekdayKeep, ]
                      }
                      btfl_ts <- btfl_ts[btfl_ts[,.I[sample(.N, 1)], by = .(week, site_id, years)][["V
                }
                if(!is.null(monitoringSeason)){
                btfl_ts <- btfl_ts[month(date) %in% monitoringSeason, ]
                }
          return(btfl_ts)
          }
```

We can apply this function to retrieve a specific year of the simulated time series and add some new variables, leaving all other parameters empty. With the same function, we can also resample weekly counts (e.g. one day from c(2:5)) and restrict the time series to a specific monitoring season (e.g. c(4:9)).

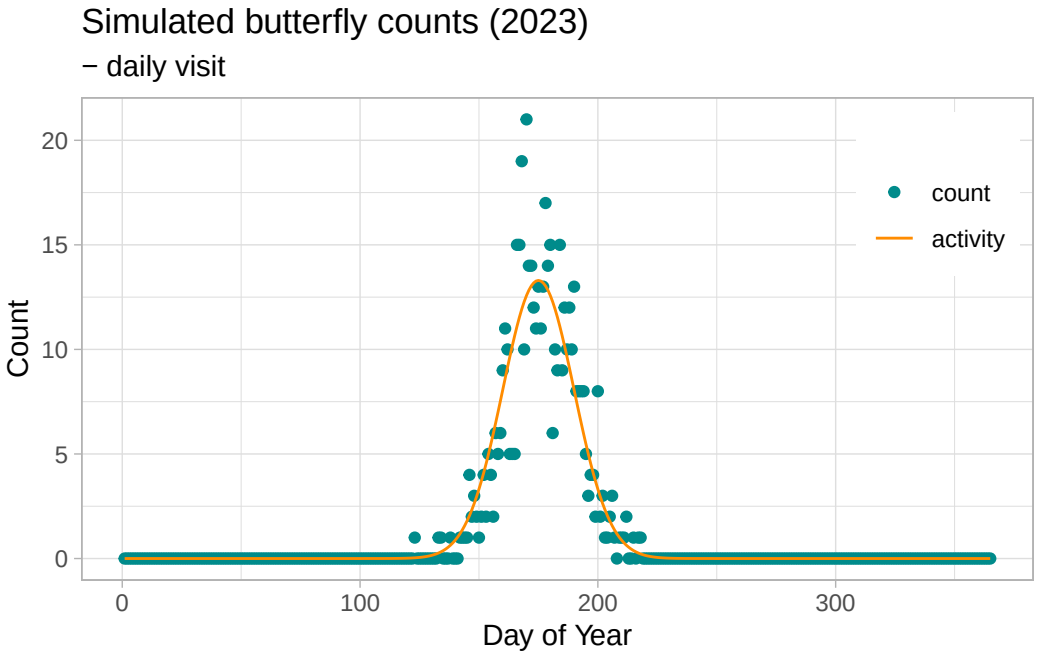
```
set.seed(13276)
y <- c(2023)

btfl_ts <- sim2bms(data = btfl_data$timeseries, yearKeep = y)

btfl_fig1 <- ggplot() +
               geom_point(data=btfl_ts, aes(x=doy, y=count, colour = "count")) +
               geom_line(data = btfl_ts,
               aes(x = doy, y = act, colour = "activity")) +
               xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
               scale_colour_manual("",
                     breaks = c("count", "activity"),
                     values = c(cnt_col, flc_col)) +
               theme_light() +
               theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
               labs(title = paste0("Simulated butterfly counts (", y,")"),
                    subtitle = "- daily visit",
                    x = "Day of Year",
                    y = "Count")

btfl_week_smpl <- sim2bms(data = btfl_data$timeseries, yearKeep = y,
                  weeklySample = TRUE,
                  weekdayKeep = c(2:5),
                  monitoringSeason = c(4:9))
btfl_fig2 <- ggplot() +
               geom_point(data=btfl_week_smpl, aes(x=doy, y=count, colour = "count")) +
```

```
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
                      breaks = c("count", "activity"),
                      values = c(cnt_col, flc_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- weekly visit (random resampled)",
                     x = "Day of Year",
                     y = "Count")

btfl_fig1
btfl_fig2
```

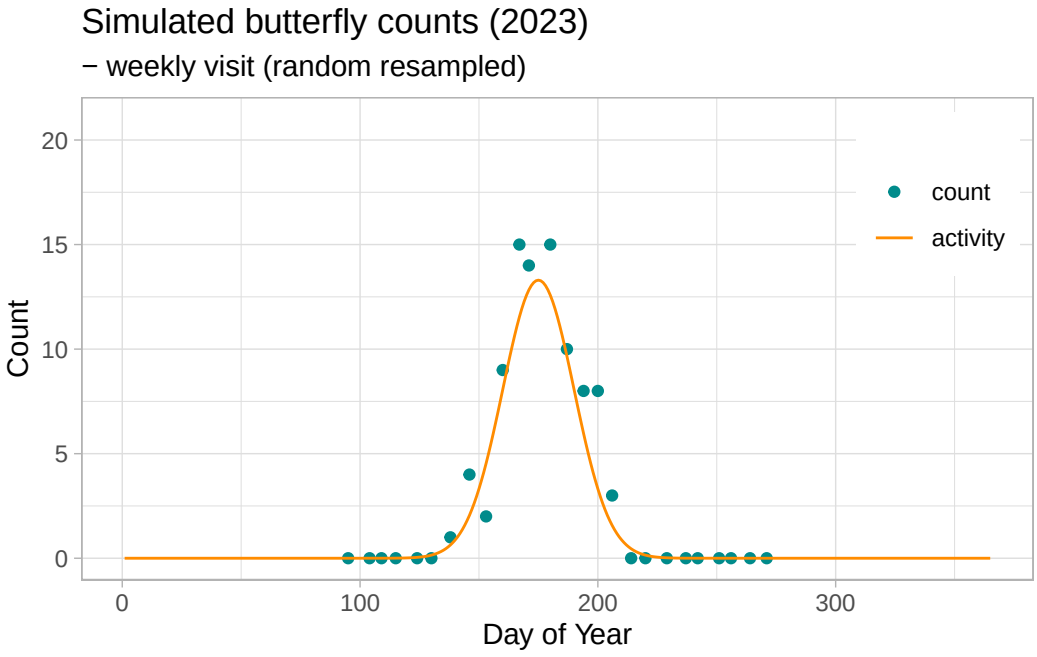
In the example above, the activity curve represented by the line has a Gaussian shape and counts presented by the points along the curve are independent random samples from a Poisson distribution. Because we sampled a count value for 365 days (day-of-year; doy), the counts are representative of the population of active adult butterflies as if the site was visited every. This implies that a proportion of butterflies are counted more than one day as their lifespan exceeds one day. On Pollard transect, this is how butterfly counts are likely to be counted and reported, but with a different frequency as visits are generally weekly, fortnightly, or even monthly. We can replicate this value by resampling the daily count weekly.

From the weekly visits, counts outside of the monitoring period will not be informed, in many cases these are 'zeros' as we expect the monitoring season to align with butterflies' activity. Some other weeks might be missing from the time series, potentially due to unsuitable weather conditions for monitoring or the recorder's availability. We can inform and exclude the missing visits by resampling a subset of the weekly counts.

```
missing_prob <- function(data, mu=NULL, alpha = 5, theta = 0.3){
                              x_ <- seq_len(nrow(data))
                              mu_ <- ifelse(is.null(mu), length(x_) / 2, mu)
                              std_ <- sqrt(mu_ / theta)
                              y_ <- abs((alpha * exp((-(x_ - mu_)^2) / std_^2)) - alpha) + alpha
                              yn_ <- y_ / (sum(y_))
                        return(yn_)
                  }

sample_missing <- function(data, propMissing = 0.25){

             missing.prob <- data.table::data.table()
             for(i in data[, unique(years)]){
                   for(j in data[, unique(site_id)]){
                   missing.prob <- rbind(missing.prob, missing_prob(data[years == i & site_id == j, ])
                   }
             }

             missing.week <- data[sample(seq_len(.N), round(propMissing * .N), prob = unlist(missing.p

       return(missing.week)
```

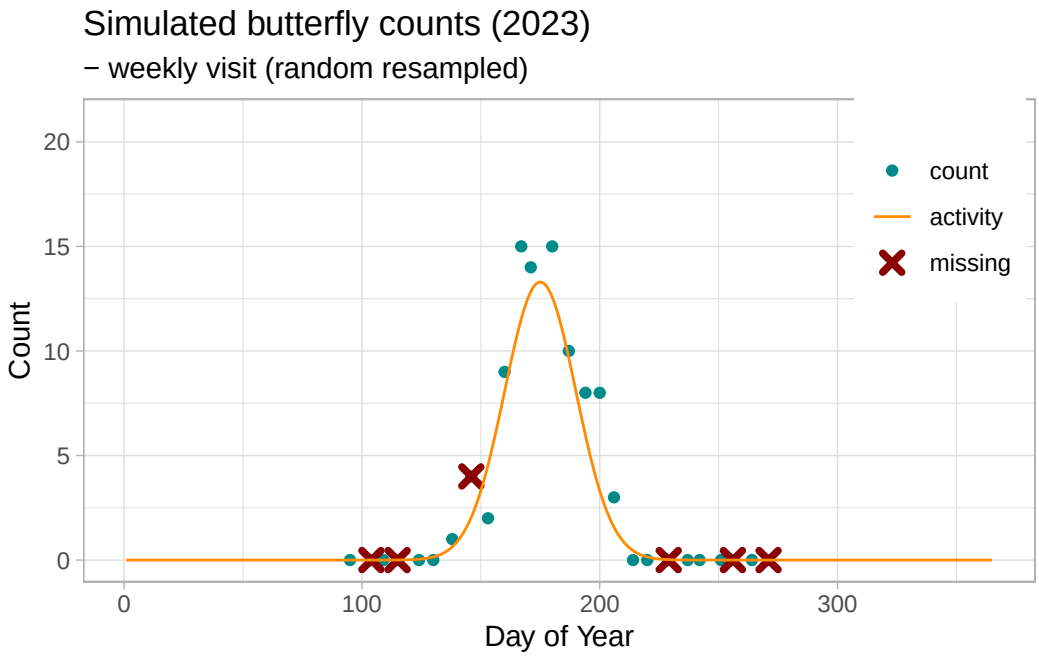

Simulated butterfly counts (2023)
− daily visit
Count
Day of Year
count
activity
0
5
10
15
20
0
100
200
300

(a) a)


Simulated butterfly counts (2023)
− weekly visit (random resampled)
Count
Day of Year
count
activity
0
5
10
15
20
0
100
200
300

(a) b)

```
}

btfl_week_missing <- sample_missing(data = btfl_week_smpl, propMissing = 0.25)

btfl_fig3 <- ggplot() +
                geom_point(data=btfl_week_smpl, aes(x=doy, y=count, colour = "count")) +
                geom_point(data=btfl_week_missing, aes(x=doy, y=count, colour = "missing"),
                           shape=4, size=2, stroke=2) +
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
                      breaks = c("count", "activity", "missing"),
                      values = c(cnt_col, flc_col, missing_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- weekly visit (random resampled)",
                     x = "Day of Year",
                     y = "Count")
btfl_fig3
```



Generalized Additive Models with rbms

We will use the simulation to test the GAM method implemented in the R package rbms (Schmucki, Harrower A., and Dennis B. 2022). Because recorders only report the number of observed butterflies, zeros are generally

not reported but can be derived from the visit dates.

Organising BMS count data

```
visit_sim <- btfl_week_smpl[!date %in% btfl_week_missing$date, .(site_id, date, count)]
count_sim <- visit_sim[count>=1,][, species := "sp1"]

names(visit_sim) <- toupper(names(visit_sim))
names(count_sim) <- toupper(names(count_sim))

ts_date <- rbms::ts_dwmy_table(InitYear = 2023, LastYear = 2023, WeekDay1 = 'monday')

ts_season <- rbms::ts_monit_season(ts_date,
                        StartMonth = 4,
                        EndMonth = 9,
                        StartDay = 1,
                        EndDay = NULL,
                        CompltSeason = TRUE,
                        Anchor = TRUE,
                        AnchorLength = 2,
                        AnchorLag = 2,
                        TimeUnit = 'd')

ts_season_visit <- rbms::ts_monit_site(ts_season, visit_sim)

ts_season_count <- rbms::ts_monit_count_site(ts_season_visit, count_sim, sp = "sp1")
```

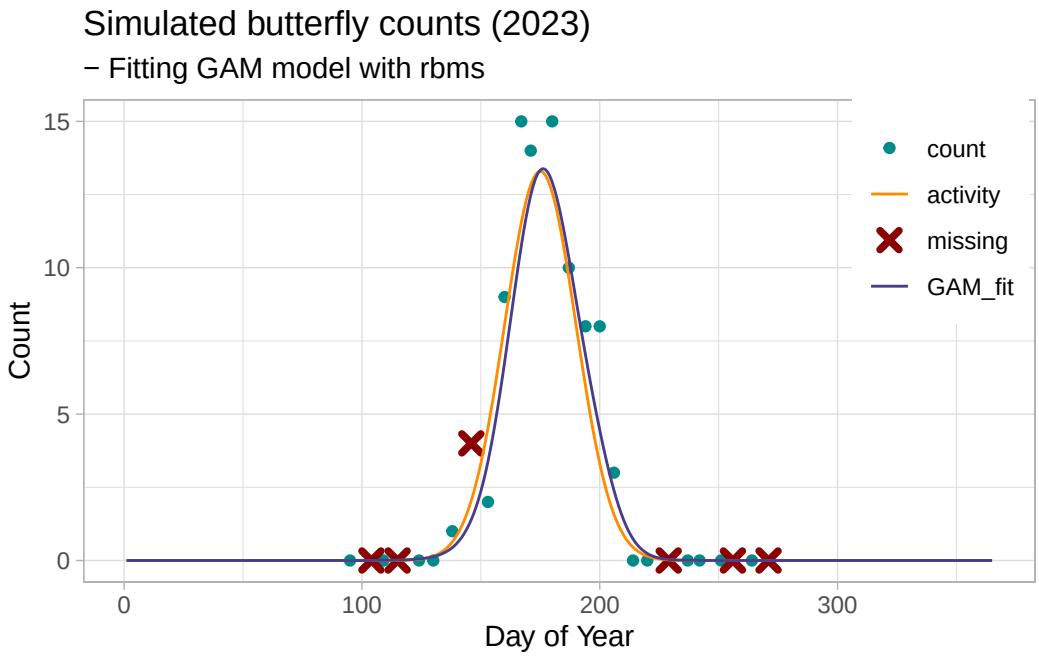
Fitting a GAM to Butterfly Counts

```
ts_flight_curve <- rbms::flight_curve(ts_season_count,
                        NbrSample = 300,
                        MinVisit = 5,
                        MinOccur = 3,
                        MinNbrSite = 1,
                        MaxTrial = 4,
                        GamFamily = 'nb',
                        SpeedGam = FALSE,
                        CompltSeason = TRUE,
                        SelectYear = NULL,
                        TimeUnit = 'd')
```

The flight curve computed by the `rbms::flight_curve()` function is stored in the …$pheno object where the days of year are stored under the variable `trimDAYNO` and the standardized flight curve under the variable `NM`. The `NM` variable is scaled to an Area Under the Curve (AUC) that sum to 1. To compare the flight curve derived from the GAM with the activity curve used for the simulation, we must rescale them to the same AUC, in other words, we must rescale the activity curve to have an AUC of 1 or rescale the `NM` to the population size used for

the simulation. Here we will rescale the `NM` to match the simulation population size, this will allow us to display the curves and the counts on the same plot with the correct scale.

```
pheno <- ts_flight_curve$pheno

btfl_fig4 <- ggplot() +
                geom_point(data=ts_season_count[ANCHOR == 0 & !is.na(COUNT), ], aes(x=DAY_SINCE, y=CO
                geom_point(data=btfl_week_missing, aes(x=doy, y=count, colour = "missing"),
                           shape=4, size=2, stroke=2) +
                geom_line(data = btfl_ts, aes(x = doy, y = act, colour = "activity")) +
                geom_line(data = pheno,
                    aes(x = trimDAYNO, y = btfl_ts[,unique(abund.true)]*NM, colour = "GAM_fit")) +
                xlim(1,365) + ylim(0, max(btfl_ts$act,
                                          pheno$NM*btfl_ts[,unique(abund.true)],
                                          btfl_week_missing$count,
                                          ts_season_count[!is.na(COUNT), COUNT] )) +
                scale_colour_manual("",
                      breaks = c("count", "activity", "missing", "GAM_fit"),
                      values = c(cnt_col, flc_col, missing_col, GAM_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- Fitting GAM model with rbms",
                     x = "Day of Year",
                     y = "Count")
btfl_fig4
```



To compare the fitted curve with the activity curve, we should use a standard AUC of 1 to enable a fair

comparison between models fitted to different population sizes. Using the standardized activity curve and the GAM-generated flight curve (`NM`), we can calculate the Root Mean Squared Error (RMSE) to estimate the goodness of fit of the flight curve generated with the rbms package.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2}$$

where y_i is the `NM` value at time i and $\tilde{y}_i$ the value from the standardized activity curve at time i, from day 1 to n of the monitoring season.

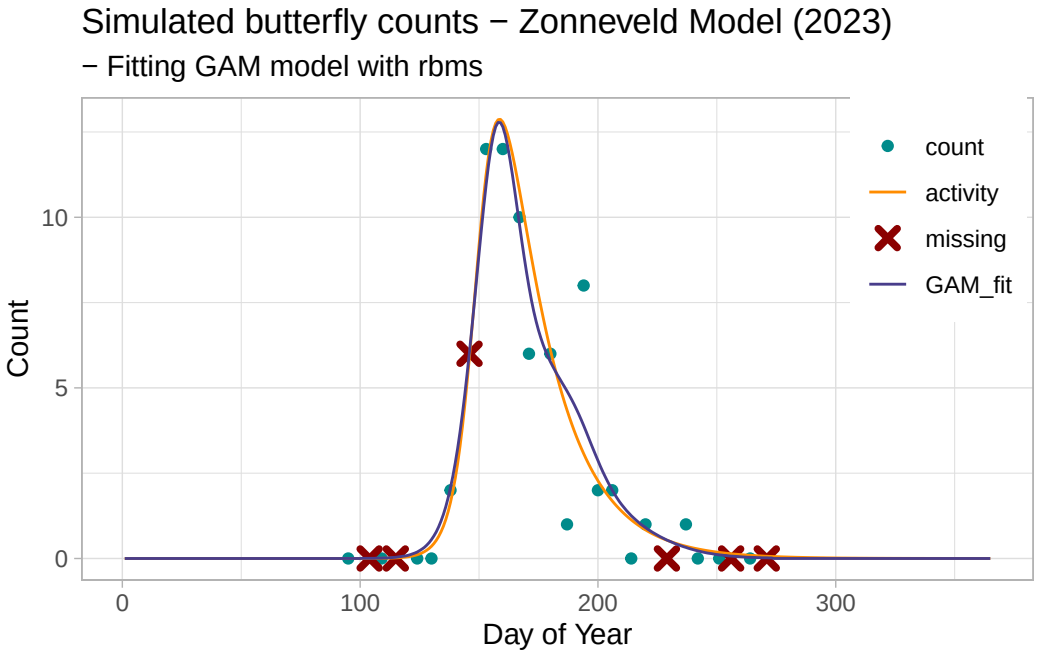
Non Gaussian flight curve

The same procedure can be applied to flight curves having more complex shapes. Here we will generate a time series of butterfly counts drawn from a known flight curve (adult activity), using simulation from a Zonneveld model.

```
btfl_data_zn <- timeseries_sim(nsims=1,
                year = c(2023),
                doy.samples = seq(from=1, to=365, by=1),
                abund.type = "exp",
                activity.type = "zon",
                sample.type = "pois",
                sim.parms = list(growth.rate = 0,
                                 init.size = 500,
                                 act.mean = 175,
                                 act.sd = 15,
                                 #theta = 5,
                                 zon.theta = 50,
                                 t0 = 100,
                                 beta = 5,
                                 alpha = 0.05)
                )


set.seed(13276)
btfl_ts <- sim2bms(data = btfl_data_zn$timeseries, yearKeep = y)

btfl_week_smpl <- sim2bms(data = btfl_data_zn$timeseries, yearKeep = y,
                  weeklySample = TRUE,
                  weekdayKeep = c(2:5),
                  monitoringSeason = c(4:9))

btfl_week_missing <- sample_missing(data = btfl_week_smpl, propMissing = 0.25)

visit_sim <- btfl_week_smpl[!date %in% btfl_week_missing$date, .(site_id, date, count)]
count_sim <- visit_sim[count>=1,][, species := "sp1"]

names(visit_sim) <- toupper(names(visit_sim))
```

```r
names(count_sim) <- toupper(names(count_sim))

ts_date <- rbms::ts_dwmy_table(InitYear = 2023, LastYear = 2023, WeekDay1 = 'monday')

ts_season <- rbms::ts_monit_season(ts_date,
                        StartMonth = 4,
                        EndMonth = 9,
                        StartDay = 1,
                        EndDay = NULL,
                        CompltSeason = TRUE,
                        Anchor = TRUE,
                        AnchorLength = 2,
                        AnchorLag = 2,
                        TimeUnit = 'd')

ts_season_visit <- rbms::ts_monit_site(ts_season, visit_sim)

ts_season_count <- rbms::ts_monit_count_site(ts_season_visit, count_sim, sp = "sp1")

# mod_k <- "COUNT ~ s(DAY_SINCE, bs =\"cr\", k = 5) + factor(SITE_ID)"

ts_flight_curve <- rbms::flight_curve(ts_season_count,
                        NbrSample = 300,
                        MinVisit = 5,
                        MinOccur = 3,
                        MinNbrSite = 1,
                        MaxTrial = 4,
                        GamFamily = 'nb',
                        SpeedGam = FALSE,
                        CompltSeason = TRUE,
                        SelectYear = NULL,
                        #mod_form = mod_k,
                        TimeUnit = 'd')

pheno <- ts_flight_curve$pheno

btfl_fig5 <- ggplot() +
                geom_point(data=ts_season_count[ANCHOR == 0 & !is.na(COUNT), ], aes(x=DAY_SINCE, y=CO
                geom_point(data=btfl_week_missing, aes(x=doy, y=count, colour = "missing"),
                            shape=4, size=2, stroke=2) +
                geom_line(data = btfl_ts, aes(x = doy, y = act, colour = "activity")) +
                geom_line(data = pheno, aes(x = trimDAYNO, y = btfl_ts[,unique(abund.true)]*NM, colou
                xlim(1,365) + ylim(0, max(btfl_ts$act,
                                          pheno$NM*btfl_ts[,unique(abund.true)],
                                          btfl_week_missing$count,
                                          ts_season_count[!is.na(COUNT), COUNT] )) +
                scale_colour_manual("",
                      breaks = c("count", "activity", "missing", "GAM_fit"),
                      values = c(cnt_col, flc_col, missing_col, GAM_col)) +
```

```
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts - Zonneveld Model (", y,")"),
                     subtitle = "- Fitting GAM model with rbms",
                     x = "Day of Year",
                     y = "Count")
btfl_fig5
```



Simple trend case

In the first scenario, we will apply the method to a simple case where we have one univoltine species that is monitored over 15 years across 100 sites where the populations follow the same trend with a known growth rate.

Statistical model and flight curve

Generalized Additive Model approach

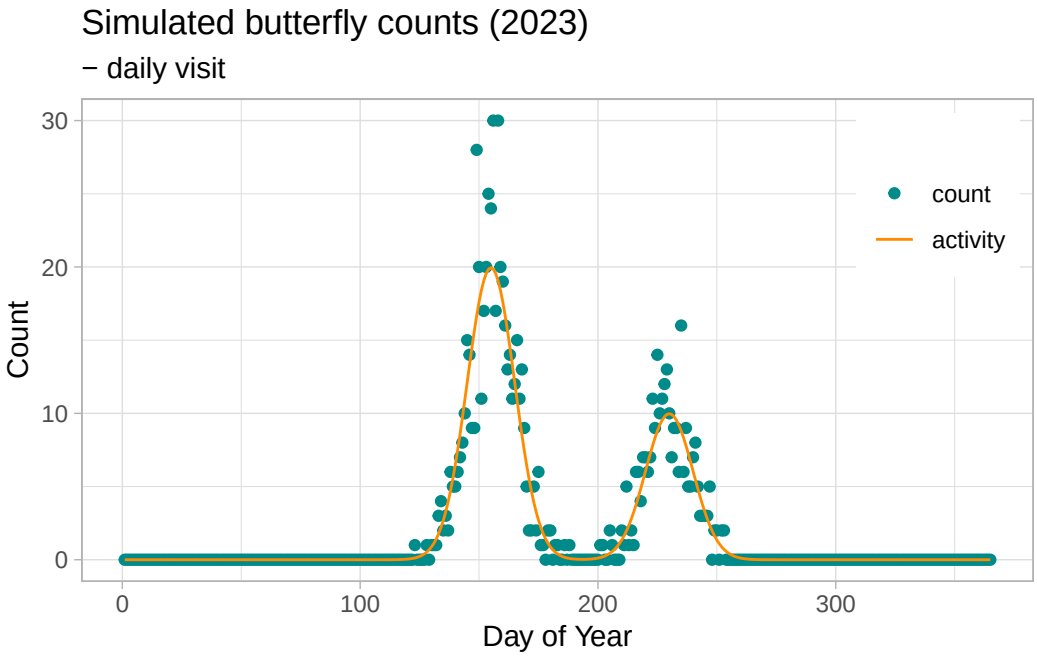
```
set.seed(13276)

if(!require("rbms")) devtools::install_github("RetoSchmucki/rbms")
if(!require("butterflyGamSims")) devtools::install_github("cbedwards/butterflyGamSims")
if(!require("mixR")) devtools::install_github("RetoSchmucki/mixR") # small fix in plot function, not

library(rbms)
library(mixR)
library(butterflyGamSims)
library(data.table)
library(ggplot2)

flc_col <- '#ff8c00'
cnt_col <- '#008b8b'
missing_col <- '#8b0000'
GAM_col <- '#483d8b'

## local functions
sim2bms <- function(data, yearKeep = NULL, weeklySample = FALSE, weekdayKeep = NULL, monitoringSeason

                  btfl_ts <- data.table::data.table(data)[, site_id := paste0("site_",sim.id)]
                  if(!is.null(yearKeep)){
                        btfl_ts <- btfl_ts[years %in% yearKeep, ]
                  }
                  btfl_ts[ , date := as.Date(doy, origin = paste0(years, "-01-01"))-1]
                  btfl_ts[ , week := isoweek(date)]
                  btfl_ts[month(date) != 1 | week < 50, weekday := rowid(week), by = .(site_id, years
                  if(isTRUE(weeklySample)){
                        if(!is.null(weekdayKeep)){
                              btfl_ts <- btfl_ts[weekday %in% weekdayKeep, ]
                        }
                        btfl_ts <- btfl_ts[btfl_ts[,.I[sample(.N, 1)], by = .(week, site_id, years)][
                  }
                  if(!is.null(monitoringSeason)){
```

```
                  btfl_ts <- btfl_ts[month(date) %in% monitoringSeason, ]
                  }
            return(btfl_ts)
            }

missing_prob <- function(data, mu=NULL, alpha = 5, theta = 0.3){
                              x_ <- seq_len(nrow(data))
                              mu_ <- ifelse(is.null(mu), length(x_) / 2, mu)
                              std_ <- sqrt(mu_ / theta)
                              y_ <- abs((alpha * exp((-(x_ - mu_)^2) / std_^2)) - alpha) + alpha
                              yn_ <- y_ / (sum(y_))
                        return(yn_)
                  }

sample_missing <- function(data, propMissing = 0.25){

            missing.prob <- data.table::data.table()
            for(i in data[, unique(years)]){
                  for(j in data[, unique(site_id)]){
                  missing.prob <- rbind(missing.prob, missing_prob(data[years == i & site_id == j, ]))
                  }
            }

            missing.week <- data[sample(seq_len(.N), round(propMissing * .N), prob = unlist(missing.prob

      return(missing.week)
}
```

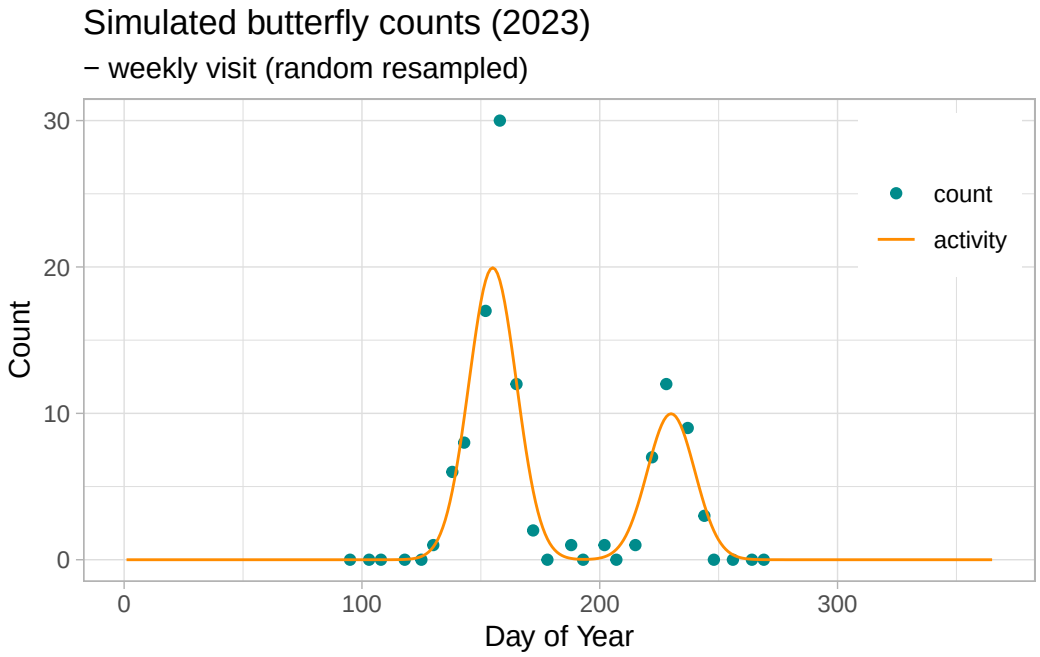
Simulated count

```
size1 <- 500
peak1 <- 155
sd1 <- 10
size2 <- 250
peak2 <- 230
sd2 <- 10
y <- c(2023)

set.seed(13276)
btfl_data1 <- timeseries_sim(nsims=1,
                year = y,
                doy.samples = seq(from=1, to=365, by=1),
                abund.type = "exp",
                activity.type = "gauss",
                sample.type = "pois",
                sim.parms = list(growth.rate = 0,
                                 init.size = size1,
```

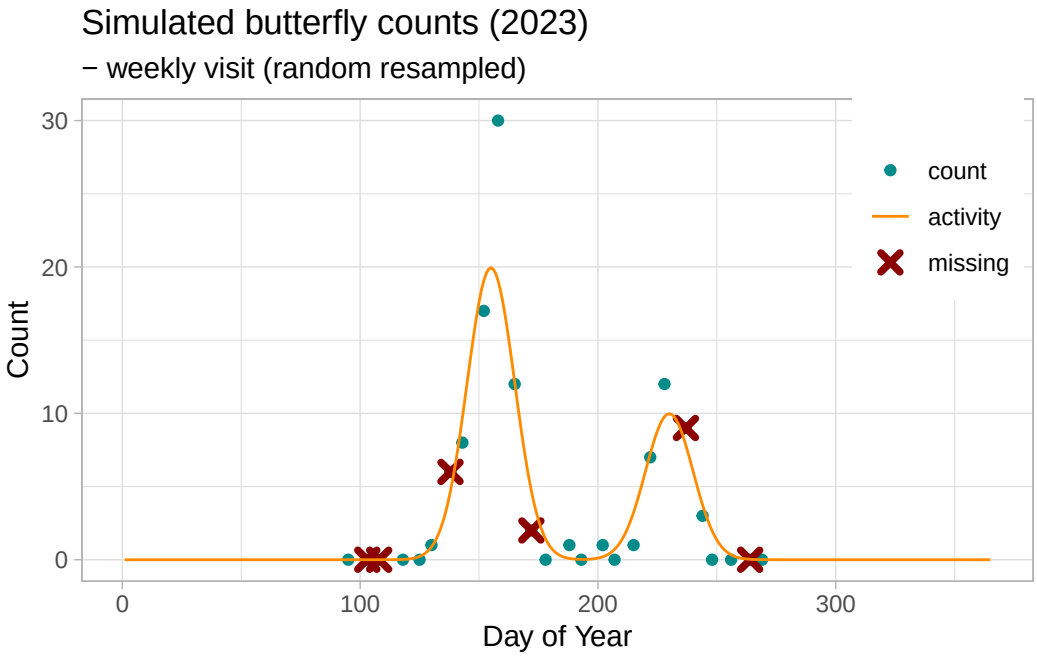
```
                                act.mean = peak1,
                                act.sd = sd1)
                )

set.seed(13276)
btfl_data2 <- timeseries_sim(nsims=1,
                year = y,
                doy.samples = seq(from=1, to=365, by=1),
                abund.type = "exp",
                activity.type = "gauss",
                sample.type = "pois",
                sim.parms = list(growth.rate = 0,
                                init.size = size2,
                                act.mean = peak2,
                                act.sd = sd2)
                )

btfl_data_b2 <- rbind(btfl_data1$timeseries[,c("years", "doy", "count", "act", "sim.id")],
                      btfl_data2$timeseries[,c("years", "doy", "count", "act", "sim.id")])
btfl_data_b2 <- unique(data.table(btfl_data_b2)[, ":="(count=sum(count), act=sum(act)), by = .(years,
btfl_data_b2[, abund.true:= sum(c(size1, size2))]

btfl_ts <- sim2bms(data = btfl_data_b2, yearKeep = y)

btfl_fig1 <- ggplot() +
                geom_point(data=btfl_ts, aes(x=doy, y=count, colour = "count")) +
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
                      breaks = c("count", "activity"),
                      values = c(cnt_col, flc_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- daily visit",
                     x = "Day of Year",
                     y = "Count")
btfl_fig1
```




```
## ============================
## degradation1: weekly count
## ============================

btfl_week_smpl <- sim2bms(data = btfl_data_b2, yearKeep = y,
                  weeklySample = TRUE,
                  weekdayKeep = c(2:5),
                  monitoringSeason = c(4:9))

btfl_fig2 <- ggplot() +
                geom_point(data=btfl_week_smpl, aes(x=doy, y=count, colour = "count")) +
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
                      breaks = c("count", "activity"),
                      values = c(cnt_col, flc_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- weekly visit (random resampled)",
                     x = "Day of Year",
                     y = "Count")

btfl_fig2
```




```
## ==============================
## degradation: add missing week
## ==============================

btfl_week_missing <- sample_missing(data = btfl_week_smpl, propMissing = 0.25)

btfl_fig3 <- ggplot() +
                geom_point(data=btfl_week_smpl, aes(x=doy, y=count, colour = "count")) +
                geom_point(data=btfl_week_missing, aes(x=doy, y=count, colour = "missing"),
                           shape=4, size=2, stroke=2) +
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
                      breaks = c("count", "activity", "missing"),
                      values = c(cnt_col, flc_col, missing_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- weekly visit (random resampled)",
                     x = "Day of Year",
                     y = "Count")

btfl_fig3
```



The flight curve underlying the observed butterfly counts can be seen as a function of time, with counts increasing as adult individuals emerge and decreasing as they die. In the case of a bivoltine species, we would expect two emergence phases over the year, one for each generation. The shape of this bimodal function can be represented by a spline derived from the GAM using the `flight_curve` function in the rbms package. By organising the simulated data into a table of monitoring visits and a table of counts, we can generate a time series of counts for the 2023 season.

The `ts_dwmy_table` function builds a *time series* where every *day*, *week*, *month* and *year* is informed for the defined period (`InitYear` and `LastYear`). We can then inform the monitoring season over the time series, informing the start and end month, if we want to include additional zeros (*anchor*) at the start and end of the monitoring season, and how strong the *anchor* should be. Here we will define a monitoring season that starts on April (4) 1 and ends on the last day of September (9). We will add anchors of two zeros positioned two days before and after the monitoring season. As the time series is defined on a daily (d) basis, the anchors and the position (lag) are in the same units.

```
visit_sim <- btfl_week_smpl[!date %in% btfl_week_missing$date, .(site_id, date, count)]
count_sim <- visit_sim[count>=1,][, species := "sp1"]

names(visit_sim) <- toupper(names(visit_sim))
names(count_sim) <- toupper(names(count_sim))

ts_date <- rbms::ts_dwmy_table(InitYear = 2023, LastYear = 2023, WeekDay1 = 'monday')

ts_season <- rbms::ts_monit_season(ts_date,
                        StartMonth = 4,
                        EndMonth = 9,
                        StartDay = 1,
                        EndDay = NULL,
```

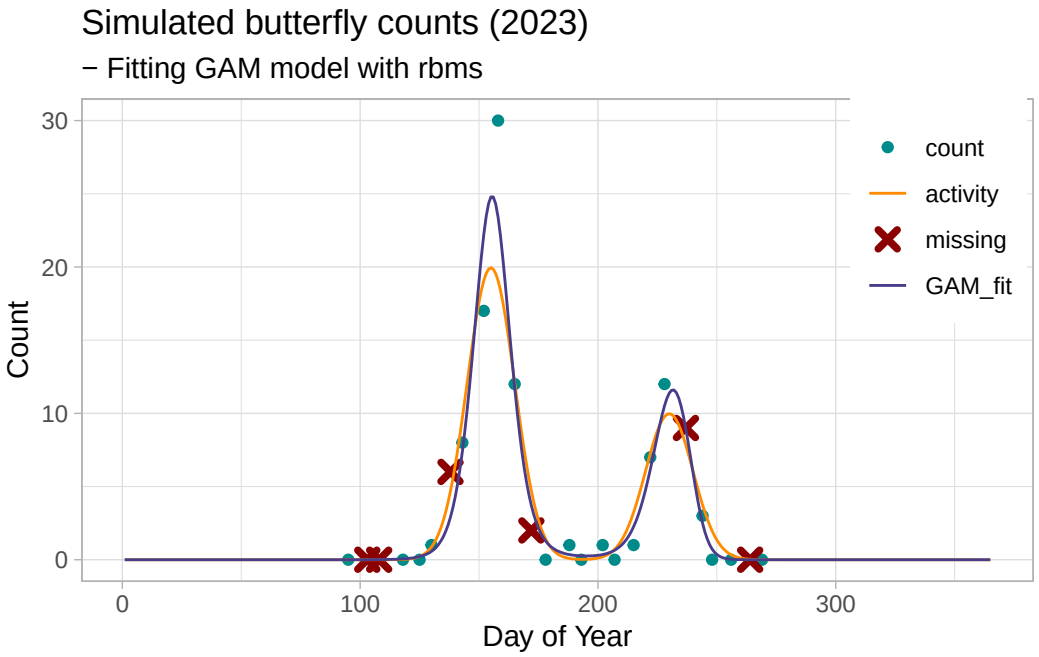
```
                        CompltSeason = TRUE,
                        Anchor = TRUE,
                        AnchorLength = 2,
                        AnchorLag = 2,
                        TimeUnit = 'd')

ts_season_visit <- rbms::ts_monit_site(ts_season, visit_sim)

ts_season_count <- rbms::ts_monit_count_site(ts_season_visit, count_sim, sp = "sp1")
```

We can now use the `flight_curve` function of the rbms package to fit a GAM and get the spline corresponding to the smoothed function of the number of butterflies recorded over the time series, days. Here we will fit the model on data simulated for one site over one year (2023). A site is only included if it has been visited at least five time over the season and had 3 visit with positive occurrence. Although this is not very relevant for this dataset, these filters can be useful to discard some poorly monitored site. Remember that the purpose of this model is to inform about temporal change in abundance related to species' phenology and that this information is better in more complete time series and where the species and change in abundance has been adequately detected over the season.

When plotting the resulting flight curve, we can choose to plot the standardised form (NM), where the area under the curve is adjusted to sum to 1, or the curve converted to the count scale by multiplying the standardised curve by the total abundance. Here we plot the flight curve on the count scale.

```
### Fitting a GAM to Butterfly Counts

ts_flight_curve <- rbms::flight_curve(ts_season_count,
                        NbrSample = 300,
                        MinVisit = 5,
                        MinOccur = 3,
                        MinNbrSite = 1,
                        MaxTrial = 4,
                        GamFamily = 'nb',
                        SpeedGam = FALSE,
                        CompltSeason = TRUE,
                        SelectYear = NULL,
                        TimeUnit = 'd')


## plot-fitted-flight-curve

pheno <- ts_flight_curve$pheno

btfl_fig4 <- ggplot() +
                geom_point(data=ts_season_count[ANCHOR == 0 & !is.na(COUNT), ], aes(x=DAY_SINCE, y=CO
                geom_point(data=btfl_week_missing, aes(x=doy, y=count, colour = "missing"),
                            shape=4, size=2, stroke=2) +
                geom_line(data = btfl_ts, aes(x = doy, y = act, colour = "activity")) +
                geom_line(data = pheno,
                    aes(x = trimDAYNO, y = btfl_ts[, unique(abund.true)]*NM, colour = "GAM_fit")) +
                xlim(1,365) + ylim(0, max(btfl_ts$act,
```

```
                                        pheno$NM*btfl_ts[,unique(abund.true)],
                                        btfl_week_missing$count,
                                        ts_season_count[!is.na(COUNT), COUNT] )) +
                  scale_colour_manual("",
                        breaks = c("count", "activity", "missing", "GAM_fit"),
                        values = c(cnt_col, flc_col, missing_col, GAM_col)) +
                  theme_light() +
                  theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                  labs(title = paste0("Simulated butterfly counts (", y,")"),
                       subtitle = "- Fitting GAM model with rbms",
                       x = "Day of Year",
                       y = "Count")
btfl_fig4
```



GAM basis for flight curve spline

The spline that represents the flight curve is build from addition of individual basis functions. Although the individual basis functions have no direct interpretation, it may be helpful to plot them to understand the building block of the spline and the mechanics of the Generalised Additive Model (GAM). To do this, we will use the R package `gratia`, which contains useful functions for exploring, evaluating and plotting GAMs. First we will retrieve the model, which is stored in the ts_flight_curve object under the slot, model. This is a list of annual flight curve models (GAM). Here we use the 1st model in the list (we only have one year, 2023). From this model we can retrieve the summary of the model and the number of knots used for the spline.

```
if(!require("gratia")) install.packages("gratia")
```

```
Loading required package: gratia
```

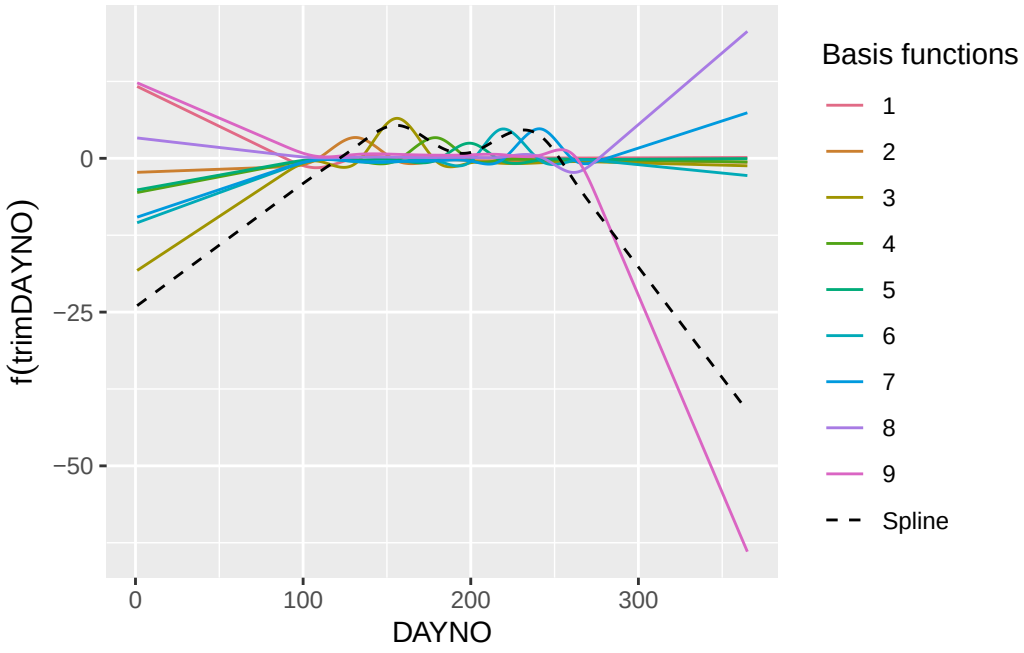
```
  library("gratia")
  library("colorspace")

  ## retrieve GAM model
  mod <- ts_flight_curve$model[[1]]
  summary(mod)
```

```
Family: Negative Binomial(29.077)
Link function: log

Formula:
COUNT ~ s(trimDAYNO, bs = "cr")

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -2.142      1.508  -1.421    0.155

Approximate significance of smooth terms:
              edf Ref.df Chi.sq p-value
s(trimDAYNO) 5.86  6.598  67.59  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.916   Deviance explained = 94.5%
-REML = 36.461  Scale est. = 1         n = 24
```

```
  mod$smooth[[1]]$bs.dim # knots for the spline
```

```
[1] 10
```

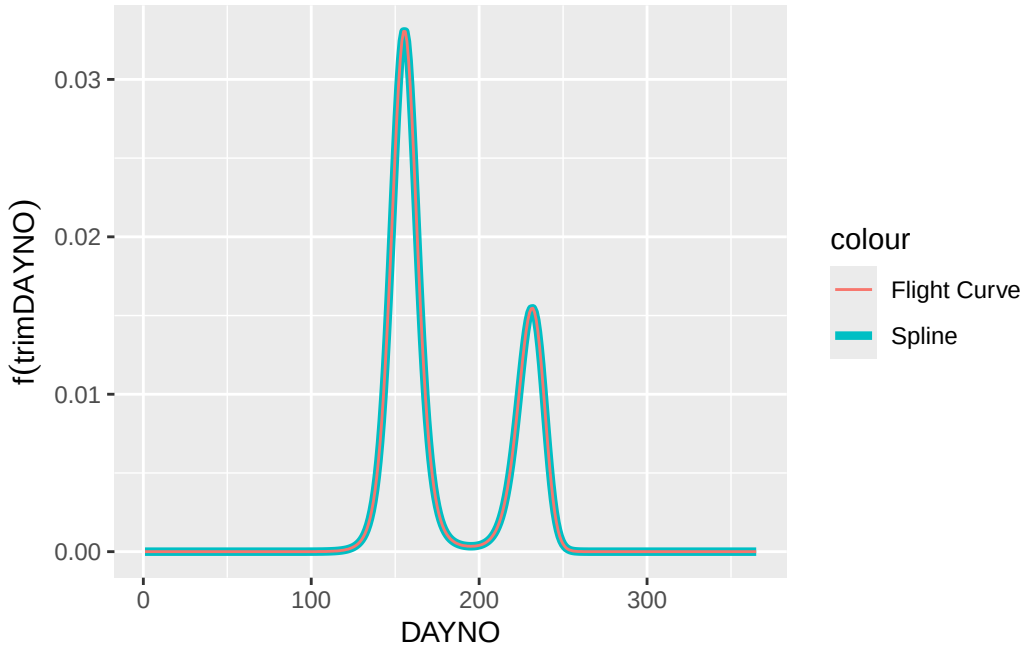
```
  ## retrieve basis for the smooth trimDAYNO
  x2_bs <- data.table(gratia::basis(mod, select = "s(trimDAYNO)", data = ts_flight_curve$data[, c("trim
  x2_spl <- x2_bs[, spline := sum(.value), by = trimDAYNO]

  # plot GAM basis
  ggplot() +
      geom_line(data= x2_bs, aes(x = trimDAYNO, y = .value, colour = .bf, group = .bf)) +
      geom_line(data = x2_spl, aes(x = trimDAYNO, y = spline, colour = "Spline"), lty =2) +
      labs(y = expression(f(trimDAYNO)), x = "DAYNO") +
      scale_colour_manual("Basis functions", breaks =  c(1:9,"Spline"), values = c(qualitative_hcl(9, p
      theme(legend.key = element_blank())
```



You might think that this (black dashed line) does not really look like the flight curve estimated and returned by the `rbms::flight_curve` function. One thing to remember is that the model is fitted using the log-link, which means that the spline is estimated on the log scale and so are the bases that underlie the spline. The spline is the sum of the 9 basis functions. To demonstrate this, we can extract the spline that is the sum of the bases and then transform the spline to the count scale (i.e. exp()). Since we assume that the count outside the monitoring season is zero, we will multiply the result by the binary variable (0,1) `M_SEASON`. We will then normalise the spline so that the area under the curve sums to one.

> **Note**
>
> Note that the individual bases have no direct interpretation in the model other than as part of the building blocks of the spline, so their contribution must be considered in the context of the other bases. Also, it is the sum of the bases, the spline, that is optimised in the likelihood function. While the spline can be transformed back to the count scale, the basis should not be, as the spline is the sum of the basis in logarithmic rather than exponential scale.

```
# retrieve the flight curve from the spline
m_data_sp <- unique(merge(pheno, x2_spl[, .(trimDAYNO, spline)], by= "trimDAYNO"))
m_data_sp[, spline_nm := exp(spline) * M_SEASON][, spline_nm := spline_nm/sum(m_data_sp$spline_nm)]

# plot scaled spline and GAM
ggplot() +
    geom_line(data= m_data_sp, aes(x = trimDAYNO, y = spline_nm, colour = "Spline"), linewidth = 1.5) +
    geom_line(data= pheno, aes(x = trimDAYNO, y = NM, colour = "Flight Curve")) +
    labs(y = expression(f(trimDAYNO)), x = "DAYNO")
```


0.03
0.02
0.01
0.00
f(trimDAYNO)
0
100
200
300
DAYNO
colour
Flight Curve
Spline

Bivoltine species

Some butterfly species produce more than one generation per year. This mean that the populations will produce successive generations within the monitoring season. This phenomenon will be reflected in the adult count, resulting in bimodal distributions when the two generations are sufficently spaced in time and that the overlap of the flight curve of the different cohort is not to large.

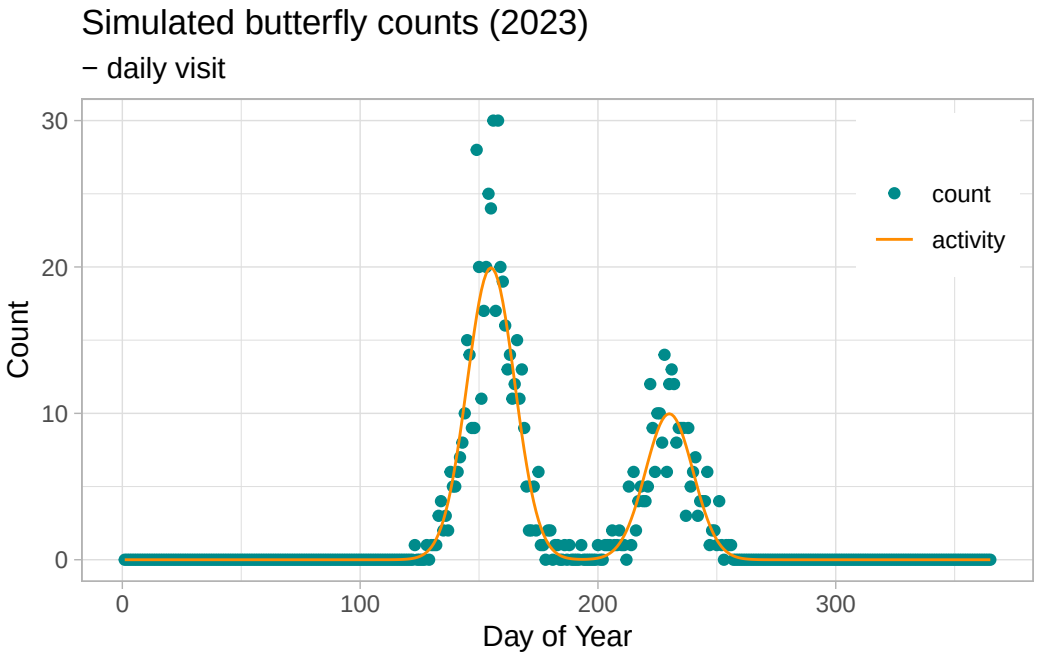
We can simulate bivoltine counts by simply overlapping two generation simulated independently, each with their specific parameters.

```
set.seed(13276)

if(!require("rbms")) devtools::install_github("RetoSchmucki/rbms")
if(!require("butterflyGamSims")) devtools::install_github("cbedwards/butterflyGamSims")
if(!require("mixR")) devtools::install_github("RetoSchmucki/mixR") # small fix in plot function, not

library(rbms)
library(mixR)
library(butterflyGamSims)
library(data.table)
library(ggplot2)

flc_col <- '#ff8c00'
cnt_col <- '#008b8b'
missing_col <- '#8b0000'
GAM_col <- '#483d8b'

## local functions
sim2bms <- function(data, yearKeep = NULL, weeklySample = FALSE, weekdayKeep = NULL, monitoringSeason

                  btfl_ts <- data.table::data.table(data)[, site_id := paste0("site_",sim.id)]
                  if(!is.null(yearKeep)){
                        btfl_ts <- btfl_ts[years %in% yearKeep, ]
                  }
                  btfl_ts[ , date := as.Date(doy, origin = paste0(years, "-01-01"))-1]
                  btfl_ts[ , week := isoweek(date)]
                  btfl_ts[month(date) != 1 | week < 50, weekday := rowid(week), by = .(site_id, years
                  if(isTRUE(weeklySample)){
                        if(!is.null(weekdayKeep)){
                              btfl_ts <- btfl_ts[weekday %in% weekdayKeep, ]
```

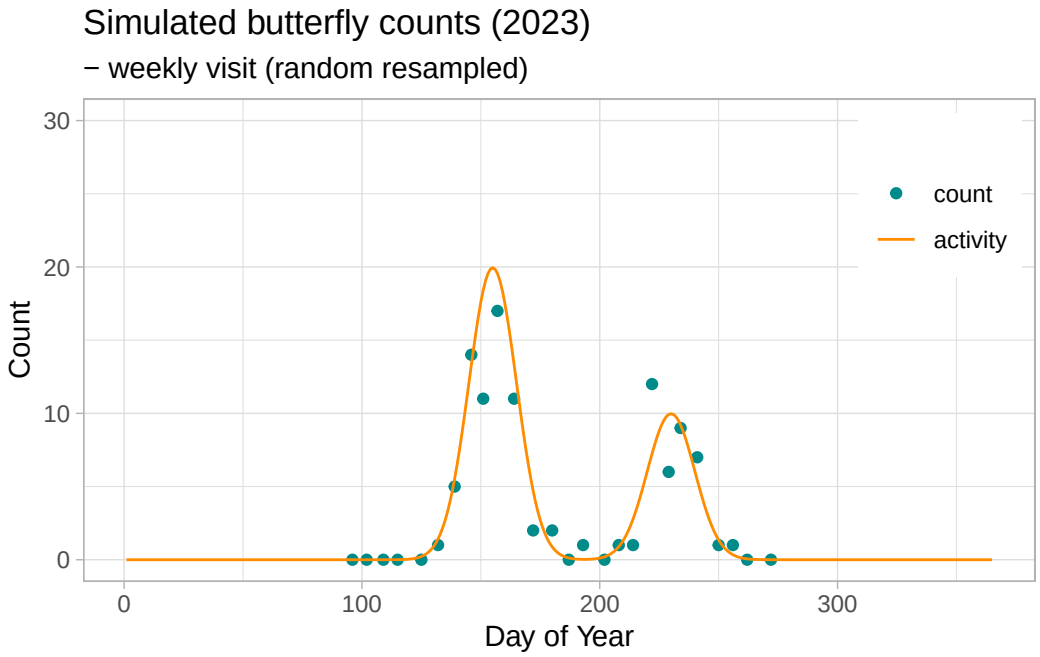
```
                        }
                        btfl_ts <- btfl_ts[btfl_ts[,.I[sample(.N, 1)], by = .(week, site_id, years)][["V
                  }
                  if(!is.null(monitoringSeason)){
                  btfl_ts <- btfl_ts[month(date) %in% monitoringSeason, ]
                  }
            return(btfl_ts)
            }

missing_prob <- function(data, mu=NULL, alpha = 5, theta = 0.3){
                              x_ <- seq_len(nrow(data))
                              mu_ <- ifelse(is.null(mu), length(x_) / 2, mu)
                              std_ <- sqrt(mu_ / theta)
                              y_ <- abs((alpha * exp((-(x_ - mu_)^2) / std_^2)) - alpha) + alpha
                              yn_ <- y_ / (sum(y_))
                        return(yn_)
                  }

sample_missing <- function(data, propMissing = 0.25){

            missing.prob <- data.table::data.table()
            for(i in data[, unique(years)]){
                  for(j in data[, unique(site_id)]){
                  missing.prob <- rbind(missing.prob, missing_prob(data[years == i & site_id == j, ]))
                  }
            }

            missing.week <- data[sample(seq_len(.N), round(propMissing * .N), prob = unlist(missing.prob

      return(missing.week)
}
```

```
size1 <- 500
peak1 <- 155
sd1 <- 10
size2 <- 250
peak2 <- 230
sd2 <- 10
y <- c(2023)

btfl_data1 <- timeseries_sim(nsims=1,
               year = y,
               doy.samples = seq(from=1, to=365, by=1),
               abund.type = "exp",
               activity.type = "gauss",
               sample.type = "pois",
               sim.parms = list(growth.rate = 0,
                                init.size = size1,
                                act.mean = peak1,
```

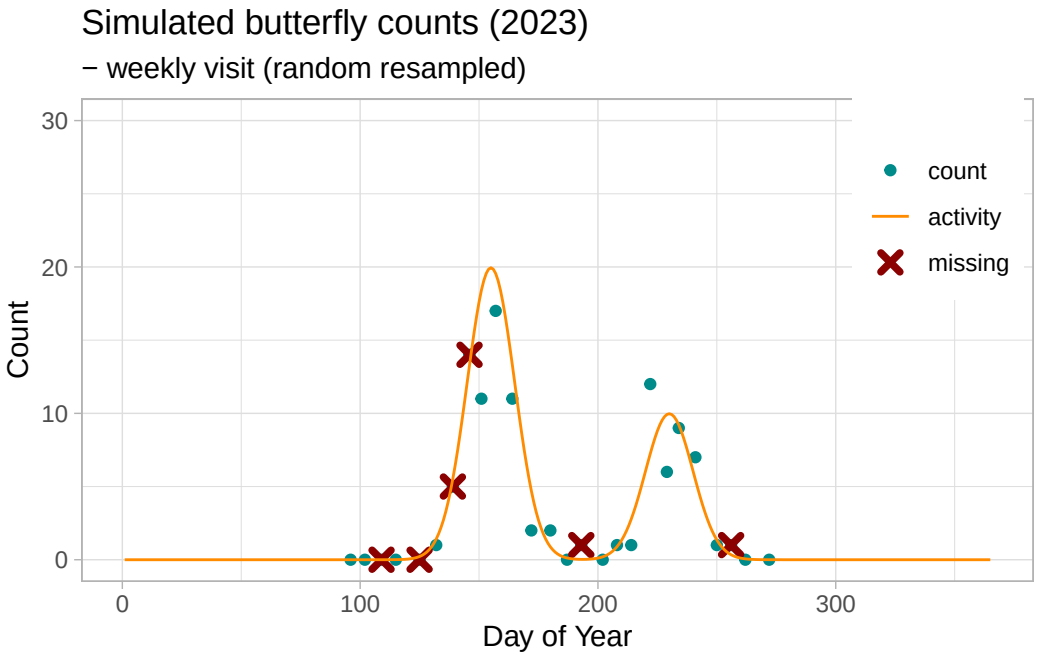
```
                                act.sd = sd1)
                )

btfl_data2 <- timeseries_sim(nsims=1,
                year = y,
                doy.samples = seq(from=1, to=365, by=1),
                abund.type = "exp",
                activity.type = "gauss",
                sample.type = "pois",
                sim.parms = list(growth.rate = 0,
                                init.size = size2,
                                act.mean = peak2,
                                act.sd = sd2)
                )

btfl_data_b2 <- rbind(btfl_data1$timeseries[,c("years", "doy", "count", "act", "sim.id")],
                      btfl_data2$timeseries[,c("years", "doy", "count", "act", "sim.id")])
btfl_data_b2 <- unique(data.table(btfl_data_b2)[, ":="(count=sum(count), act=sum(act)), by = .(years,
btfl_data_b2[, abund.true:= sum(c(size1, size2))]

btfl_ts <- sim2bms(data = btfl_data_b2, yearKeep = y)

btfl_fig1 <- ggplot() +
                geom_point(data=btfl_ts, aes(x=doy, y=count, colour = "count")) +
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
                      breaks = c("count", "activity"),
                      values = c(cnt_col, flc_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- daily visit",
                     x = "Day of Year",
                     y = "Count")
btfl_fig1
```




```
## ============================
## degradation1: weekly count
## ============================

btfl_week_smpl <- sim2bms(data = btfl_data_b2, yearKeep = y,
                  weeklySample = TRUE,
                  weekdayKeep = c(2:5),
                  monitoringSeason = c(4:9))

btfl_fig2 <- ggplot() +
                geom_point(data=btfl_week_smpl, aes(x=doy, y=count, colour = "count")) +
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
                      breaks = c("count", "activity"),
                      values = c(cnt_col, flc_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- weekly visit (random resampled)",
                     x = "Day of Year",
                     y = "Count")

btfl_fig2
```



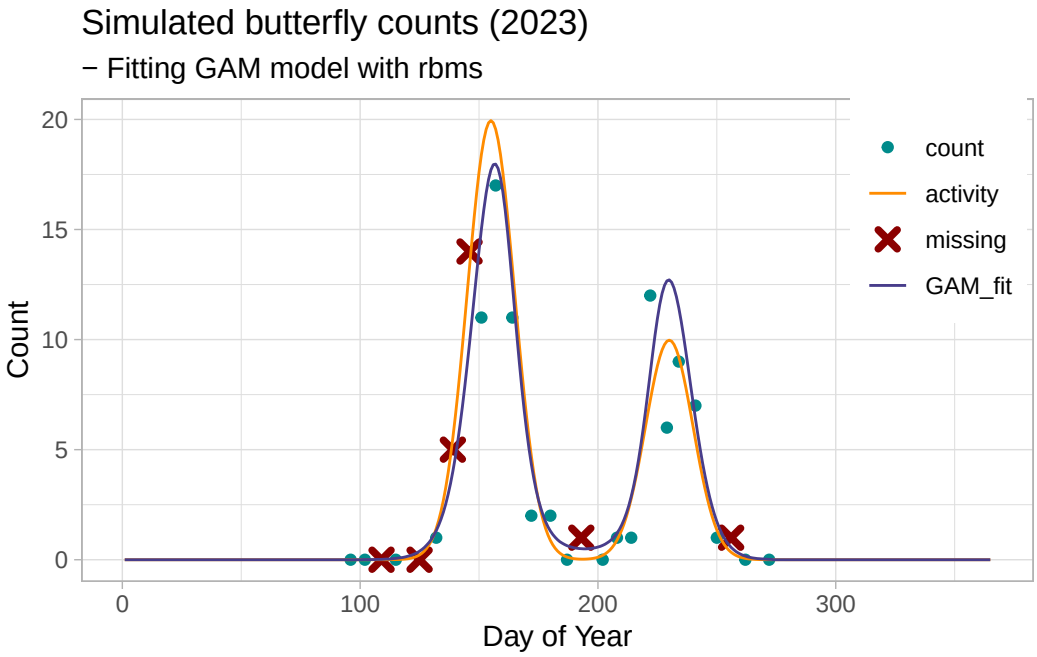

```
## ==============================
## degradation: add missing week
## ==============================

btfl_week_missing <- sample_missing(data = btfl_week_smpl, propMissing = 0.25)

btfl_fig3 <- ggplot() +
                geom_point(data=btfl_week_smpl, aes(x=doy, y=count, colour = "count")) +
                geom_point(data=btfl_week_missing, aes(x=doy, y=count, colour = "missing"),
                           shape=4, size=2, stroke=2) +
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
                      breaks = c("count", "activity", "missing"),
                      values = c(cnt_col, flc_col, missing_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- weekly visit (random resampled)",
                     x = "Day of Year",
                     y = "Count")

btfl_fig3
```



From the weekly counts recorded (simulated) for the bivoltine species, we can use the rbms package to fit a Generalized Additive Model (GAM) to retrieve the overall shape of the annual flith curve. Althouhg we only use one site here, this can be extended and more powerfull if we had more than one site monitored within the region.

```
visit_sim <- btfl_week_smpl[!date %in% btfl_week_missing$date, .(site_id, date, count)]
count_sim <- visit_sim[count>=1,][, species := "sp1"]

names(visit_sim) <- toupper(names(visit_sim))
names(count_sim) <- toupper(names(count_sim))

ts_date <- rbms::ts_dwmy_table(InitYear = 2023, LastYear = 2023, WeekDay1 = 'monday')

ts_season <- rbms::ts_monit_season(ts_date,
                        StartMonth = 4,
                        EndMonth = 9,
                        StartDay = 1,
                        EndDay = NULL,
                        CompltSeason = TRUE,
                        Anchor = TRUE,
                        AnchorLength = 2,
                        AnchorLag = 2,
                        TimeUnit = 'd')

ts_season_visit <- rbms::ts_monit_site(ts_season, visit_sim)

ts_season_count <- rbms::ts_monit_count_site(ts_season_visit, count_sim, sp = "sp1")
```

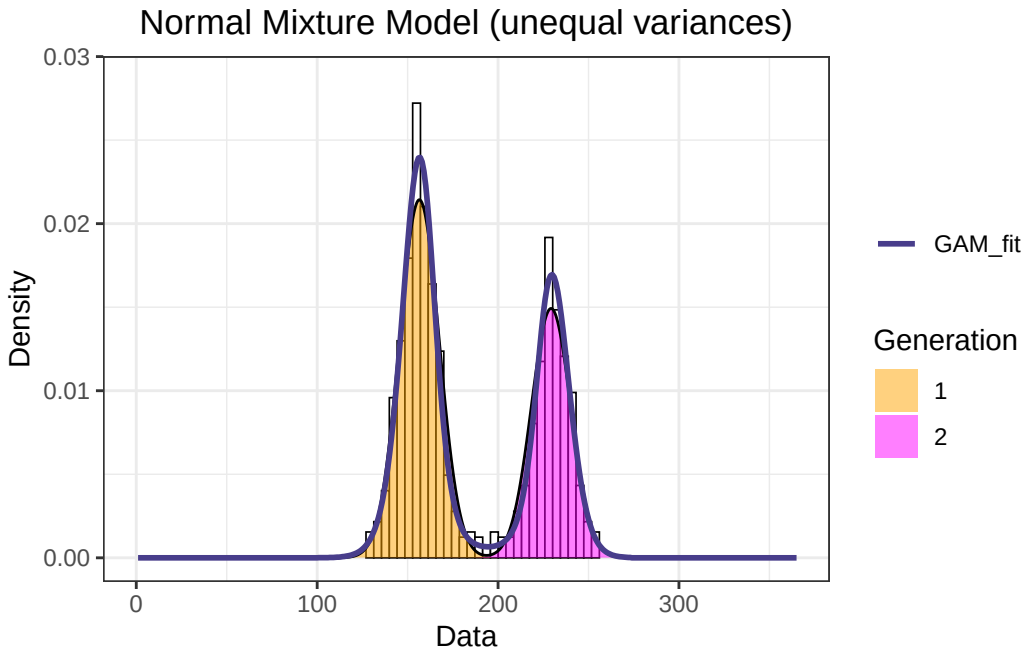
```
### Fitting a GAM to Butterfly Counts

ts_flight_curve <- rbms::flight_curve(ts_season_count,
                        NbrSample = 300,
                        MinVisit = 5,
                        MinOccur = 3,
                        MinNbrSite = 1,
                        MaxTrial = 4,
                        GamFamily = 'nb',
                        SpeedGam = FALSE,
                        CompltSeason = TRUE,
                        SelectYear = NULL,
                        TimeUnit = 'd')


## plot-fitted-flight-curve

pheno <- ts_flight_curve$pheno

btfl_fig4 <- ggplot() +
                geom_point(data=ts_season_count[ANCHOR == 0 & !is.na(COUNT), ], aes(x=DAY_SINCE, y=CO
                geom_point(data=btfl_week_missing, aes(x=doy, y=count, colour = "missing"),
                           shape=4, size=2, stroke=2) +
                geom_line(data = btfl_ts, aes(x = doy, y = act, colour = "activity")) +
                geom_line(data = pheno,
                    aes(x = trimDAYNO, y = btfl_ts[, unique(abund.true)]*NM, colour = "GAM_fit")) +
                xlim(1,365) + ylim(0, max(btfl_ts$act,
                                          pheno$NM*btfl_ts[,unique(abund.true)],
                                          btfl_week_missing$count,
                                          ts_season_count[!is.na(COUNT), COUNT] )) +
                scale_colour_manual("",
                      breaks = c("count", "activity", "missing", "GAM_fit"),
                      values = c(cnt_col, flc_col, missing_col, GAM_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- Fitting GAM model with rbms",
                     x = "Day of Year",
                     y = "Count")
btfl_fig4
```



Retrieve Generation Parameters

The GAM fitted very well the cummulative fligth curve (e.i., two generations activity curves). While the GAM returns the overall shape resulting from all observation the model does not distinguish the different generations. We can however fit a mixture model with k components (generations) to estimate and retrieve the parameters of the different generations. Here we will simply try to fit two generation with parameter of a Gaussian distribution (mean and standard deviation). The R package mixR provides an efficient algorithm (C++) to fit and assess the goodness of fit of such model(Yu 2022) .

```
set.seed(102)
btnbr <- pheno$NM*btfl_ts[,unique(abund.true)]
x1 <- rep(1:365, round(btnbr))

# fit a Normal mixture model (unequal variances)
mod1 = mixfit(x1, ncomp = 2)
gen_plot1 <- plot(mod1, title = 'Normal Mixture Model (unequal variances)')

gen_plot1 +
      xlim(0, 365) +
      scale_fill_manual("Generation", values=c("orange","magenta")) +
      geom_line(data = pheno, aes(x = trimDAYNO, y = NM, colour = "GAM_fit"), lwd = 1) +
      scale_colour_manual("", breaks =  "GAM_fit", values = GAM_col)
```




```
mod1
```

```
Normal mixture model with 2 components
          comp1       comp2
pi    0.5909181   0.4090819
mu  156.4979230 229.3178504
sd   11.0005338  10.9353467

EM iterations: 7 AIC: 6756.14 BIC: 6779.26 log-likelihood: -3373.07
```

Parameter	simulation	Estimated (mixture)
generation 1 relative size	0.6666667	0.59
generation 1 peak	155	156
generation 1 sd	10	11
generation 2 relative size	0.3333333	0.41
generation 2 peak	230	229
generation 2 sd	10	10.94

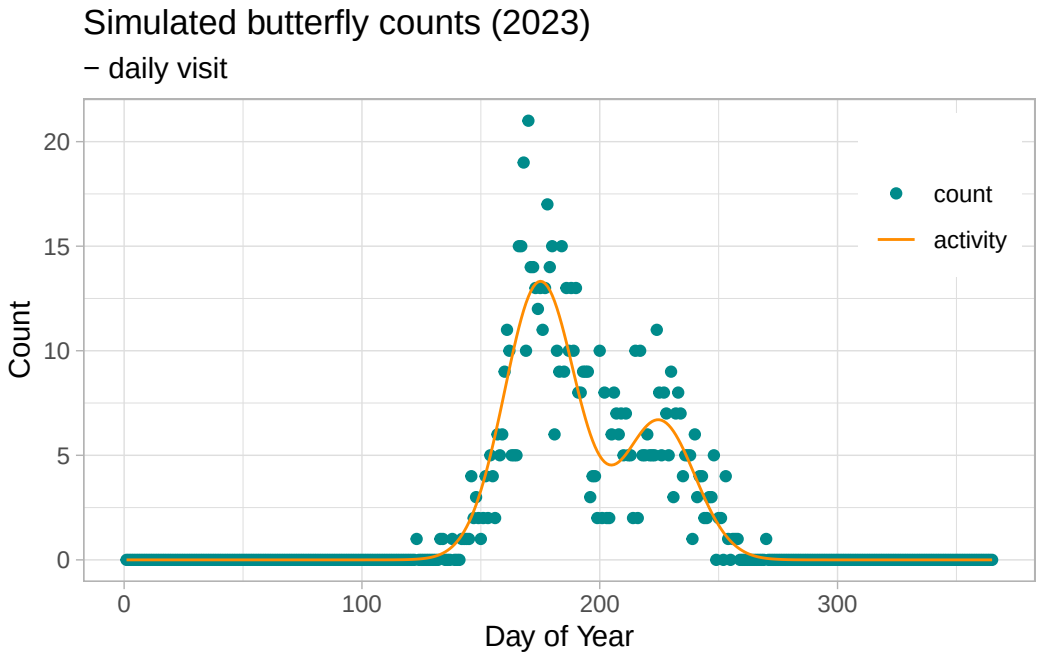
The mixture model converged and indicate that the first generation is 0.59 percent and the second 0.41 percent, in other words, generation one has 1.44 the number of individuals of generation two. The peak of the first generation is located at day 156 and the second at day 229, with standard deviation of 11 and 10.94 respectively.

These results compare nicely with the parameters used in our simulation where the first generation had 500 individual, with a peak at day 155 and a standard deviation of 10, and the second generation had 250 individuals with a peak at day 230 and a standard deviation of 10.

Overlapping generations

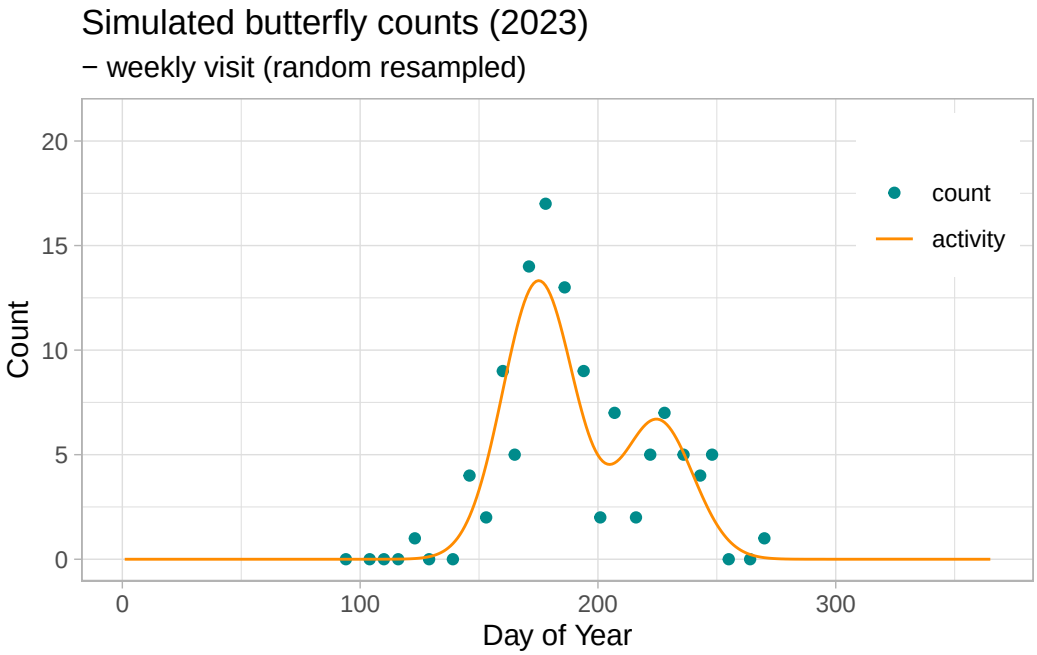
```
set.seed(13276)

size1 <- 500
peak1 <- 175
sd1 <- 15
size2 <- 250
peak2 <- 225
sd2 <- 15


btfl_data1 <- timeseries_sim(nsims=1,
                year = y,
                doy.samples = seq(from=1, to=365, by=1),
                abund.type = "exp",
                activity.type = "gauss",
                sample.type = "pois",
                sim.parms = list(growth.rate = 0,
                                 init.size = size1,
                                 act.mean = peak1,
                                 act.sd = sd1)
                )

btfl_data2 <- timeseries_sim(nsims=1,
                year = y,
                doy.samples = seq(from=1, to=365, by=1),
                abund.type = "exp",
                activity.type = "gauss",
                sample.type = "pois",
                sim.parms = list(growth.rate = 0,
                                 init.size = size2,
                                 act.mean = peak2,
                                 act.sd = sd2)
                )

btfl_data_b2 <- rbind(btfl_data1$timeseries[,c("years", "doy", "count", "act", "sim.id")],
                      btfl_data2$timeseries[,c("years", "doy", "count", "act", "sim.id")])
btfl_data_b2 <- unique(data.table(btfl_data_b2)[, ":="(count=sum(count), act=sum(act)), by = .(years, do
btfl_data_b2[, abund.true:=sum(unique(c(btfl_data1$timeseries$abund.true, btfl_data2$timeseries$abund.tr

btfl_ts <- sim2bms(data = btfl_data_b2, yearKeep = y)

btfl_fig1 <- ggplot() +
                geom_point(data=btfl_ts, aes(x=doy, y=count, colour = "count")) +
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
```

```
                      breaks = c("count", "activity"),
                      values = c(cnt_col, flc_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- daily visit",
                     x = "Day of Year",
                     y = "Count")

btfl_fig1
```



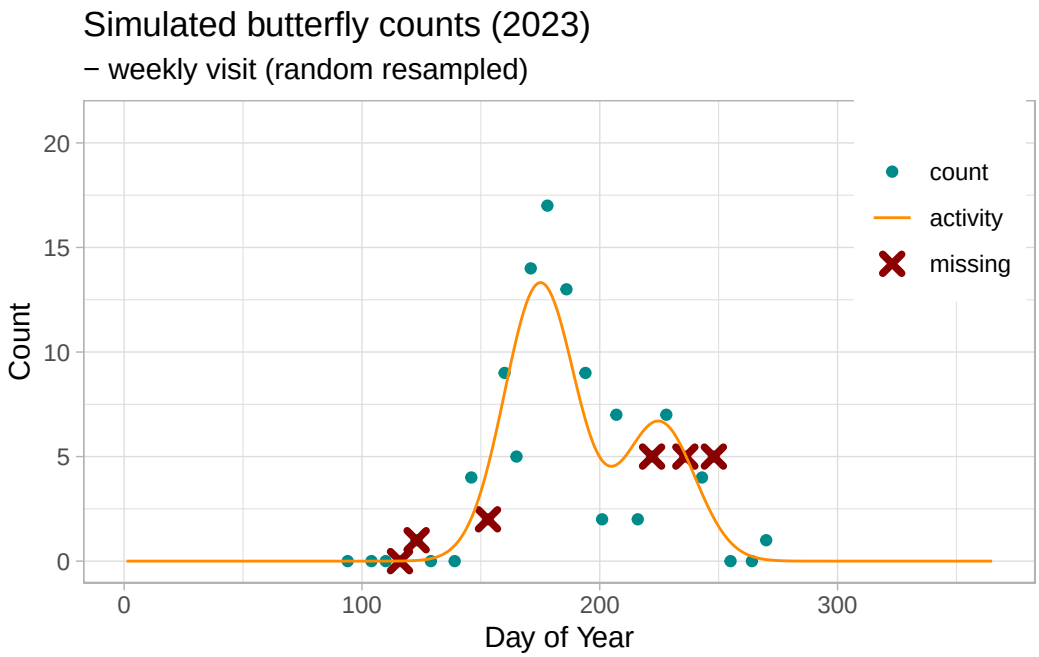

```
btfl_week_smpl <- sim2bms(data = btfl_data_b2, yearKeep = y,
                  weeklySample = TRUE,
                  weekdayKeep = c(2:5),
                  monitoringSeason = c(4:9))

btfl_fig2 <- ggplot() +
                geom_point(data=btfl_week_smpl, aes(x=doy, y=count, colour = "count")) +
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
                      breaks = c("count", "activity"),
                      values = c(cnt_col, flc_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
```

```
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- weekly visit (random resampled)",
                     x = "Day of Year",
                     y = "Count")


btfl_fig2
```




```
btfl_week_missing <- sample_missing(data = btfl_week_smpl, propMissing = 0.25)

btfl_fig3 <- ggplot() +
                geom_point(data=btfl_week_smpl, aes(x=doy, y=count, colour = "count")) +
                geom_point(data=btfl_week_missing, aes(x=doy, y=count, colour = "missing"),
                           shape=4, size=2, stroke=2) +
                geom_line(data = btfl_ts,
                aes(x = doy, y = act, colour = "activity")) +
                xlim(1,365) + ylim(0, max(btfl_ts$count, btfl_ts$act)) +
                scale_colour_manual("",
                      breaks = c("count", "activity", "missing"),
                      values = c(cnt_col, flc_col, missing_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- weekly visit (random resampled)",
                     x = "Day of Year",
                     y = "Count")
```

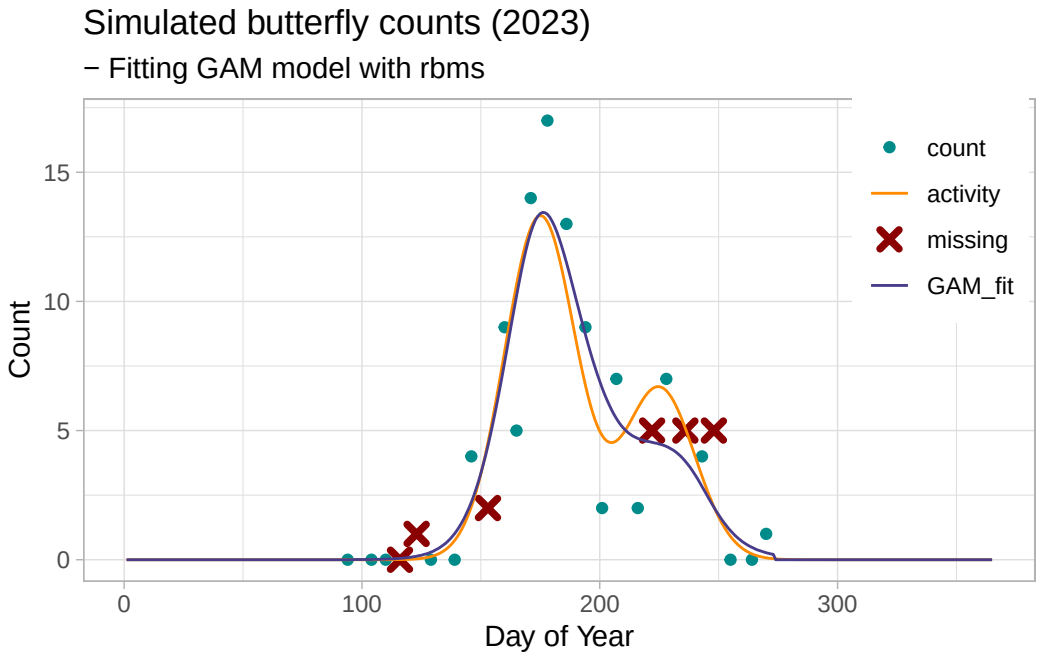
```
btfl_fig3
```



Fitting the annual flith curve.

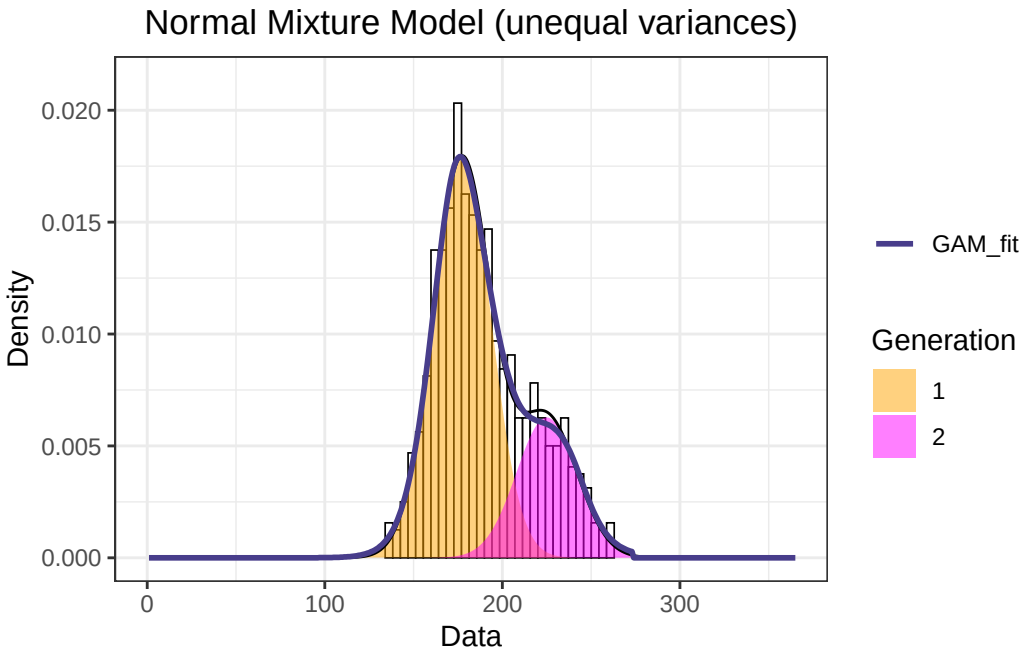
```
visit_sim <- btfl_week_smpl[!date %in% btfl_week_missing$date, .(site_id, date, count)]
count_sim <- visit_sim[count>=1,][, species := "sp1"]

names(visit_sim) <- toupper(names(visit_sim))
names(count_sim) <- toupper(names(count_sim))

ts_date <- rbms::ts_dwmy_table(InitYear = 2023, LastYear = 2023, WeekDay1 = 'monday')

ts_season <- rbms::ts_monit_season(ts_date,
                        StartMonth = 4,
                        EndMonth = 9,
                        StartDay = 1,
                        EndDay = NULL,
                        CompltSeason = TRUE,
                        Anchor = TRUE,
                        AnchorLength = 2,
                        AnchorLag = 2,
                        TimeUnit = 'd')

ts_season_visit <- rbms::ts_monit_site(ts_season, visit_sim)

ts_season_count <- rbms::ts_monit_count_site(ts_season_visit, count_sim, sp = "sp1")
```

```
### Fitting a GAM to Butterfly Counts

ts_flight_curve <- rbms::flight_curve(ts_season_count,
                        NbrSample = 300,
                        MinVisit = 5,
                        MinOccur = 3,
                        MinNbrSite = 1,
                        MaxTrial = 4,
                        GamFamily = 'nb',
                        SpeedGam = FALSE,
                        CompltSeason = TRUE,
                        SelectYear = NULL,
                        TimeUnit = 'd')

## plot-fitted-flight-curve

pheno <- ts_flight_curve$pheno

btfl_fig4 <- ggplot() +
                geom_point(data=ts_season_count[ANCHOR == 0 & !is.na(COUNT), ], aes(x=DAY_SINCE, y=COUNT
                geom_point(data=btfl_week_missing, aes(x=doy, y=count, colour = "missing"),
                           shape=4, size=2, stroke=2) +
                geom_line(data = btfl_ts, aes(x = doy, y = act, colour = "activity")) +
                geom_line(data = pheno,
                    aes(x = trimDAYNO, y = btfl_ts[, unique(abund.true)]*NM, colour = "GAM_fit")) +
                xlim(1,365) + ylim(0, max(btfl_ts$act,
                                          pheno$NM*btfl_ts[,unique(abund.true)],
                                          btfl_week_missing$count,
                                          ts_season_count[!is.na(COUNT), COUNT] )) +
                scale_colour_manual("",
                      breaks = c("count", "activity", "missing", "GAM_fit"),
                      values = c(cnt_col, flc_col, missing_col, GAM_col)) +
                theme_light() +
                theme(legend.position = "inside", legend.position.inside = c(0.9, 0.8)) +
                labs(title = paste0("Simulated butterfly counts (", y,")"),
                     subtitle = "- Fitting GAM model with rbms",
                     x = "Day of Year",
                     y = "Count")
btfl_fig4
```



We can see that the GAM fitted a cummulative fligth curve (e.i., the two generations activity curves) without distinguishing the two generations very well. We can try to retrieve the generation by fitting a mixture of k components (generations) defined by a Gaussian distribution.

```
set.seed(102)
btnbr <- pheno$NM*btfl_ts[,unique(abund.true)]
x1 <- rep(1:365, round(btnbr))
# fit a Normal mixture model (unequal variances)
mod1 = mixfit(x1, ncomp = 2)
gen_plot2 <- plot(mod1, title = 'Normal Mixture Model (unequal variances)')

gen_plot2 +
      xlim(0, 365) +
      scale_fill_manual("Generation", values=c("orange","magenta")) +
      geom_line(data = pheno, aes(x = trimDAYNO, y = NM, colour = "GAM_fit"), lwd = 1) +
      scale_colour_manual("", breaks =  "GAM_fit", values = GAM_col)
```




```
mod1
```

```
Normal mixture model with 2 components
         comp1       comp2
pi   0.7312038   0.2687962
mu 177.1682929 224.9959818
sd  16.3513803  17.0979745

EM iterations: 141 AIC: 6920.48 BIC: 6943.54 log-likelihood: -3455.24
```

Parameter	simulation	Estimated (mixture)
generation 1 relative size	0.6666667	0.73
generation 1 peak	175	177
generation 1 sd	15	16.35
generation 2 relative size	0.3333333	0.27
generation 2 peak	225	225
generation 2 sd	15	17.1

The mixture model converged and indicate that the first generation is 0.7312038 percent and the second 0.2687962 percent, in other words, generation one has 2.7202908 the number of individuals of generation two. The peak of the first generation is located at day 177.1682929 and the second at day 224.9959818, with standard deviation of 16.3513803 and 17.0979745 respectively.

Part III

Butterfly Trends

Trends Simulation

> Content under construction

Multisite and trends

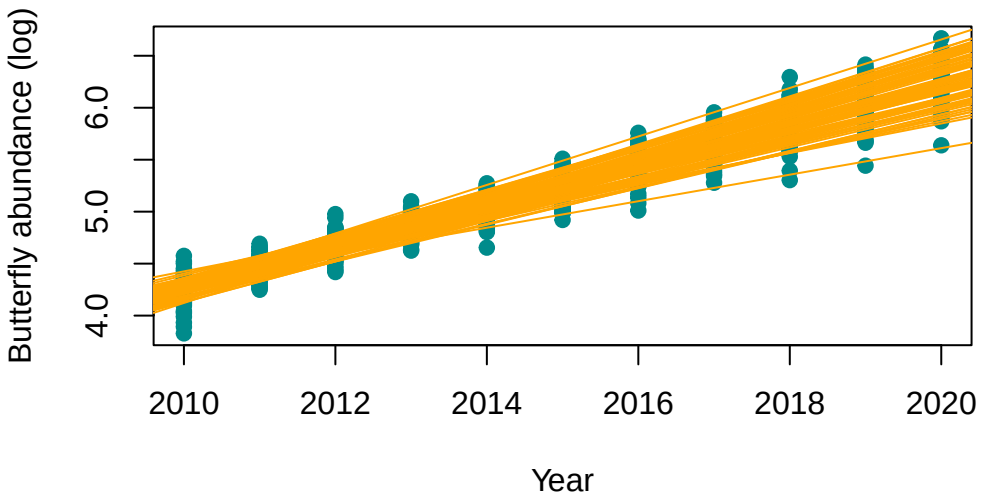
Here we will simulate butterfly count data for multiple sites (100), for a univoltine species that has a simple Gaussian flight curve (peak at day 175), where each population has an initial number of 500 individuals (total number of observations accumulated over the season) and a mean population growth rate of 20% over 10 years (2010 to 2020). We introduce some variation (standard deviation of 0.02) in the growth rate observed across the 100 sites.

For this simulation, we will use the `timeseries_sim()` function from the butterflyGamSims package and the `sim2bms()` function written above to reformat the simulated data into a format that can be used in the rbms package.

```
set.seed(13276)

if(!require("butterflyGamSims")) devtools::install_github("cbedwards/butterflyGamSims")

trend_sim <- data.table()

for(i in 1:100){

      btfl_data <- timeseries_sim(nsims=1,
                  year = c(2010:2020),
                  doy.samples = seq(from=1, to=365, by=1),
                  abund.type = "exp",
                  activity.type = "gauss",
                  sample.type = "pois",
                  sim.parms = list(growth.rate = rnorm(1, mean = 0.2, sd=0.02),
                                   init.size = 500,
                                   act.mean = 175,
                                   act.sd = 15)
                  )

      trend_sim <- rbind(trend_sim,
```

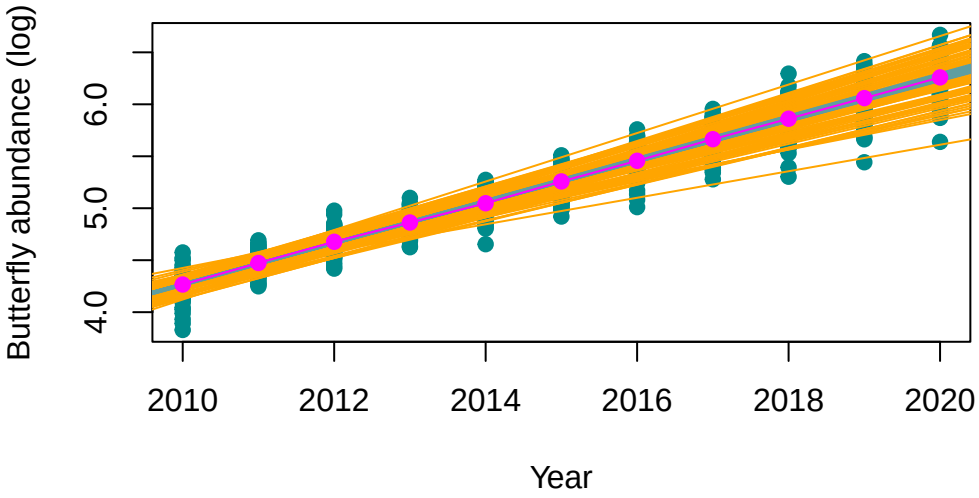
```
                        sim2bms(data = btfl_data$timeseries,
                               weeklySample = TRUE,
                               weekdayKeep = c(2:5),
                               monitoringSeason = c(4:9))[,
                               growth.rate := btfl_data$parms$growth.rate][,
                               site_id := paste0("site_id", i)])

}
knitr::kable(head(trend_sim))
```

```
## plot and add lm

trend_sim[, y_count := sum(count), by = .(site_id, years)]
plot(unique(trend_sim[, .(years, log(y_count))]), pch = 19, col = "darkcyan",
      ylab = "Butterfly abundance (log)", xlab = "Year")

for(i in 1:100){
    abline(lm(log(y_count) ~ years, data = trend_sim[site_id == paste0("site_id", i), .(years, y_count)]
           col = 'orange')
}
```



The overall trend can be calculated by fitting a Generalized Linear Model (GLM), with a Poisson error distribution on the simulated dataset. Note that here we did not include the site effect as all sites had the same abundance (500 individuals) and we excluded the intercept from our model. To illustrate the effect of the variation in the trends across sites on our overall trend estimate, we can bootstrap (resample with replacement) the sites and fit the GLM on the bootstrap sample. We can generate 100 (or more) iterations to illustrate the variation around the trend derived from our 100 sites.

```
plot(unique(trend_sim[, .(years, log(y_count))]), pch = 19, col = "darkcyan",
      ylab = "Butterfly abundance (log)", xlab = "Year")

for(i in 1:100){
    abline(lm(log(y_count) ~ years, data = trend_sim[site_id == paste0("site_id", i), .(years, y_coun
           col = 'orange')
}

for(i in seq_len(100)){

boot_site <- sample(unique(trend_sim$site_id), trend_sim[, uniqueN(site_id)], replace = TRUE)

#points(c(2010:2020), as.numeric(coef(glm(y_count~ factor(years) - 1, data = trend_sim[site_id %in% b
#                                         family = "poisson"))), col = 'magenta', pch = 19)

abline(lm(as.numeric(coef(glm(y_count~ factor(years) - 1, data = trend_sim[site_id %in% boot_site, ],
                              family = "poisson"))) ~ c(2010:2020)), col = 'cadetblue', type = 'l')
}

points(c(2010:2020), as.numeric(coef(glm(y_count~ factor(years) - 1, data = trend_sim, family = "pois
      col = 'magenta', pch = 19)
points(c(2010:2020), as.numeric(coef(glm(y_count~ factor(years) - 1, data = trend_sim, family = "pois
      col = 'magenta', type = 'l')
```



Part IV

Bibliography

References

Edwards, Collin, Cheryl Schultz, David Sinclair, Daniel Marschalek, and Elizabeth Crone. 2023. "Estimating Butterfly Population Trends from Sparse Monitoring Data Using Generalized Additive Models." December 8, 2023. https://doi.org/10.1101/2023.12.07.570644.

Schmucki, Reto, Colin Harrower A., and Emily Dennis B. 2022. "rbms: Computing generalised abundance indices for butterfly monitoring count data." https://github.com/RetoSchmucki/rbms.

Yu, Youjiao. 2022. "mixR: An r Package for Finite Mixture Modeling for Both Raw and Binned Data." *Journal of Open Source Software* 7 (69): 4031. https://doi.org/10.21105/joss.04031.