virtualPollen

Generation of virtual pollen curves

Blas M. Benito

Contents

Summary

This document describes in detail the methods used to generate a virtual environmental driver with a given temporal autocorrelation, to be used as an input for a population model simulating synthetic pollen curves generated by virtual taxa with different life-traits (life-span and fecundity) and environmental niche features (niche position and breadth). We also describe how we generated a virtual sediment accumulation rate to aggregate the results of the population model to mimic taphonomic processes producing real pollen curves, and how we resampled virtual pollen data at different depth intervals. Finally, we present the code used to generate the 16 virtual taxa used for the analyses described in the paper.

IMPORTANT: An Rmarkdown version of this document can be found at: https://github.com/BlasBenito/EcologicalMemory.

1 Generating a virtual environmental driver

1.0.1 Rationale

To simulate virtual pollen curves with the population model described in section **2** a virtual driver representing changes in environmental conditions is required. This section explains how to generate such a driver as a time series with a given temporal autocorrelation simulating the temporal structure generally shown by observed environmental drivers.

1.0.2 Generating a random walk

The following steps are required to generate a **random walk** in R:

1. Generate a vector *time* with time values.
2. Re-sample with replacement the set of numbers (-1, 0, 1) as many times as values are in the *time* vector, to produce the vector *moves* (as in *moves of a random walk*).
3. Compute the cumulative sum of *moves*, which generates the random walk.

```r
#sets a state for the generator of pseudo-random numbers
set.seed(1)
#defines the variable "time"
time <- 1:10000
#samples (-1, 0, 1) with replacement
moves <- sample(x=c(-1, 0, 1), size=length(time), replace=TRUE)
#computes the cumulative sum of moves
random.walk <- cumsum(moves)
```

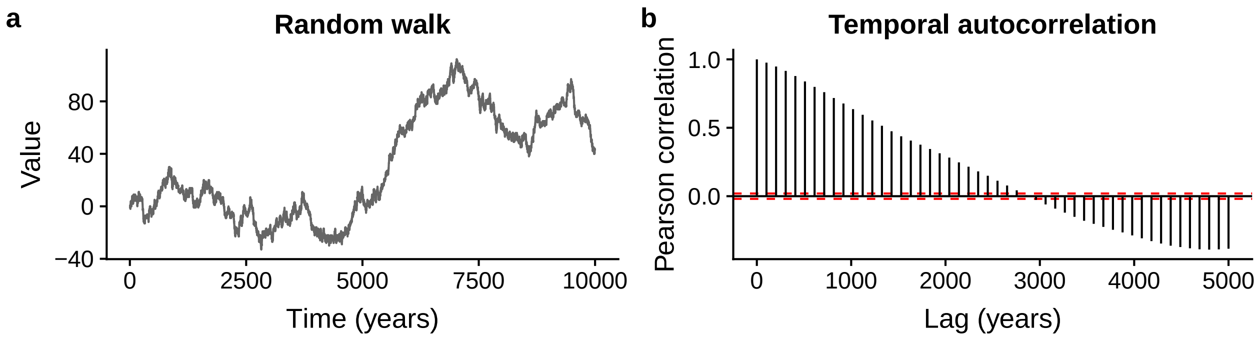


Figure 1: Random walk (a) and its temporal autocorrelation (b).

Every time this code is executed with a different number in *set.seed()*, it generates a different random walk with a different temporal autocorrelation, but still, this simple method is not useful to generate a time series with a given temporal autocorrelation.

1.0.3 Applying a convolution filter to generate a given temporal autocorrelation

Applying a **convolution** filter to a time series allows to generate dependence among sets of consecutive cases. Below we show an example of how it works on a random sequence *a* composed by five numbers in the range [0, 1]. The operations to compute the filtered sequence are shown in **Table 1**.

```r
#setting a fixed seed for the generator of pseudo-random numbers
set.seed(1)

#generating 5 random numbers in the range [0, 1]
a <- runif(5)

#applying a convolution filter of length 2 and value 1
b <- filter(a, filter=c(1,1), method=”convolution”, circular=TRUE)
```

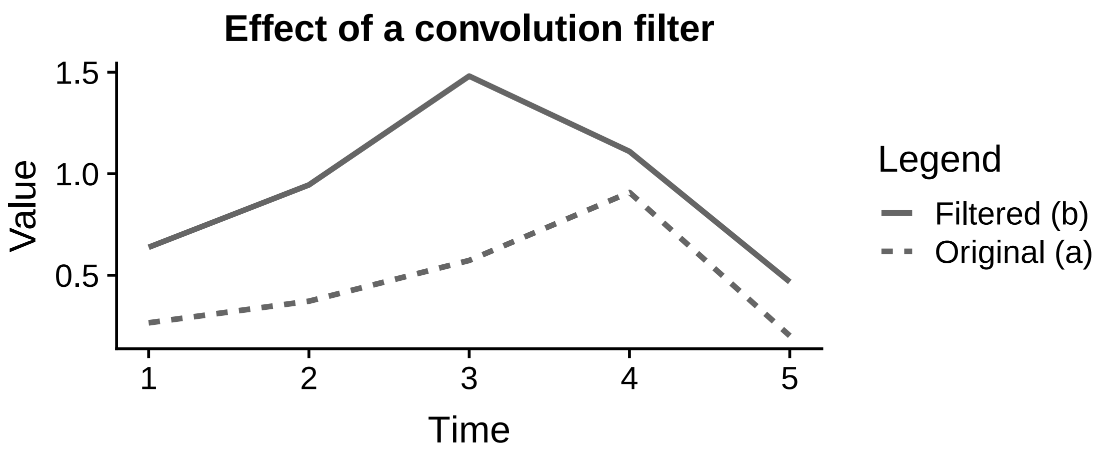


Figure 2: Original sequence (dashed line) and filtered sequence with filter (solid line).

The *operation* column in **Table 1** shows how the convolution filter generates a dependence between values located within the length of the filter. This positional dependence creates a temporal autocorrelation pattern with a significant length equal to the length of the filter. The following piece of code demonstrates this by generating two versions of the same *moves* vector used before, one with a length

Table 1: Original sequence (a), filtered sequence (b), and filtering operations. Numbers beside letters a and b represent row numbers, while f1 and f2 represent the values of the convolution filter (both equal to 1 in this case).

row	a	b	operation
1	0.27	0.64	b1 = a1 x f2 + a2 x f1
2	0.37	0.94	b2 = a2 x f2 + a3 x f1
3	0.57	1.48	b3 = a3 x f2 + a4 x f1
4	0.91	1.11	b4 = a4 x f2 + a5 x f1
5	0.20	0.47	b5 = a5 x f2 + a6 x f1

of the significant temporal autocorrelation equal to 10 (defined in the same units of the *time* vector), and another with a length of 100. Results are shown in **Figure 3**.

```r
moves.10 <- filter(moves, filter=rep(1, 10), circular=TRUE)
moves.100 <- filter(moves, filter=rep(1, 100), circular=TRUE)
```

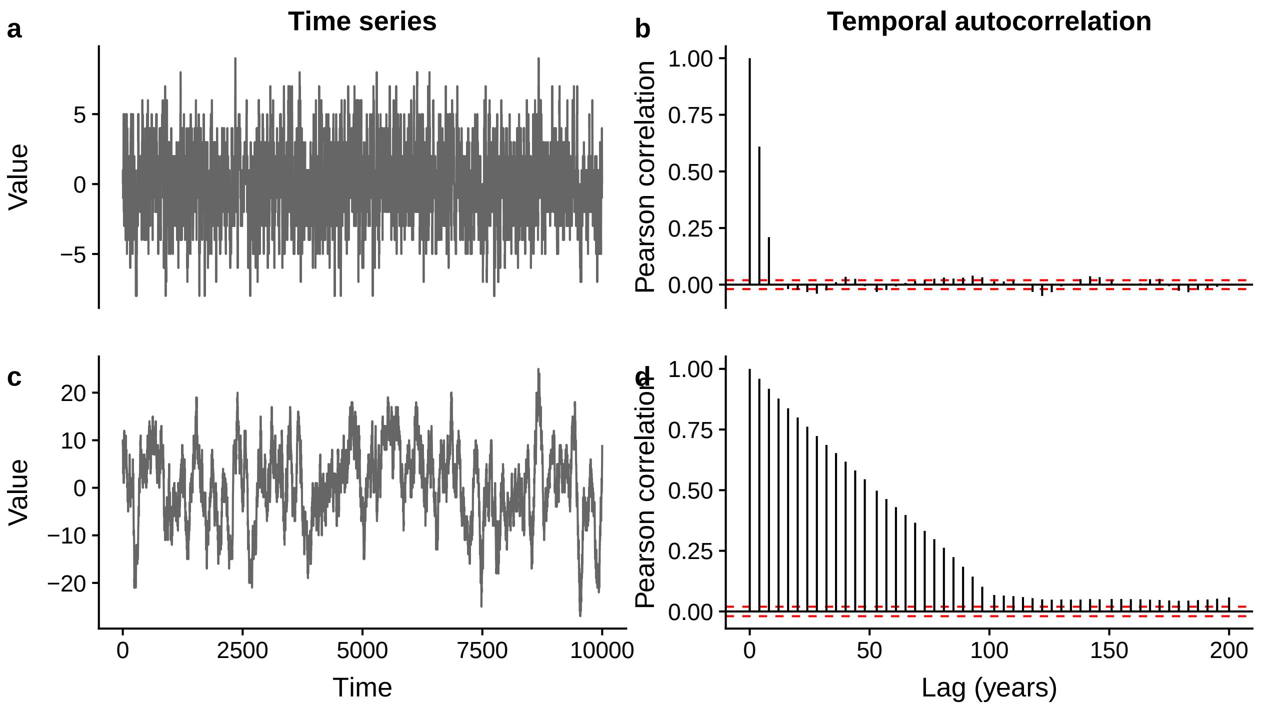


Figure 3: Sequences filtered with different filter lengths: a) Sequence with autocorrelation length equal to 10; b) Temporal autocorrelation of a); c) Sequence with autocorrelation length equal to 100: d) Temporal autocorrelation of c).

A major limitation is that this method does not work well when the required length of the temporal autocorrelation is a large fraction of the overall length of the time series. The code below and **Figure 4** show an example with a filter of length 5000 (note that the *time* variable has 10000 elements).

```
moves.5000 <- filter(moves, filter=rep(1, 5000), circular=TRUE)
```

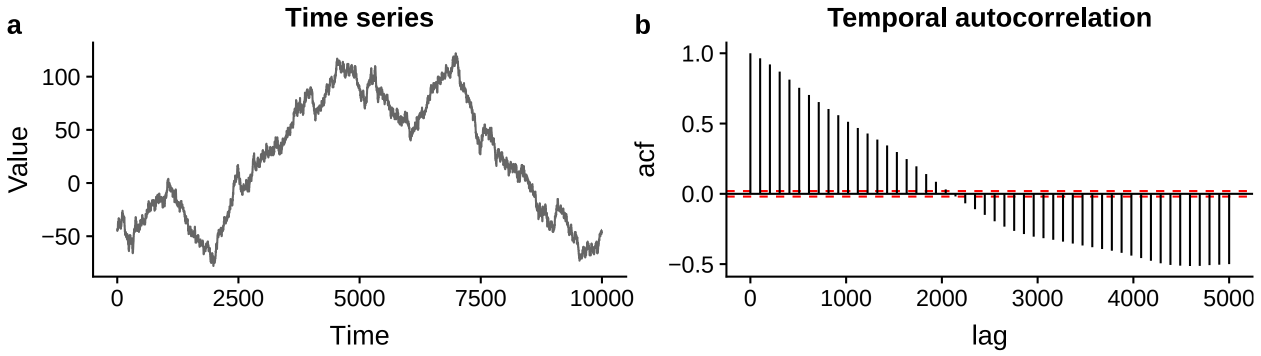


Figure 4: Sequence moves.5000 (a) and its temporal autocorrelation (b). In this case there is a large deviation between the required temporal autocorrelation (5000) and the outcome (2000).

All the ideas presented above are implemented in a function named **simulateDriver()**, with the following arguments:

- **random.seed**: an integer for *set.seed()*, to use the same random seed in the generation of random numbers and make sure that results are reproducible.
- **age**: a vector (i.e. 1:1000), representing the length of the output.
- **autocorrelation.length**: length of the desired temporal autocorrelation structure, in the same units as *age*.
- **output.min**: minimum value of the driver.
- **output.max**: maximum value of the driver.

```
simulateDriver <- function(random.seed,
                           time,
                           autocorrelation.length,
                           output.min,
                           output.max){

  #setting random seed
  set.seed(random.seed)

  #generating driver
  driver = filter(rnorm(max(time)),
                        filter=rep(1, autocorrelation.length),
                        circular=TRUE)

  #rescaling it between output.min and output.max
  driver = as.vector(((driver - min(driver)) /
                           (max(driver) - min(driver))) *
                          (output.max - output.min) +
                          output.min)

  return(driver)
}
```

we used this function to generate a driver with annual resolution in the range [0, 100], over a period of 10000 years (see **Figure 5**), with a temporal autocorrelation length of 600 years.

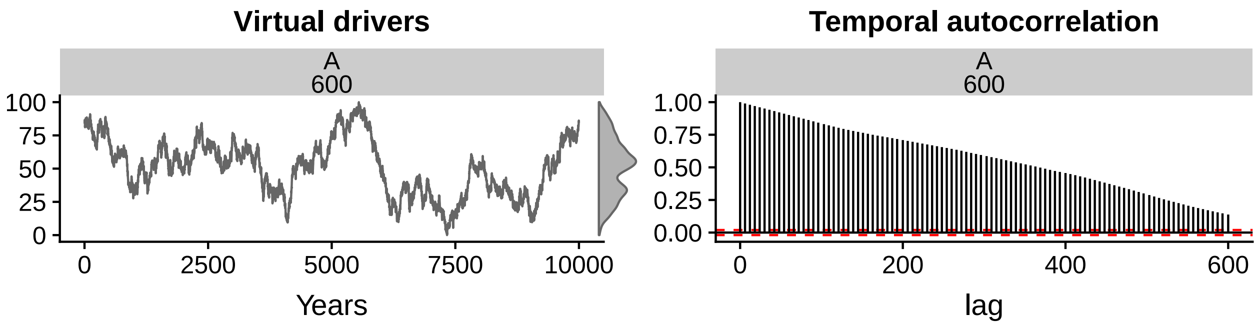


Figure 5: Virtual driver used for the simulations in the paper.

2 Simulating pollen curves from virtual taxa

2.0.1 Rationale

The ability of plant populations to respond more or less quickly to environmental change is mediated by a set of species' traits, such as niche optimum and breadth, growth rate, sexual maturity age, effective fecundity, and life span. Even though we have values for these traits for many plant species, pollen types are often clusters of species of the same genus or family rather than single species, making the assignment of trait values uncertain. For the purpose of this paper it is more practical to work with **virtual pollen curves** generated by **virtual taxa** with known relationships with the environment and traits. These virtual taxa are the result of a population model with different parametrizations.

2.0.2 Assumptions

The model follows these assumptions:

- **The spatial structure of the population is not important to explain its pollen productivity**. This is an operative assumption, to speed-up model executions. The lack of spatial structure is partially compensated by the model parameter **charge capacity**, which simulates a limited space for population expansion.
- **The environmental niche of the species follows a Gaussian distribution**, characterized by a mean (niche optimum, also niche position) and a standard deviation (niche breadth or tolerance).
- **Different drivers can have a different influence on the species dynamics**, and that influence can be defined by the user by tuning the weights of each driver.
- **Environmental suitability**, expressed in the range [0, 1], is the result of an additive function of the species niches (normal function defined by the species' mean and standard deviation for each driver), the drivers' values, and the relative influence of each driver (*driver weights*).
- **Pollen productivity is a function of the individual's biomass and environmental suitability**, so under a hypothetical constant individual's biomass, its pollen production depends linearly on environmental suitability values.
- **Effective fecundity is limited by environmental suitability**. Low environmental suitability values limit recruitment, acting as an environmental filter. Therefore, even though the fecundity of the individuals is fixed by the **fecundity** parameter, the overall population fecundity is limited by environmental suitability.

2.0.3 Parameters

We have designed a simple individual-based population model which input parameters are:

- **Drivers**: Values of up to two drivers (provided as numeric vectors) for a given time span.
- **Niche mean**: The average (in drivers units) of the normal functions describing the species niche for each driver. This parameter defines the niche optimum of the species.
- **Niche breadth**: Standard deviations (in driver units) of the normal functions describing the species niche for each driver. Smaller values simulate more specialized species, while larger values simulate generalist species.
- **Driver weight**: Importance of each driver (note that in the paper we only used one driver) in defining climatic suitability for the given species. Balanced weights mean each driver is contributing equally to climatic suitability. The sum of both drivers must be 1.
- **Maximum age**: age (in years) of senescence of the individuals. Individuals die when reaching this age.

- **Reproductive age**: age (in years) at which individuals start to produce pollen and seeds.
- **Fecundity**: actually, **effective fecundity**, which is the maximum number of viable seeds produced by mature individuals per year under under fully suitable climatic conditions.
- **Growth rate**: amount of the maximum biomass gained per year by each individual. Used as input in the logistic equation of the growth model.
- **Maximum biomass**: maximum biomass (in relative units) individuals can reach. Used as input in the logistic equation of the growth model to set a maximum growth.
- **Carrying capacity**: maximum sum of the biomass of all individuals allowed in the model. Ideally, to keep model performance, it should be equal to **maximum biomass** multiplied by 100 or 1000. The model removes individuals at random when this number is reached is reached.

In order to explain the model dynamics in the most simplified way, the parameters in **Table 1** are considered.

Table 2: Parameters of a virtual species.

	Parameter	Species 1
1	maximum.age	50
2	reproductive.age	20
3	fecundity	2
4	growth.rate	0.2
5	pollen.control	0
6	maximum.biomass	100
7	carrying.capacity	10000
8	driver.A.weight	1
10	niche.A.mean	50
11	niche.A.sd	10

2.0.4 Ecological niche and environmental suitability

The model assumes that normal distributions can be used as simple mathematical representations of ecological niches. Considering a virtual species and an environmental predictor, the abundance of the species along the range of the predictor values can be defined by a normal distribution with a mean (niche optimum or niche position), and standard deviation (niche breadth or tolerance).

The equation to compute the density of a normal distribution has the form:

Equation 1:

$$f(x) = 1/((2))e^{-((x-)^2/(2^2))}$$

Where:

- x is the value of the predictor.
- is the mean of the normal distribution.
- is the standard deviation.

The following code shows a simple example on how *dnorm()* uses **Equation 1** to compute the density of a normal function over a data range from a mean and a standard deviation:

```
niche.A <- dnorm(x=0:100, mean=50, sd=10)
```

We use the code above and a computation on the density of the driver to plot the ecological niche of the virtual taxa against the availability of driver values (**Figure 6**).

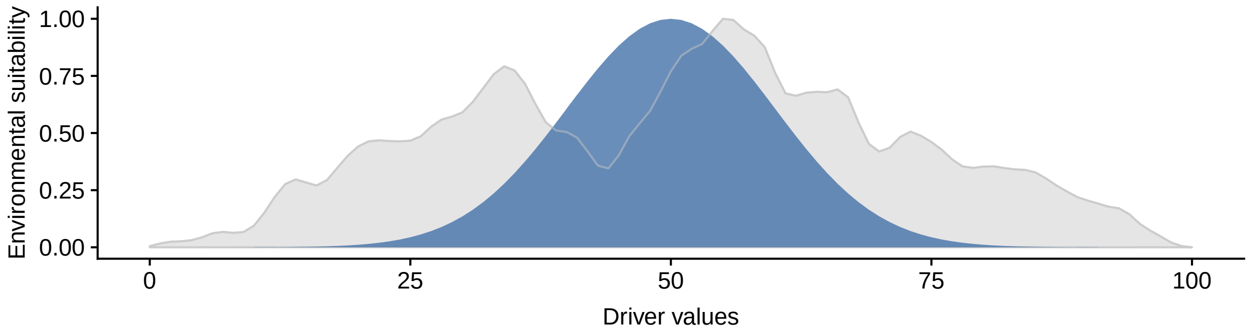


Figure 6: Ecological niche of the virtual species (blue) against the density (relative availability of values over time) of the driver (gray). Both densities have been scaled in the range [0, 1].

Environmental suitability for the given species over the study period is computed as follows:

1. *dnorm()* is computed on the mean and standard deviation defined for the species niche for a given driver.
2. The output of *dnorm()* is scaled to the range [0, 1].
3. The scaled values of the output are multiplied by the driver weights (which equals 1 if only one driver is used).
4. If there are two drivers, suitability values of each individual driver are summed together.

A **burn-in period** with a length of ten generations of the virtual taxa is added to the environmental suitability computed from the species niche and the driver/drivers. The added segment starts at maximum environmental suitability, stays there for five generations, and decreases linearly for another five generations until meeting the first suitability value of the actual simulation time. The whole burn-in segment has a small amount of white noise added (**Figure 7**). The burn-in period lets the population model to warm-up and go beyond starting conditions before simulation time starts.

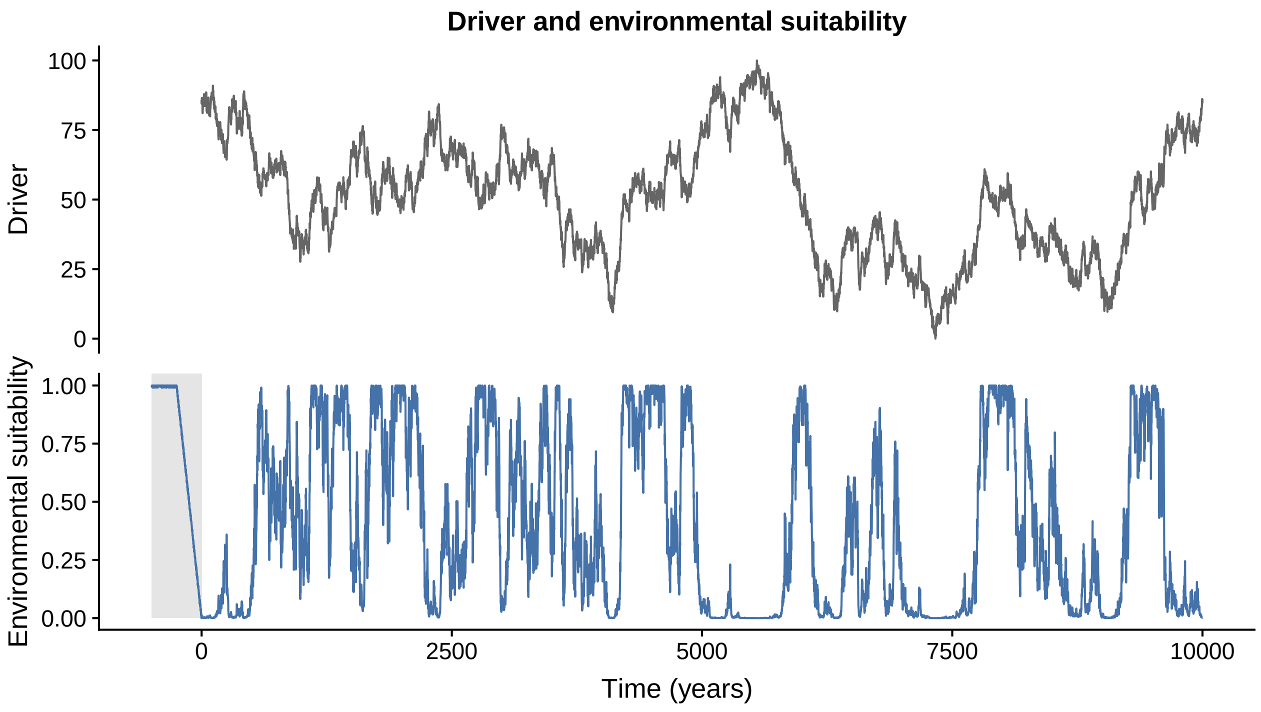


Figure 7: Driver and environmental suitability of the virtual taxa. Burn-in period is highlighted by a gray box in the Environmental suitability panel.

2.0.5 Biomass growth

Individuals age one year on each simulation step, and their biomass at any given age is defined by the following equation of logistic growth (**Equation 2**). **Figure 8** shows different growth curves for different growth rates for a virtual taxon with a maximum age of 400 years.

Equation 2:

$$f(x) = \frac{B}{1 + B} \times e^{(-\alpha \times t)}$$

Where:

- B is the maximum biomass an individual can reach.
- α is the growth rate.
- t is the age of the individual at a given time.

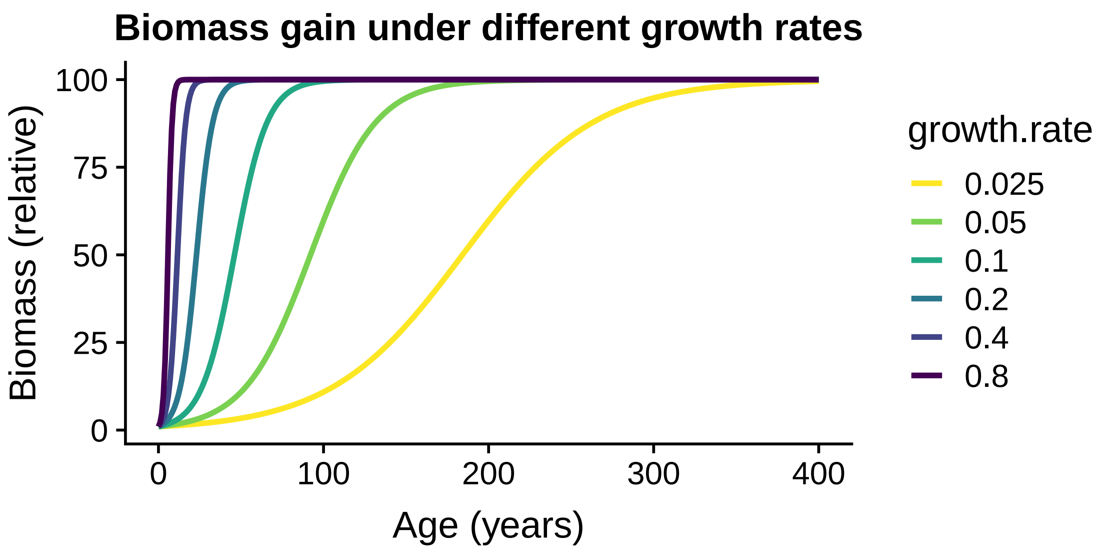


Figure 8: Biomass vs. age curves resulting from different growth rates for a 400 years life-span.

2.0.6 Population dynamics

The model starts with a population of 100 individuals with random ages, in the range [1, maximum age], taken from a uniform distribution (all ages are equiprobable). For each environmental suitability value, including the burn-in period, the model performs the following operations:

1. **Aging:** adds one year to the age of the individuals.
2. **Mortality due to senescence:** individuals reaching the maximum age are removed from the simulation.
    - **Local extinction and immigration** If the number of individuals drops to zero, the population is replaced by a "seed bank" of 100 individuals with age zero, and the simulation jumps to step **7.**. This is intended to simulate the arrival of seeds from nearby regions, and will only lead to population growth if environmental suitability is higher than zero.
3. **Plant growth:** Applies a plant growth equation to compute the biomass of every individual (see **Figure 8**).
4. **Carrying capacity:** If maximum population biomass is reached, individuals are iteratively selected for removal according to a mortality risk curve computed by **Equation 3** (see **Figure 9**). This curve gives removal preference to younger individuals, matching observed patterns in natural populations.
5. **Pollen productivity:** In each time step the model computes the pollen productivity (in relative values) of the population using **Equation 4**.
6. **Reproduction:** Generates as many seeds as reproductive individuals are available multiplied by the maximum fecundity and the environmental suitability of the given time.

The model returns a table with climatic suitability, pollen production, and population size (reproductive individuals only) per simulation year. **Figure 10** shows the results of the population model when applied to the example virtual species.

Equation 3:

$$P_m = 1 - sqrt(a/A)$$

Where:

- P_m is the probability of mortality.
- a is the age of the given individual.
- A is the maximum age reached by the virtual taxa.

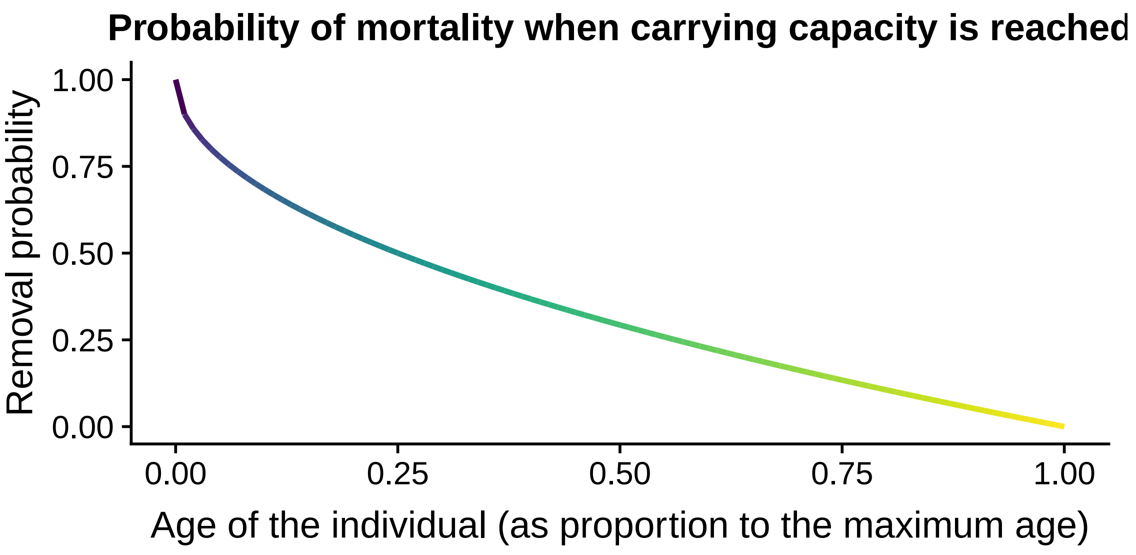


Figure 9: Risk curve defining the probability of removal of a given individual as a function of its fractional age when maximum carrying capacity is reached.

Equation 4:

$$P_t = \sum x_{it} \times max(S_t, B)$$

Where:

- t is time (a given simulation time step).
- P is the pollen productivity of the population at a given time.
- x_i represents the biomass of every adult individual.

- S is the environmental suitability at the given time.
- B is the contribution of biomass to pollen productivity regardless of environmental suitability (*pollen.control* parameter in the simulation, 0 by default). If B equals 1, P is equal to the total biomass sum of the adult population, regardless of the environmental suitability. If B equals 0, pollen productivity depends entirely on environmental suitability values.

The code below shows the core of the *simulatePopulation* function. It is slightly simplified to improve readability, and only pollen counts are written as output. Note that age of individuals is represented as a proportion of the maximum age to facilitate operations throughout the code.

```
#parameters (1st line in dataframe "parameters")
maximum.age <- parameters[1, "maximum.age"]
reproductive.age <- parameters[1, "reproductive.age"]
growth.rate <- parameters[1, "growth.rate"]
carrying.capacity <- parameters[1, "carrying.capacity"]
fecundity <- parameters[1, "fecundity"]

#reproductive age to proportion
reproductive.age <- reproductive.age / maximum.age

#years scaled taking maximum.age as reference
scaled.year <- 1/maximum.age

#vector to store pollen counts
pollen.count <- vector()

#starting population
population <- sample(seq(0, 1, by=scaled.year),
                     100,
                     replace=TRUE)

#iterating through suitability (once per year)
#------------------------------------
for(suitability.i in suitability){

  #AGING -----------------------------------------------
  population <- population + scaled.year
```

```r
#SENESCENCE ------------------------------------------
#1 is the maximum age of ages expressed as proportions
population <- population[population < 1]

#LOCAL EXTINCTION AND RECOLONIZATION -----------------
if (length(population) == 0){

  #local extinction, replaces population with a seedbank
  population <- rep(0, floor(100 * suitability.i))

  #adds 0 to pollen.count
  pollen.count <- c(pollen.count, 0)

  #jumps to next iteration
  next
}

#PLANT GROWTH ---------------------------------------
#biomass of every individual
biomass <- maximum.biomass /
  (1 +  maximum.biomass *
     exp(- (growth.rate * suitability.i) *
           (population * maximum.age)
         )
   )

#SELF-THINNING --------------------------------------
#carrying capacity reached
while(sum(biomass) > carrying.capacity){

  #removes a random individual based on risk curve
  individual.to.remove <- sample(
    x = length(population),
    size = 1,
    replace = TRUE,
      prob = 1 - sqrt(population) #risk curve
    )

    #removing individuals from population and biomass
    population <- population[-individual.to.remove]
    biomass <- biomass[-individual.to.remove]

  }#end of while

  #REPRODUCTION --------------------------------------
  #identifyies adult individuals
  adults <- population > reproductive.age

  #seeds (vector of 0s)
  #fractional biomass of adults * fecundity * suitability
  seeds <- rep(0, floor(sum((biomass[adults]/maximum.biomass) *
                            fecundity) * suitability.i))

  #adding seeds to the population
  population <- c(population, seeds)

  #POLLEN OUTPUT -------------------------------------
  #biomass of adults multiplied by suitability
  pollen.count <- c(pollen.count,
                    sum(biomass[adults]) * suitability.i)

} #end of loop through suitability values
```

The code below executes the simulation, and plots the outcome using the function *plotSimulation*.

```
#simulate population based on parameters
simulation <- simulatePopulation(parameters=parameters[1, ],
                                 drivers=drivers.10k)

#plotting simulation output
plotSimulation(simulation.output=simulation,
               burnin=TRUE,
               panels=c("Driver A",
                        "Suitability",
                        "Population",
                        "Pollen"),
               plot.title="",
               text.size=12,
               line.size=0.4)
```

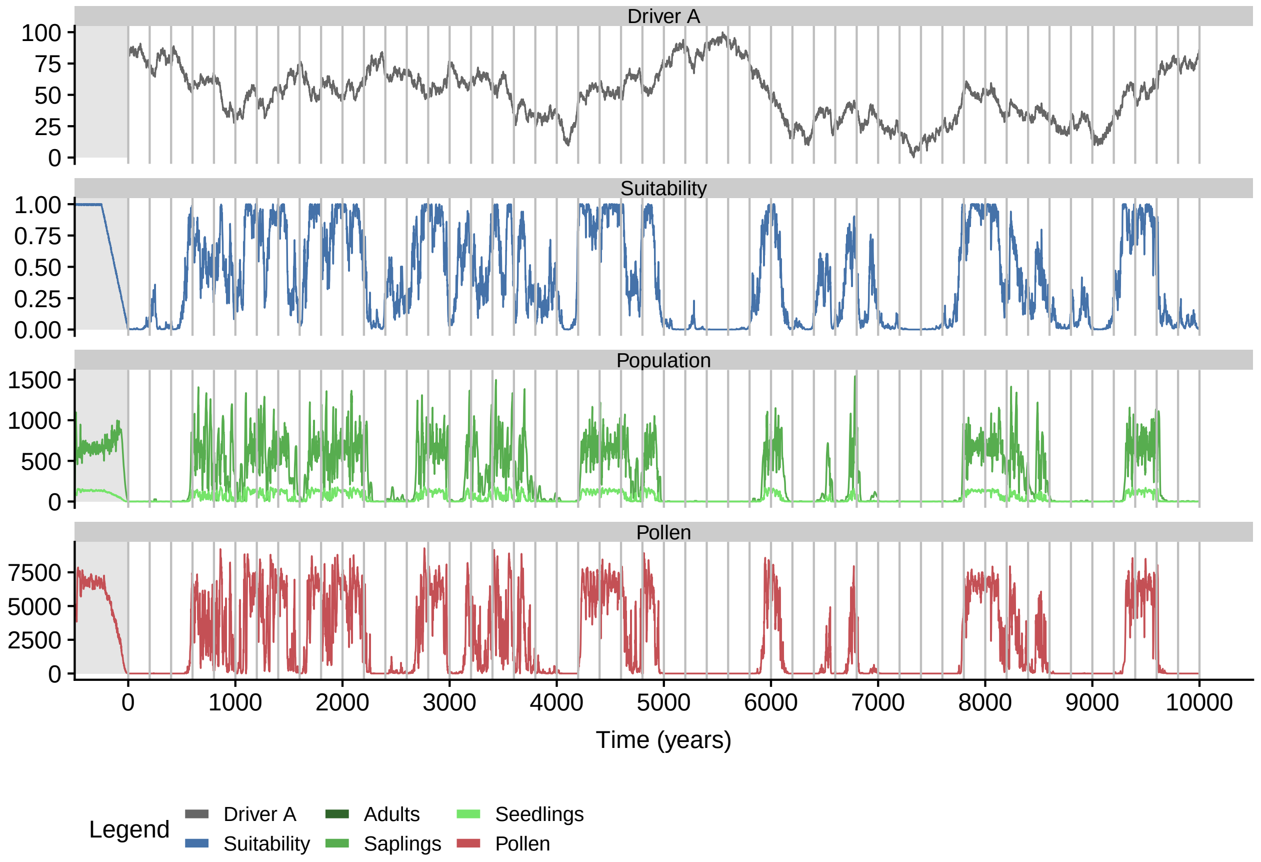


Figure 10: Simulation outcome. Green shades represent different age groups (seedlings, saplings, and adults).

The simulation outcomes can vary with the traits of the virtual species. **Table 2** shows the parameters of two new taxa named **Species 2** and **Species 3**. These species have a higher niche breadth than **Species 1**, and **Species 3** has a pollen productivity depending more on biomass than suitability (parameter *pollen.control* higher than zero). The comparison of both simulations (**Figure 11**) along with **Species 1** shows that different traits generate different pollen curves in our simulation.

Table 3: Parameters of the three virtual species.

Parameter	Species 1	Species 2	Species 3
maximum.age	50	50	50
reproductive.age	20	20	20
fecundity	2	4	6
growth.rate	0.2	0.3	0.4
pollen.control	0.0	0.0	0.5
maximum.biomass	100	100	100
carrying.capacity	10000	10000	10000
driver.A.weight	1	1	1
niche.A.mean	50	50	50
niche.A.sd	10	15	20

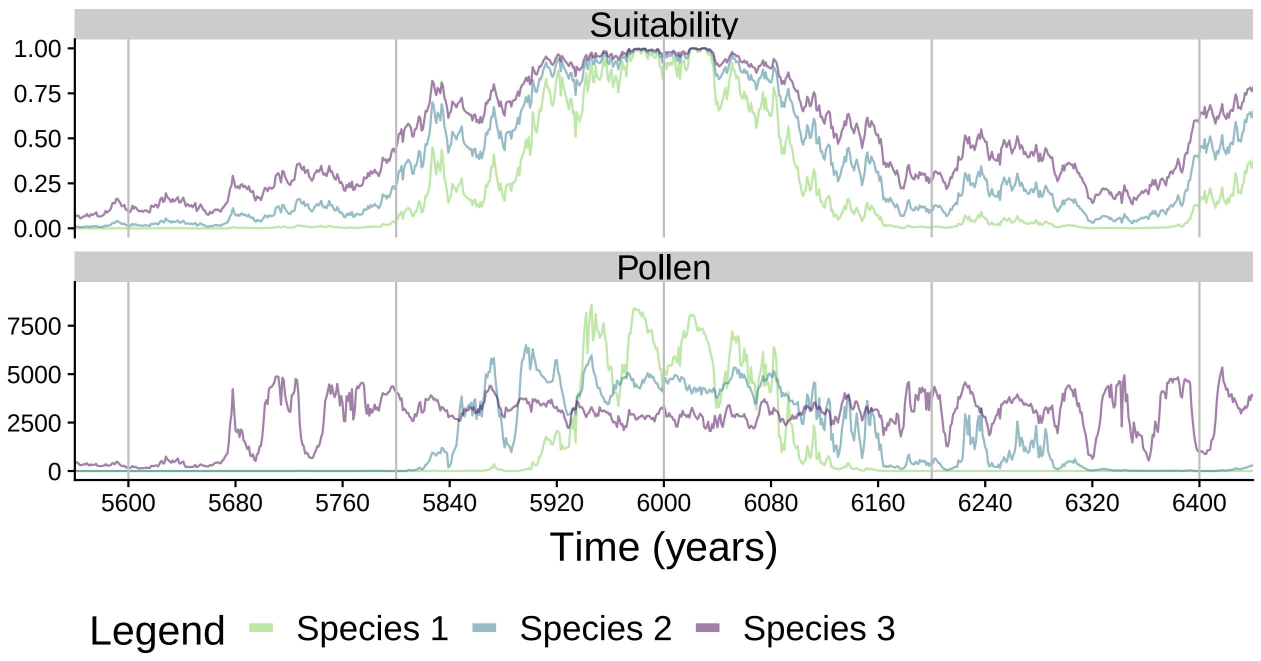


Figure 11: Comparison of the pollen abundance and environmental suitability (same in all cases) for the three virtual species shown in Table 2 within the period 4000-5000. Species 2 has a higher fecundity than Species 1 (1 vs 10)

2.0.7 Testing the model limits

We searched for the minimum values of the parameters required to keep a simulated population viable under fully suitable conditions. The taxa *Test 1* and *Test 2* shown below

```
simulation.test.1 <- simulatePopulation(
  parameters=parameters.test,
  driver.A=jitter(rep(50, 500), amount=4)
  )
```

The test driver used had an average of 50 (optimum values according the environmental niche of both species) for 500 years, with random noise added through the *jitter* function. The model results (column *Pollen* only) for both virtual taxa are shown below.

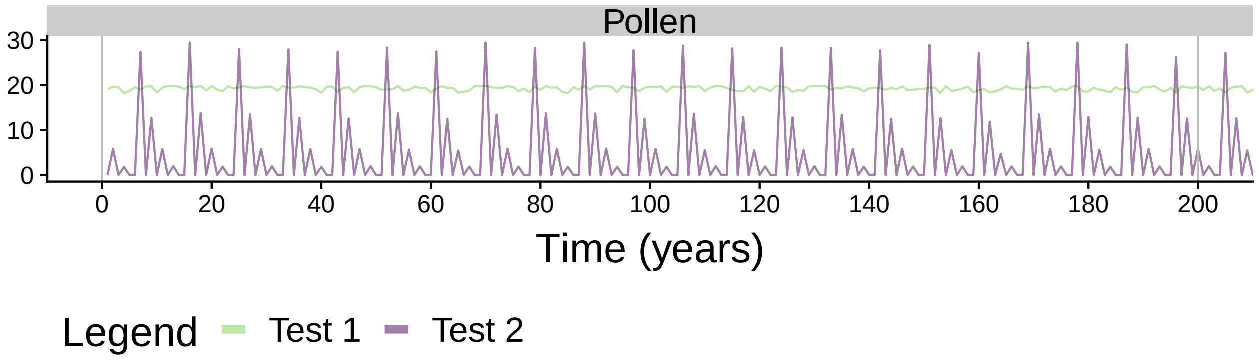


Figure 12: Pollen output of virtual taxa Test 1 and Test 2 for a 200 years time-window.

The outputs of the test taxa show how a minimal change in the parameters can lead to fully unstable results when the considered taxa are short lived. Considering such a result, values for life-traits (*maximum.age*, *reproductive.age*, *fecundity*, *growth.rate*, and *maximum.biomass*) of taxon *Test 1* should be taken as safe lower limits for these traits.

A similar situation can happen with long lived species when the age of sexual maturity is close to the maximum age. The table below shows three new test species with long life-spans and increasing maturity ages. The driver is again centered in 50, with added white noise, and 2000 years length.

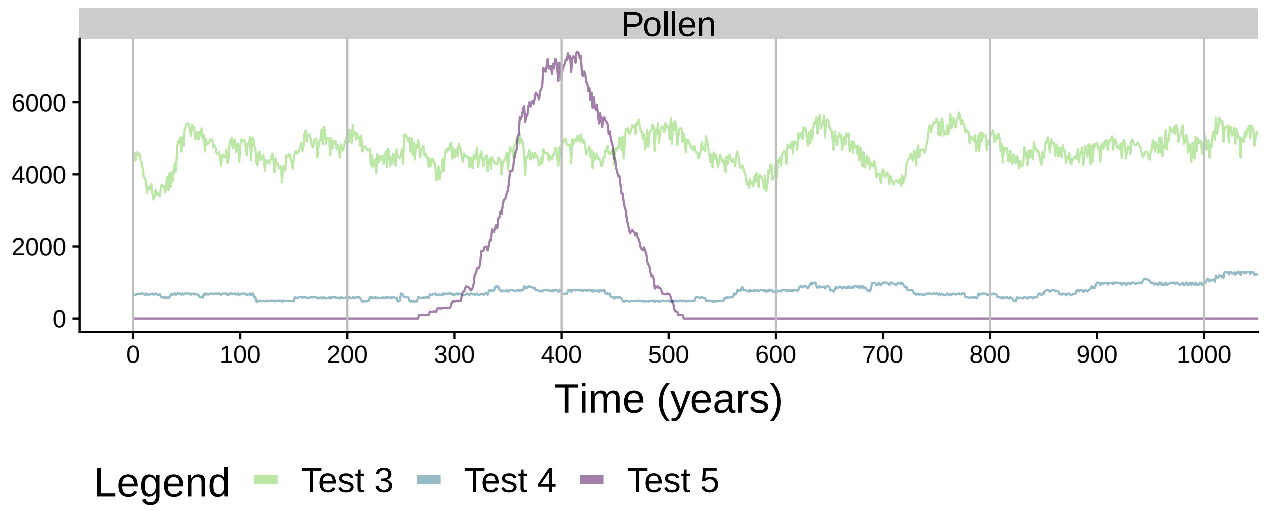


Figure 13: Pollen output of virtual taxa Test 1 and Test 2 for a 200 years time-window.

The figure shows how *Test 3* yields a stable pollen productivity across time, while *Test 4* and *Test 5* show, respectively, a very low productivity due to scarcity of adults, a total inability to sustain stable populations. Considering these results, it is important to keep a careful balance between the parameters *maximum.age* and *reproductive.age* to obtain viable virtual populations.

3 Simulating a virtual accumulation rate

Sediments containing pollen grains accumulate at varying rates, generally measured in *years per centimeter* (y/cm). Accumulation rates found in real datasets are broadly between 10 and 70 y/cm, with a paulatine increase towards the present time. To simulate such an effect and aggregate the annual data produced by the simulation in a realistic manner we have written a function named *simulateAccumulationRate* that takes a random seed, a time-span, and a range of possible accumulation rate values, and generates a virtual accumulation rate curve. It does so by generating a random walk first, smoothing it through the application of a GAM model, and scaling it between given minimum and maximum accumulation rates (see **Figure 12**).

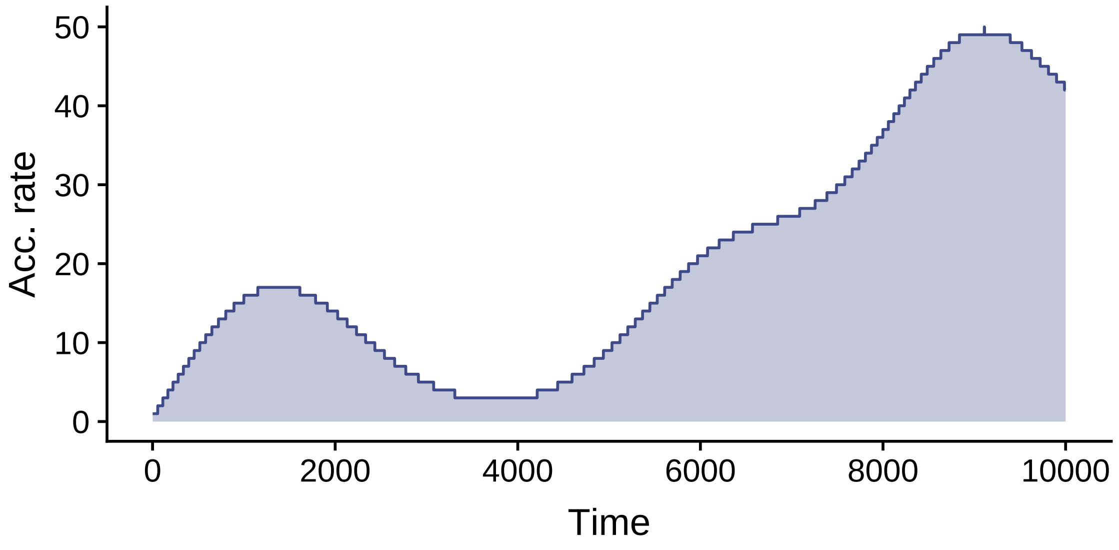


Figure 14: Virtual accumulation rate.

The output is a dataframe with three columns: *time*, *accumulation.rate*, and *grouping* (see **Table 4**).

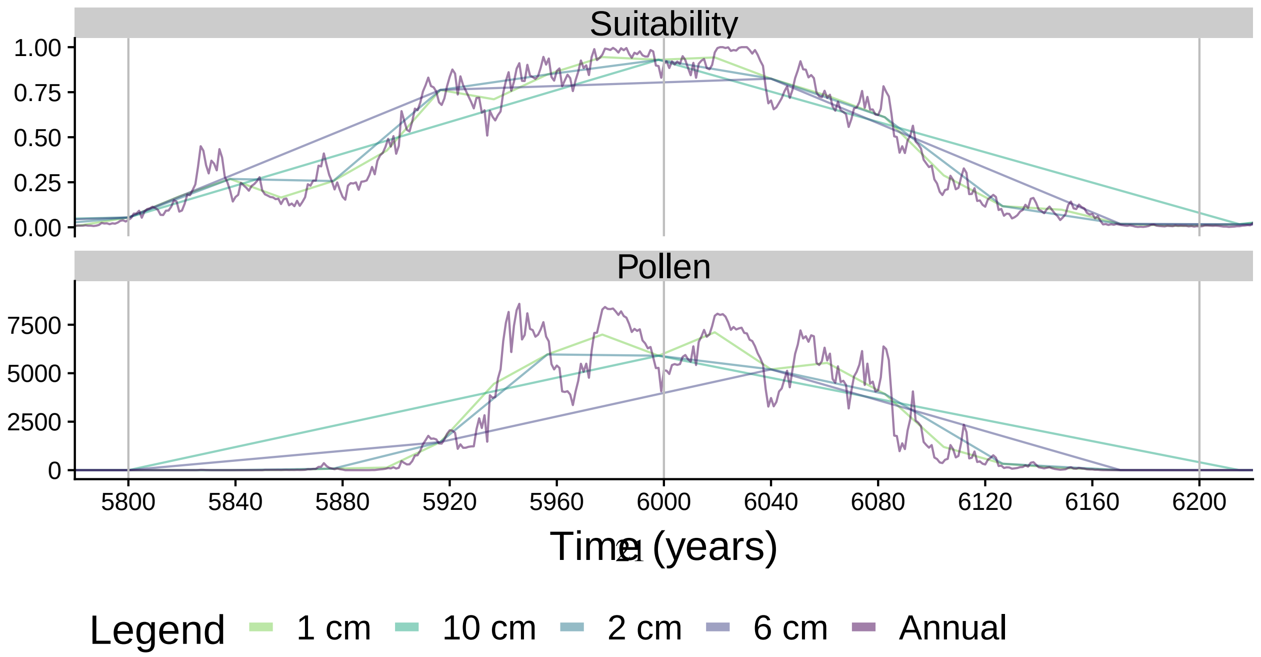
Cases of the simulation data belonging to the same group according to the *grouping* column are aggregated together to simulate a *centimeter* of sedimented data. The data are aggregated by the function *aggregateSimulation*, that can additionally sample the resulting data at given depth intervals expressed in centimeters between consecutive samples (2, 6, and 10 cm in the code below).

```
simulation.aggregated <- aggregateSimulation(
  simulation.output=simulation,
  accumulation.rate=accumulation.rate,
  sampling.intervals=c(2, 6, 10)
  )
```

The function returns a matrix-like list with as many rows as simulations are available in *simulation.output*, a column containing the data of the original simulations, a column with the data aggregated every centimeter, and the sampling intervals requested by the user. The data are accessed individually by list subsetting, as shown below (see **Figure 13**), to allow easy analysis and visualization.

Table 4: Dataframe resulting from the function to generate virtual accumulation rates. Each group in the grouping column has as many elements as accumulation.rate the given group has.

time	accumulation.rate	grouping
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	1	6
7	1	7
8	1	8
9	1	9
10	1	10
11	1	11
12	1	12
13	1	13
14	1	14
15	1	15
16	1	16
17	1	17
18	1	18
19	1	19
20	1	20



4 Sampling virtual pollen curves at different depth intervals

Applying a virtual accumulation rate to the data generated by the population model at given depth intervals simulates to a certain extent how pollen deposition and sampling work in reality, and the outcome of that is data-points separated by regular depth intervals, but not regular time intervals. **Figure 14** shows that time intervals between consecutive samples produced by *aggregateSimulation* are not regular. However, analyzing ecological memory requires to organize the input data in regular time lags, and to do that the data need to have regular time intervals between consecutive cases.

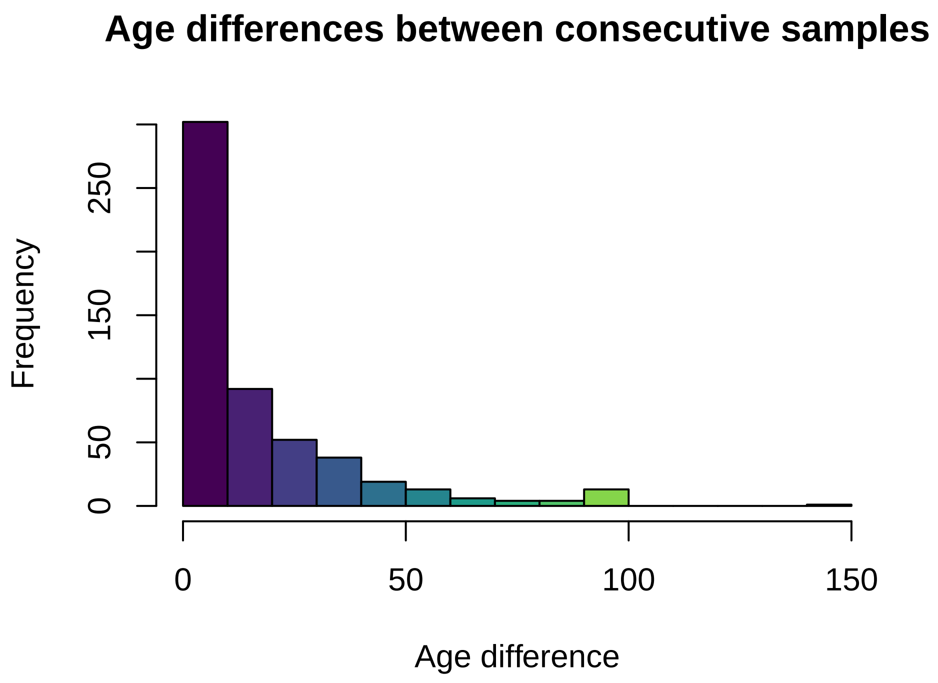


Figure 16: Histogram of the time differences (in years) between consecutive samples for the outcome of aggregateSimulation when resampled at intervals of 6 centimeters on Species 1. It clearly shows how the data are not organized in regular time intervals, and therefore are unsuitable for analyses requiring regular time lags.

Irregular time series can be interpolated into regular time series by using the *loess* function. This function fits a polynomial surface representing the relationship between two (or more) variables. The smoothness of this polynomial surface is modulated by the *span* parameter, and finding the right value for this parameter is critical to obtain an interpolation result as close as possible to the real data. The following code is able to find the value of *span* that maximizes the correlation between input and interpolated data for any given time series.

```r
#getting example data sampled at 2cm intervals
simulated.data = simulation.aggregated[[1, 3]]


#span values to be explored
span.values=seq(20/nrow(simulated.data),
                5/nrow(simulated.data),
                by=-0.0005)


#plotting the optimization process in real time
x11(height=12, width=24)

#iteration through candidate span values
for(span in span.values){

  #function to interpolate the data
  interpolation.function = loess(
    Pollen ~ Time,
    data=simulated.data,
    span=span,
    control=loess.control(surface="direct"))

  #plot model fit
  plot(simulated.data$Pollen, type="l", lwd=2)
  lines(interpolation.function$fitted, col="red", lwd=2)

  #if correlation equals 0.9985 loop stops
  if(cor(interpolation.function$fitted,
    simulated.data$Pollen) >=  0.9985){break}

}

#gives time to look at result before closing the plot window
Sys.sleep(5)
```

The function *toRegularTime* (usage shown below), uses the code above to interpolate the data produced by *aggregateSimulation* into a given time interval, expressed in years, returning a list of the same dimensions of the input list.

```r
simulation.interpolated <- toRegularTime(
  x=simulation.aggregated,
  time.column="Time",
  interpolation.interval=10,
  columns.to.interpolate=c("Pollen",
                           "Suitability",
                           "Driver.A")
  )
```

Figure 15 shows the same data segment shown in **Figure 13**, but with samples re-interpolated into a regular time grid at 10 years intervals.

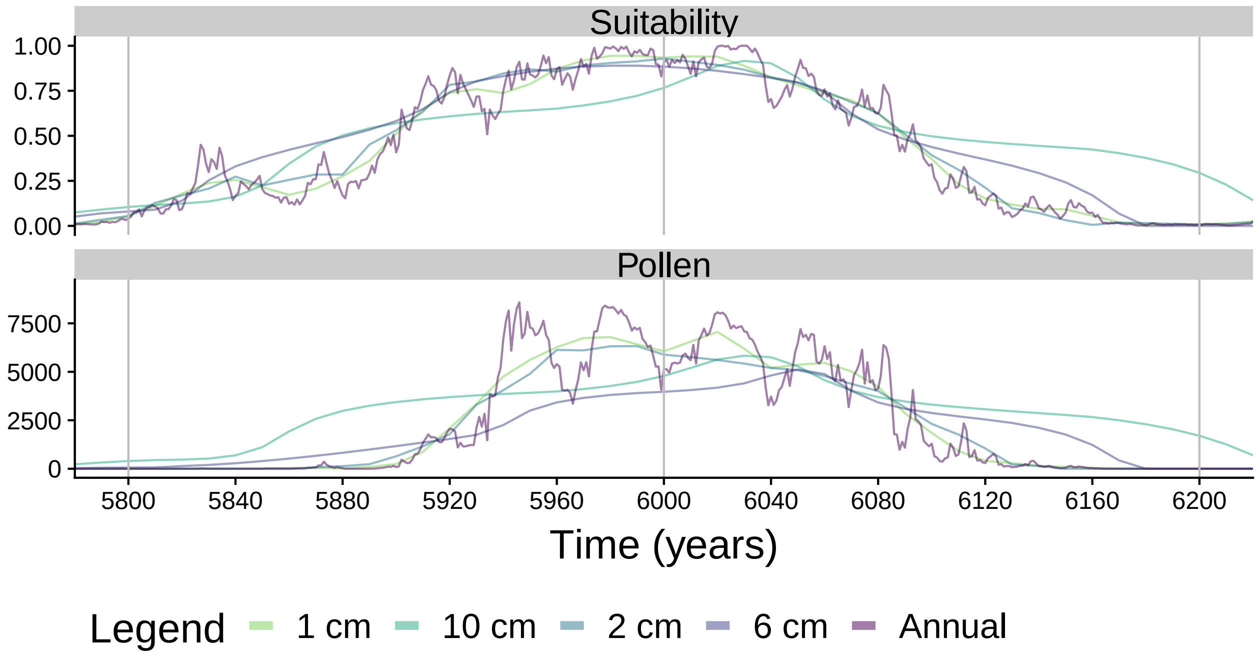


Figure 17: Data aggregated using virtual accumulation rate and reinterpolated into a regular time grid of 10 years resolution.