Appendix II

Analyzing ecological memory with Random Forest

Blas M. Benito

Contents

Summary

This document describes in detail how we analyzed ecological memory patterns in simulated pollen curves with Random Forest. First, we describe the complex statistical properties of the virtual pollen curves produced by the simulation explained in **Appendix I** and how these may impact ecological memory analyses; second we explain how Random Forest works, from its basic components (regression trees) to the way in which it computes variable importance; Third, we explain how we applied Random Forest to analyze ecological memory patterns on the simulation outputs.

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

1 Statistical properties of simulated pollen curves

The simulation explained in **Appendix I** generates synthetic pollen curves from virtual taxa with different life-traits and niche features as a response to changes in the values of an environmental driver. The driver values are transformed into suitability values by a Gaussian function representing the environmental niche of the given taxa. **Figure 1** shows a sample of pollen abundances, suitability, and driver values generated by the simulation for a virtual taxa with 1000 years life-span, and a wide environmental niche centered on the average values of the driver.

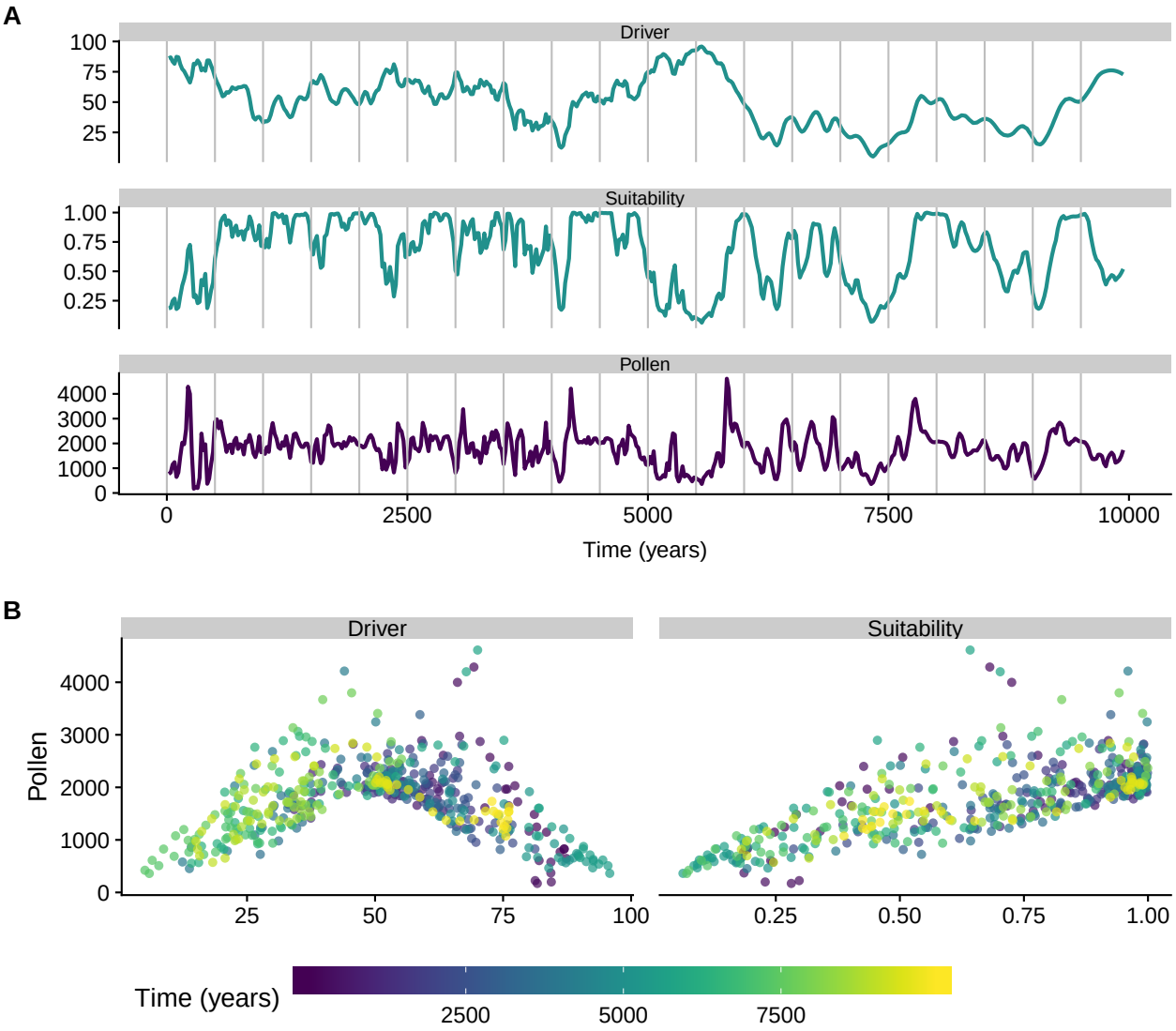

Figure 1: Example data from a virtual taxa represented with 1000 years life-span, and a centered and wide environmental niche. Time series are shown in panel A. Panel B shows relationships between variables.

Our primary target is to fit a model of the form shown in **Equation 1** on the data shown in **Table 1**:

Equation 1 (simplified from the one in the paper):

$$p_t = p_{t-1} + ... + p_{t-n} + d_t + d_{t-1} + ... + d_{t-n}$$

Where:

- p is *Pollen*.
- d is *Driver*.
- t is the time of any given value of the response p.
- $t-1$ is the lag 1.
- $p_{t-1} + ... + p_{t-n}$ represents the endogenous component of ecological memory.
- $d_{t-1} + ... + d_{t-n}$ represents the exogenous component of ecological memory.
- d_t represents the concurrent effect of the driver over the response.

To organize the data in **Figure 1** as required to fit **Equation 1** it is necessary to define a set of *time lags*. The function *prepareLaggedData* shown below organizes the data in such a format. It requires to identify what columns in the original data should act as response, drivers, and time, and what lags are to be computed.

```
#generating vector of lags (same as in paper)
lags <- seq(20, 240, by = 20)

#organizing data in lags
sim.lags <- prepareLaggedData(simulation.data = sim,
                              response = "Pollen",
                              drivers = c("Driver", "Suitability"),
                              time = "Time",
                              lags = lags,
                              scale = FALSE)
```

This function returns the data shown in **Table 1**. This kind of data structure is known as *lagged data* or *time delayed data*. Note that the function can use a *scale* argument (set to FALSE above) to standardize the data before generating the lags. Random Forest does not generally require standardization to fit accurate models of the data, but computing variable importance with variables having large differences in range (i.e. [1, 10] vs. [1, 10000]) might yield biased results, making standardization a recommended step in data preparation. In this appendix all data are shown without any standardization to let the reader to keep track of the different variables across analyses and have a sense of their magnitude, but note that all analyses presented in the paper were based on standardized data.

Table 1: First rows of the lagged data. Numbers represent lag in years, letter p represents pollen, and letter d represents driver. Column p0 (in bold) indicates the response variable

p0	p20	p40	p60	p80	p100	p120	p140	p160	p180	p200	p220	p240	d0	d20	d40	d60	d80	d100	d120	d140	d160	d180	d200	d220	d240
1611.2	3997.8	4289.3	2571.5	1964.9	2030.3	1485.4	829.1	646.4	1244.3	1147.0	823.9	769.6	71.4	66.1	69.3	72.4	75.5	77.3	81.1	86.9	87.3	81.6	83.5	86.9	86.1
171.1	1611.2	3997.8	4289.3	2571.5	1964.9	2030.3	1485.4	829.1	646.4	1244.3	1147.0	823.9	81.7	71.4	66.1	69.3	72.4	75.5	77.3	81.1	86.9	87.3	81.6	83.5	86.9
224.7	171.1	1611.2	3997.8	4289.3	2571.5	1964.9	2030.3	1485.4	829.1	646.4	1244.3	1147.0	81.3	81.7	71.4	66.1	69.3	72.4	75.5	77.3	81.1	86.9	87.3	81.6	83.5
199.2	224.7	171.1	1611.2	3997.8	4289.3	2571.5	1964.9	2030.3	1485.4	829.1	646.4	1244.3	84.4	81.3	81.7	71.4	66.1	69.3	72.4	75.5	77.3	81.1	86.9	87.3	81.6
606.0	199.2	224.7	171.1	1611.2	3997.8	4289.3	2571.5	1964.9	2030.3	1485.4	829.1	646.4	82.0	84.4	81.3	81.7	71.4	66.1	69.3	72.4	75.5	77.3	81.1	86.9	87.3
2398.4	606.0	199.2	224.7	171.1	1611.2	3997.8	4289.3	2571.5	1964.9	2030.3	1485.4	829.1	76.1	82.0	84.4	81.3	81.7	71.4	66.1	69.3	72.4	75.5	77.3	81.1	86.9
1725.4	2398.4	606.0	199.2	224.7	171.1	1611.2	3997.8	4289.3	2571.5	1964.9	2030.3	1485.4	77.9	76.1	82.0	84.4	81.3	81.7	71.4	66.1	69.3	72.4	75.5	77.3	81.1
1966.6	1725.4	2398.4	606.0	199.2	224.7	171.1	1611.2	3997.8	4289.3	2571.5	1964.9	2030.3	75.8	77.9	76.1	82.0	84.4	81.3	81.7	71.4	66.1	69.3	72.4	75.5	77.3
370.5	1966.6	1725.4	2398.4	606.0	199.2	224.7	171.1	1611.2	3997.8	4289.3	2571.5	1964.9	84.3	75.8	77.9	76.1	82.0	84.4	81.3	81.7	71.4	66.1	69.3	72.4	75.5
576.6	370.5	1966.6	1725.4	2398.4	606.0	199.2	224.7	171.1	1611.2	3997.8	4289.3	2571.5	84.3	84.3	75.8	77.9	76.1	82.0	84.4	81.3	81.7	71.4	66.1	69.3	72.4
1165.4	576.6	370.5	1966.6	1725.4	2398.4	606.0	199.2	224.7	171.1	1611.2	3997.8	4289.3	80.1	84.3	84.3	75.8	77.9	76.1	82.0	84.4	81.3	81.7	71.4	66.1	69.3
1651.4	1165.4	576.6	370.5	1966.6	1725.4	2398.4	606.0	199.2	224.7	171.1	1611.2	3997.8	76.1	80.1	84.3	84.3	75.8	77.9	76.1	82.0	84.4	81.3	81.7	71.4	66.1
2871.5	1651.4	1165.4	576.6	370.5	1966.6	1725.4	2398.4	606.0	199.2	224.7	171.1	1611.2	70.5	76.1	80.1	84.3	84.3	75.8	77.9	76.1	82.0	84.4	81.3	81.7	71.4
2973.3	2871.5	1651.4	1165.4	576.6	370.5	1966.6	1725.4	2398.4	606.0	199.2	224.7	171.1	66.5	70.5	76.1	80.1	84.3	84.3	75.8	77.9	76.1	82.0	84.4	81.3	81.7
2609.9	2973.3	2871.5	1651.4	1165.4	576.6	370.5	1966.6	1725.4	2398.4	606.0	199.2	224.7	63.3	66.5	70.5	76.1	80.1	84.3	84.3	75.8	77.9	76.1	82.0	84.4	81.3
2890.5	2609.9	2973.3	2871.5	1651.4	1165.4	576.6	370.5	1966.6	1725.4	2398.4	606.0	199.2	59.1	63.3	66.5	70.5	76.1	80.1	84.3	84.3	75.8	77.9	76.1	82.0	84.4
2443.9	2890.5	2609.9	2973.3	2871.5	1651.4	1165.4	576.6	370.5	1966.6	1725.4	2398.4	606.0	54.7	59.1	63.3	66.5	70.5	76.1	80.1	84.3	84.3	75.8	77.9	76.1	82.0
2111.9	2443.9	2890.5	2609.9	2973.3	2871.5	1651.4	1165.4	576.6	370.5	1966.6	1725.4	2398.4	55.5	54.7	59.1	63.3	66.5	70.5	76.1	80.1	84.3	84.3	75.8	77.9	76.1
1657.2	2111.9	2443.9	2890.5	2609.9	2973.3	2871.5	1651.4	1165.4	576.6	370.5	1966.6	1725.4	58.2	55.5	54.7	59.1	63.3	66.5	70.5	76.1	80.1	84.3	84.3	75.8	77.9
1906.4	1657.2	2111.9	2443.9	2890.5	2609.9	2973.3	2871.5	1651.4	1165.4	576.6	370.5	1966.6	57.7	58.2	55.5	54.7	59.1	63.3	66.5	70.5	76.1	80.1	84.3	84.3	75.8
1956.9	1906.4	1657.2	2111.9	2443.9	2890.5	2609.9	2973.3	2871.5	1651.4	1165.4	576.6	370.5	59.2	57.7	58.2	55.5	54.7	59.1	63.3	66.5	70.5	76.1	80.1	84.3	84.3
1609.0	1956.9	1906.4	1657.2	2111.9	2443.9	2890.5	2609.9	2973.3	2871.5	1651.4	1165.4	576.6	62.2	59.2	57.7	58.2	55.5	54.7	59.1	63.3	66.5	70.5	76.1	80.1	84.3
2099.4	1609.0	1956.9	1906.4	1657.2	2111.9	2443.9	2890.5	2609.9	2973.3	2871.5	1651.4	1165.4	61.2	62.2	59.2	57.7	58.2	55.5	54.7	59.1	63.3	66.5	70.5	76.1	80.1
2225.9	2099.4	1609.0	1956.9	1906.4	1657.2	2111.9	2443.9	2890.5	2609.9	2973.3	2871.5	1651.4	60.7	61.2	62.2	59.2	57.7	58.2	55.5	54.7	59.1	63.3	66.5	70.5	76.1
1776.7	2225.9	2099.4	1609.0	1956.9	1906.4	1657.2	2111.9	2443.9	2890.5	2609.9	2973.3	2871.5	62.1	60.7	61.2	62.2	59.2	57.7	58.2	55.5	54.7	59.1	63.3	66.5	70.5

The data in **Table 1** are organized to fit the model described by **Equation 1**, but to select a proper method to fit the model, three main features of the data have to be considered first: **temporal autocorrelation**, **multicollinearity**, and **non-linearity**.

1.0.1 Temporal autocorrelation

Temporal autocorrelation (also *serial correlation*) refers to the relationship between successive values of the same variable present in most time series. Temporal autocorrelation generates autocorrelated residuals in regression analysis, violating the assumption of "independence of errors" required to correctly interpret regression coefficients. Several methods can be used to address temporal autocorrelation in regression analysis, such as increasing time intervals between consecutive samples, or incorporating an auto-regressive structure into the model.

Every variable used in our study presents this characteristic. The driver was generated *ex profeso* with a temporal autocorrelation significant for periods of 600 years. The suitability produced by the niche function of the virtual taxa based on the values of the driver also presents temporal autocorrelation, but generally lower than the one of the driver. Finally, the response, since it is the result of a dynamic model in which every data-point depends on the previous one, also shows a temporal structure, which varies depending on the taxa's traits, as so does the suitability (see **Figure 2**).

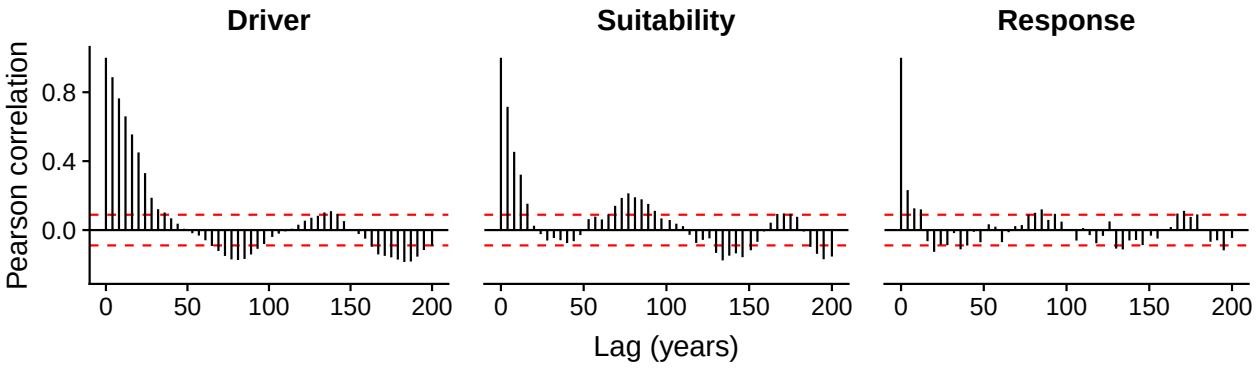

Figure 2: Temporal autocorrelation of the variables in the example data.

1.0.2 Multicollinearity

Multicollinearity occurs when there is a high correlation between predictors in a model definition. It increases the standard error of the coefficients, meaning that their estimates for important predictors can become statistically insignificant, wildly impacting model interpretation.

Adding consecutive time-lags of the same variables to the data, as required by the model expressed in **Equation 1** largely increases multicollinearity. Furthermore, this effect is amplified by the data sampling at increasing depth intervals and posterior re-interpolation to 20 years resolution, as shown in **Table 3**.

Table 2: Variance inflation factor (VIF) of the predictors across datasets available for a virtual taxa with 1000 years life-span, and wide and central niche. VIF values higher than 5 indicate that the given predictor is a linear combination of other predictors.

	Annual	1cm	2cm	6cm	10cm
p20	1.9	3.2	3.9	9.8	16.9
p40	2.7	6.6	8.5	33.0	69.6
p60	2.7	6.8	8.4	41.6	93.7
p80	2.7	6.7	8.2	43.4	97.4
p100	2.7	6.7	8.2	43.8	99.3
p120	2.7	6.8	8.3	42.5	101.2
p140	2.7	6.8	8.4	42.5	101.5
p160	2.7	6.9	8.3	43.3	101.1
p180	2.7	6.9	8.3	42.5	99.7
p200	2.7	6.9	8.4	40.2	93.4
p220	2.7	6.7	8.2	31.6	65.1
p240	1.9	3.2	3.7	9.4	15.5
d0	17.7	50.4	43.4	97.3	152.2
d20	34.4	153.4	131.9	349.7	662.2
d40	34.3	162.1	143.3	410.8	958.6
d60	34.5	161.7	143.7	424.4	1154.0
d80	34.5	162.8	145.3	431.1	1232.4
d100	34.6	163.6	146.1	427.6	1237.2
d120	34.9	164.6	145.7	421.0	1212.7
d140	35.0	163.7	145.6	421.7	1221.3
d160	35.2	161.2	145.4	420.1	1235.6
d180	35.4	160.1	144.2	414.7	1275.9
d200	35.5	160.2	142.9	403.5	1247.2
d220	35.6	154.3	134.9	343.7	874.6
d240	18.3	51.7	45.2	96.6	182.4

1.0.3 Non-linearity

The simulation model described in **Appendix I** has the ability to produce pollen abundances variably decoupled from environmental conditions depending on the life-traits and niche features of the virtual taxa considered. This model property increases the chance of finding non-linear relationships between time-lagged predictors and the response (see **Figure 3**), hindering the detection of meaningful relationships with methods not able to account for non-nonlinearity.

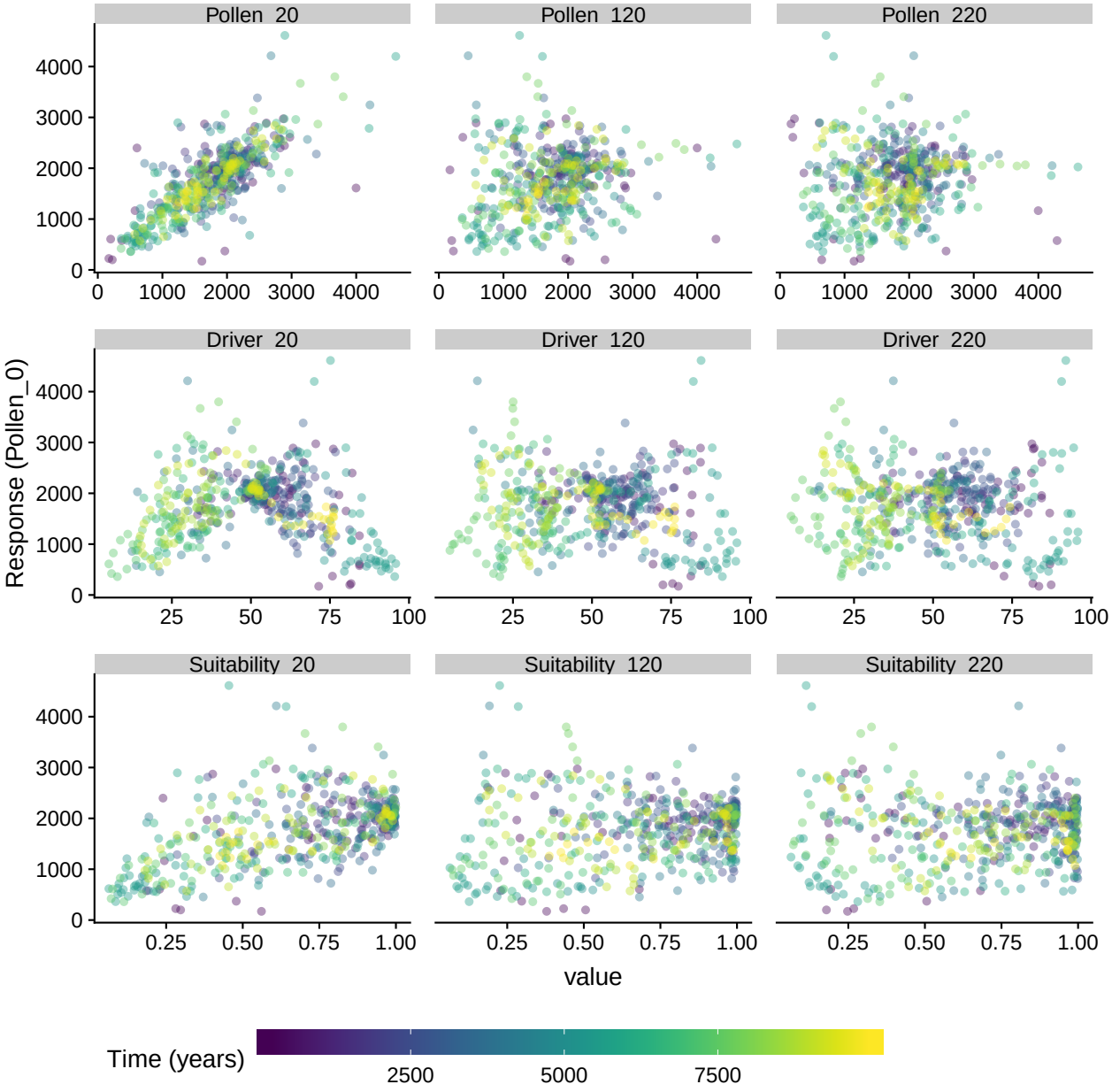

Figure 3: Linear and non-linear relationships arising from lagged data in the annual dataset.

After considering carefully the properties of our data, and how they changed across datasets, and after testing several methods to evaluate the importance of lagged variables in predicting pollen output, we decided to use **Random Forest** to evaluate ecological memory patterns arising from our simulation. The following section of the documents explains in detail how Random Forest works, and why it is a suitable tool to analyze ecological memory in our data.

2 The logics behind Random Forest

2.1 The trees

The fundamental units of a Random Forest model are **regression trees**. A regression tree grows through **binary recursive partition**. Considering a *response* variable, and a set of *predictive variables* (also named *features* in the machine learning language), the following steps grow a regression tree:

- For each variable, the point in their ranges that optimizes the separation (partition) of the response in two groups of cases is searched for. The selected point minimizes the sum of the squared deviations from the mean in the two separated partitions.
- The variable with the lower sum of the squared deviations from the means is selected as the *root node* of the tree, and the data are separated in two partitions, one on each side of the *split value* of the given variable.
- The steps above are recursively applied to each partition to create new partitions, until all cases are in partitions that can be no longer separated in smaller partitions because they are too homogeneous, or because they have reached a minimum sample size. These final partitions are named *terminal nodes*.

The code below shows how to fit a recursive partition tree with the *rpart* library on the first lag (20 years) of pollen and driver of the data shown in **Figure 2**.

```
#fitting model (only two predictors)
rpart.model <- rpart(formula = Response_0 ~ Response_20 + Driver_20,
                     data = sim.lags,
                     control = rpart.control(minbucket = 5))

#plotting tree
rpart.plot(rpart.model, type = 0, box.palette = viridis(10, alpha = 0.2))
```

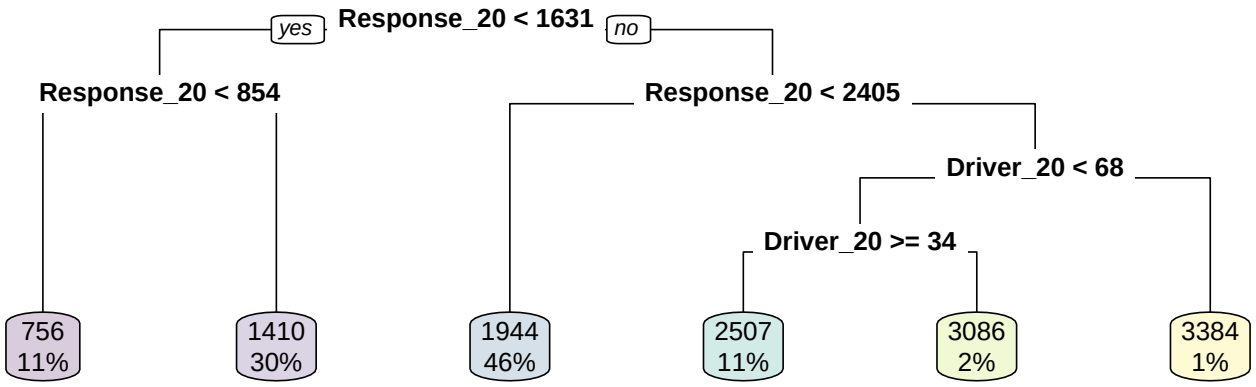

Figure 4: Recursive partition tree (also regression tree) of pollen abundance (noted as Response_0 in the model) as a function of Response_20 (antecedent pollen abundance at lag 20) and Driver_20 (antecedent driver values at lag 20). Numbers in branches represent split values, while numbers in terminal nodes represent the predicted average for that particular node. Percentages represent the relative number of cases included in each terminal node.

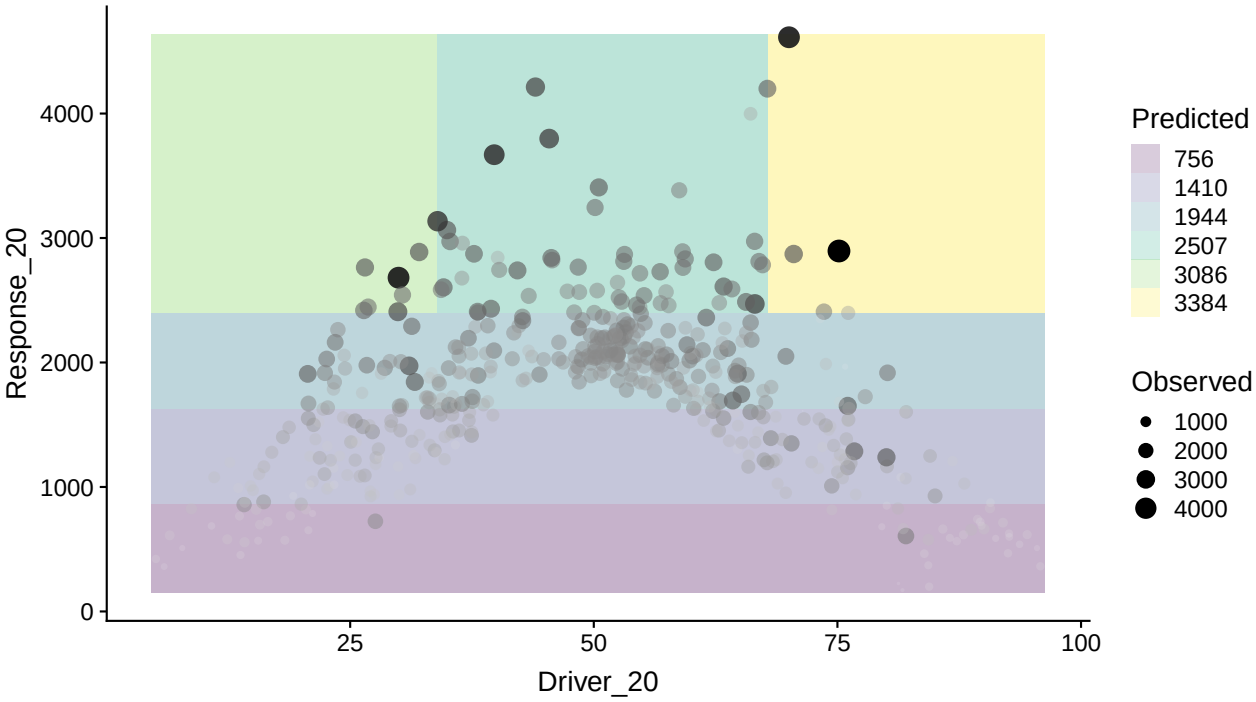

Figure 5: Recursive partition surface generated by the model fitted above. Dots represent observed data, and colours identify partitions shown in the recursive partition tree. Note that this partition surface can be generalized to any number of predictors/dimensions.

Figure 4 shows the recursive partition tree fitted on Pollen_0 as a function of the first lag of pollen (Pollen_20) and the driver (Driver_20), while **Figure 5** shows the partitions in the space of both

variables. As shown in both figures, the recursive partition tree is, in essence, separating the cases into regions in which given combinations of the predictors lead to certain average values of the response. The tree also shows the hierarchy in importance between both predictors, with *Pollen_20* defining all splits but one. Only when *Pollen_20* is higher than 3772, the variable *Driver_20* becomes important, indicating that maximum abundances are only reached after that point, and only if *Driver_20* has a value lower than 71. This is how partial interactions among predictors are expressed in recursive partition trees.

The tree has grown until data in the terminal nodes cannot be separated further into additional partitions, or has reached the minimum number of cases defined by the variable *minbucket*. The minimum amount of cases in a terminal node defines the overall resolution of the model. Smaller numbers lead to a higher amount of terminal nodes, and therefore to more partitions in the data space. This can be confirmed by changing the *minbucket* value in the code above, and assessing subsequent changes in tree structure and number of partitions.

As a drawback, the splits of a recursive partition trees are highly sensitive to small changes in the input data, specially when sample size is small. This instability has led to the development of more sophisticated methods to fit recursive partition trees, such as *conditional inference trees* (see function *ctree* in library *partykit*), or ensemble methods such as Random Forest.

2.2 The forest

A Random Forest model is composed by a large number of individual regression trees (500 or more) generated on random subsamples of the predictors and the cases. For a random set of cases, each tree is fitted as follows:

- A random subset of predictors of size *mtry* is selected. The default *mtry* is the rounded-up squared root of the total number of predictors, but the user can modify it.
- The predictor from the random subset that better separates the data into two partitions is selected as *root node*, an the data are separated in two partitions, one at each side of the *split value*.
- On each partition, a new random subset of predictors of size *mtry* is selected (and this is the main difference between a recursive partition tree and a Random Forest tree, the former uses all variables on each split), and again the predictor that better separates the partition into two new partitions is selected, and new partitions are defined.
- The tree is grown until minimum node size is reached in all terminal nodes, or no further partitions can be defined.
- The tree is evaluated by computing its mean squared error (mse) on the ~37% of the data not used to train it (named *out-of-bag data*).

- For each variable in the tree the algorithm performs a permutation test as follows:
- The column with the given variable is randomly permuted.
- A new tree is fitted with the permuted variable.
- Mean squared error is computed again on the *out-of-bag* data.
- Difference in mse between the tree fitted with the original variable and the tree fitted with the permuted one is computed and stored.

Once all trees have been fitted, for every given case, the prediction is computed as the mode of the individual predictions of every tree (but not the ones fitted with permuted variables). The importance of every variable is computed as the average of the differences in mean squared error between trees computed with the variable and trees computed with the permuted variable, normalized by the standard deviation of the differences.

Random Forest does not require any assumptions to be fulfilled by the data or the model outcomes, and therefore it can be applied to a wide range of analytic cases where data are non-linear. As a drawback, the randomness in the selection of cases and predictors to fit individual regression trees turns it into a non-deterministic algorithm, and therefore, fine-scale variations in the outcomes are to be expected between different runs with the same data.

To fit Random Forest models on the simulated data we selected the package *ranger* over the more traditional *randomForest* because the former allows multithread computing (uses all available cores in a computer while fitting the forest), achieving a better performance for large datasets than the later. The code below shows how to use *ranger* to fit a Random Forest model.

```
#getting columns containing "Response" or "Driver"
sim.lags.rf <- sim.lags[, grepl("Driver|Response", colnames(sim.lags))]

#fitting a Random Forest model
rf.model <- ranger(
  data = sim.lags.rf,
  dependent.variable.name = "Response_0",
  num.trees = 500,
  min.node.size = 5,
  mtry = 2,
  importance = "permutation",
  scale.permutation.importance = TRUE)

#model summary
print(rf.model)
```

```r
#R-squared (computed on out-of-bag data)
rf.model$r.squared

#variable importance
rf.model$variable.importance

#obtain case predictions
rf.model$predictions

#getting information of the first tree
treeInfo(rf.model, tree=1)
```

The function *ranger* has the following key arguments:

- **data**: dataframe with the variables to be included in the model.
- **dependent.variable.name**: model definition can be done in two ways, either through a formula, or through the argument *dependent.variable.name*, that names the response variable, and uses the remaining variables in the dataset as predictors, which forces us to be careful with what variables are available in the dataset.
- **num.trees**: controls number of trees generated (the default value is 500).
- **mtry**: controls the number of variables randomly selected to fit each tree. In the code above this argument is set to 2, indicating that the model only considers interactions among two predictors only on each tree. This allows to compute variable importance as independently as possible from other variables.
- **min.node.size**: minimum number of cases in a terminal node, which determines the overall resolution of the model.
- **importance**: when set to "permutation" it triggers the computation of variable importance through permutation tests.
- **scale.permutation.importance**: scales importance values computed through the permutation tests by the overall standard error.

The relationship between the response variable and the predictors can be examined through *partial dependence plots* (see **Figure 6**). A partial dependence plot is a simplified view of the inner structure of the model. Since regression trees consider interactions among variables, the prediction for any given case depends on the values of all predictors considered at the same time. Since it is not possible to generate such a representation in 2D or 3D, partial dependence plots set all variables not represented in the plot to their respective means. Therefore, partial dependence plots must be interpreted as simple approximations to the true shape of the model surface.

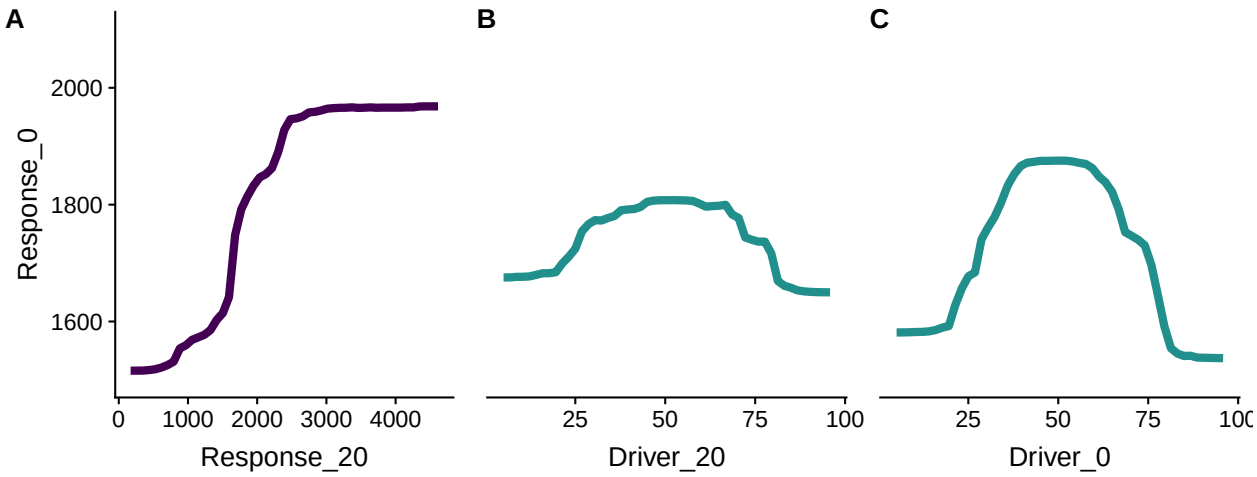

Figure 6: Partial dependence plots of the lags 20, of Pollen (A) and Driver (B), and the concurrent effect (Driver_0, panel C).

Interactions among predictors can be represented in the same way done before for recursive partition trees (see **Figure 7**). Again, all variables not represented in the plot are set to their average to generate the prediction.

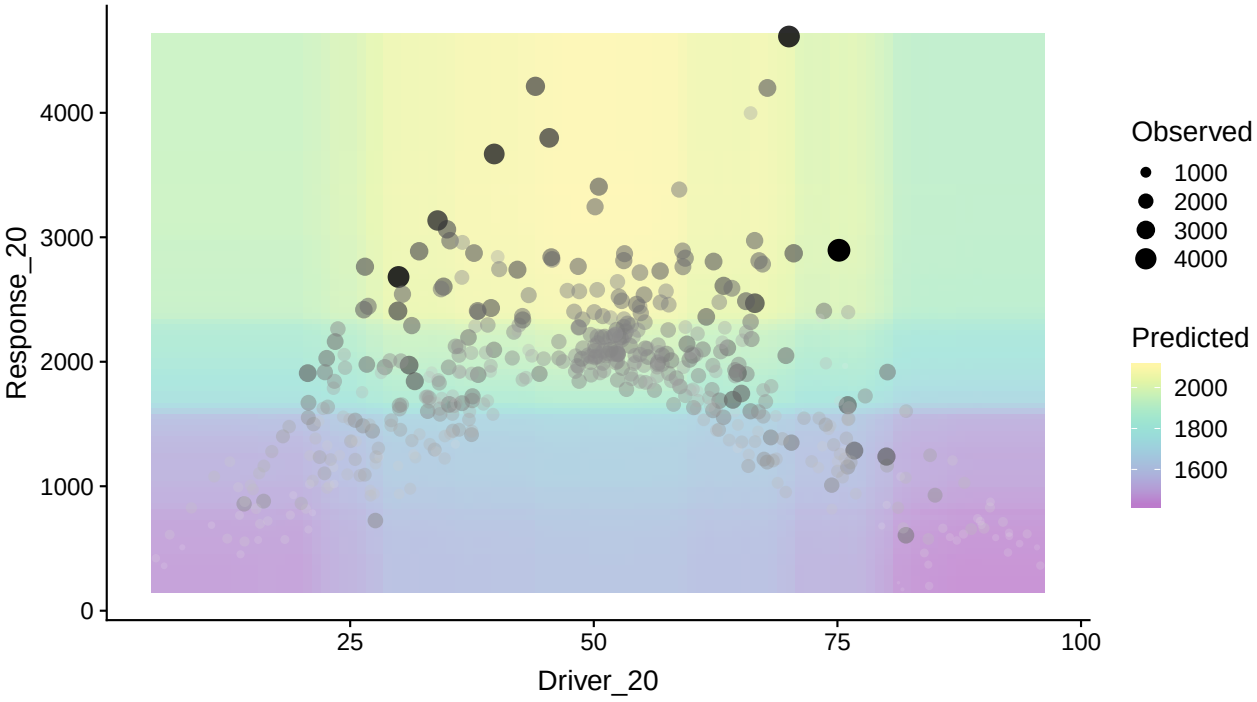

Figure 7: Interaction between Pollen_20 (first lag of the endogenous memory) and Suitability_0 (concurrent effect).

2.3 Variable importance

Random forest variable importance computation works under the assumption that if a given variable is not important, then permuting its values does not degrade the prediction accuracy. Variable importance scores are extracted with the *importance* function (see code below and **Table 4**), and are interpreted as "how much model fit degrades when the given variable is removed from the model".

```
importance(rf.model)
```

Values shown in **Table 4** are the result of one particular Random Forest run. For variables with small differences in importance, the ranking shown in the table could change in a different model run. This situation can be addressed by running the model several times, and computing the average and confidence intervals of the importance scores of each predictor across runs. This is shown in the code below (see output in **Figure 8**).

```
#number of repetitions
repetitions <- 30

#list to save importance results
importance.list <- list()

#repetitions
for(i in 1:repetitions){
  #fitting a Random Forest model
  rf.model <- ranger(
    data = sim.lags.rf,
    dependent.variable.name = "Response_0",
    mtry = 2,
    importance = "permutation",
    scale.permutation.importance = TRUE)

  #extracting importance
  importance.list[[i]] <- data.frame(t(importance(rf.model)))
}

#into a single dataframe
importance.df <- do.call("rbind", importance.list)
```

Table 3: Importance scores of a Random Forest model ordered from higher to lower importance. Importance scores are interpreted as increase in model error when the given variable is removed from the model.

Variable	Importance
Response_20	22.70
Driver_0	20.48
Driver_20	15.64
Driver_40	14.02
Response_40	13.33
Driver_100	12.69
Driver_80	12.63
Driver_60	12.40
Response_60	11.93
Driver_120	11.92
Response_80	11.21
Driver_220	10.54
Driver_240	10.39
Driver_180	10.29
Driver_140	9.49
Driver_160	9.45
Response_100	9.21
Driver_200	9.18
Response_120	8.59
Response_140	8.46
Response_160	8.17
Response_220	7.68
Response_240	7.46
Response_200	7.46
Response_180	7.06

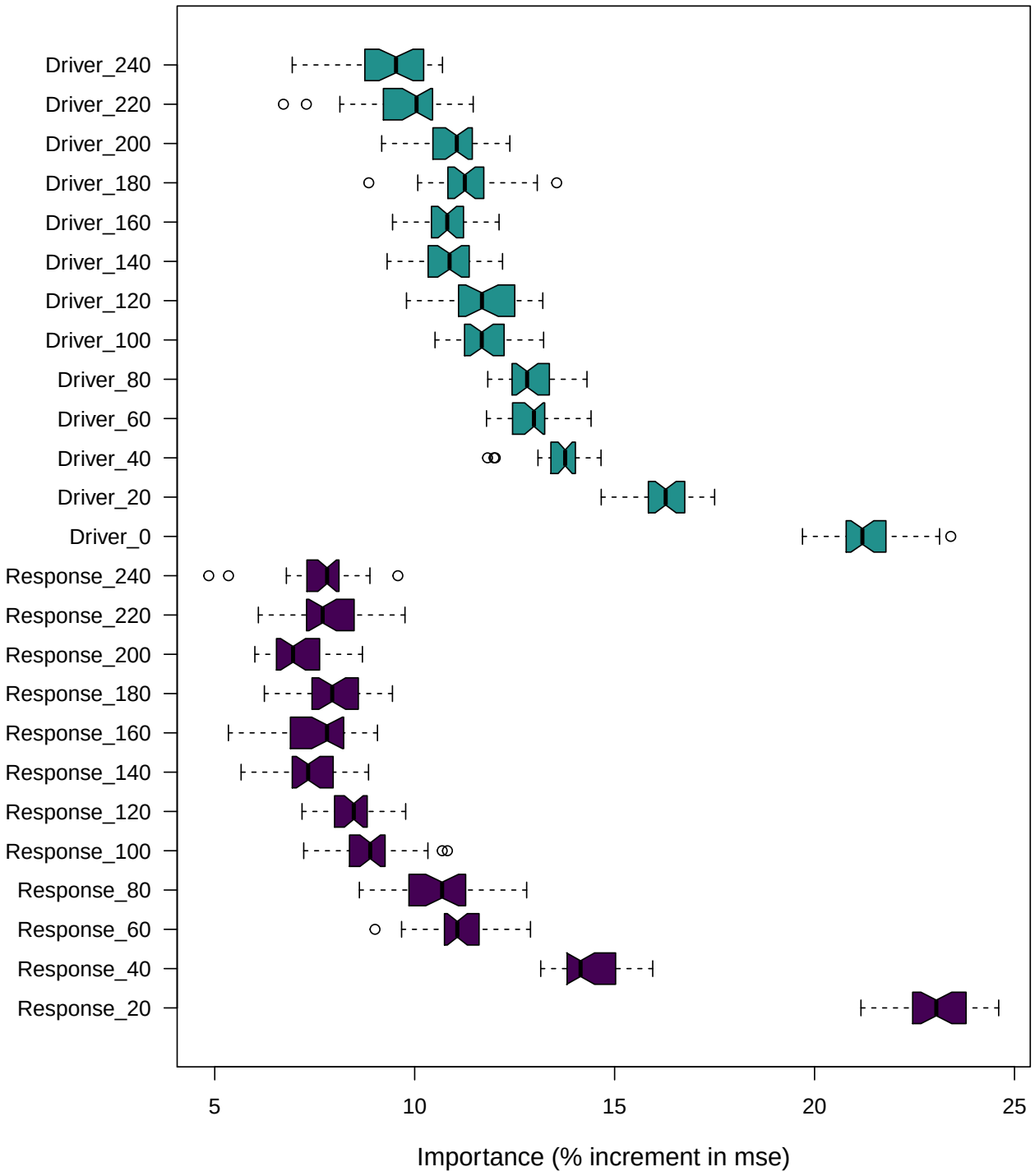

Figure 8: Importance scores of predictors in Random Forest model after 100 repetitions. Note that Response_X predictor refers to the endogenous memory, and Driver_X predictors from lag 20 refer to exogenous memory. Driver_0 represents the concurrent effect.

2.3.1 Testing the significance of variable importance scores

Random Forest does not provide any tool to assess the significance of these importance scores, and it is therefore impossible to know at what point they become irrelevant. A simple solution is to add a random variable as an additional predictor to the model and compute its importance. If the importance of other variables is equal or lower than the importance of the random variable, it can be assumed that these variables do not have a meaningful effect on the response, and can therefore be considered irrelevant.

Two types of random variables can be considered to be used as benchmarks to test variable importance scores provided by Random Forest: **white noise** (without any temporal structure), and **random walk with temporal structure** (as explained in Appendix I). In both cases the idea is to generate a *null model* providing a baseline to assess to what extent importance scores are higher than what is expected by chance. To test the suitability of both methods, the code below generates 100 Random Forest models, each one with two additional random variables: *random.white* representing white noise, and *random.autocor* representing an autocorrelated random walk. The length of the autocorrelation period of *random.autocor* is changed for every iteration.

```
#number of repetitions
repetitions <- 100

#list to save importance results
importance.list <- list()

#rows of the input dataset
n.rows <- nrow(sim.lags.rf)

#repetitions
for(i in 1:repetitions){

  #adding/replacing random.white column
  sim.lags.rf$random.white <- rnorm(n.rows)

  #adding/replacing random.autocor column
  #different filter length on each run = different temporal structure
  sim.lags.rf$random.autocor <- as.vector(filter(rnorm(n.rows),
                      filter = rep(1, sample(1:floor(n.rows/4), 1)),
                      method = "convolution",
```

```
                        circular = TRUE))

  #fitting a Random Forest model
  rf.model <- ranger(
    data = sim.lags.rf,
    dependent.variable.name = "Response_0",
    mtry = 2,
    importance = "permutation",
    scale.permutation.importance = TRUE)

  #extracting importance
  importance.list[[i]] <- data.frame(t(importance(rf.model)))

}

#into a single dataframe
importance.df <- do.call("rbind", importance.list)
```

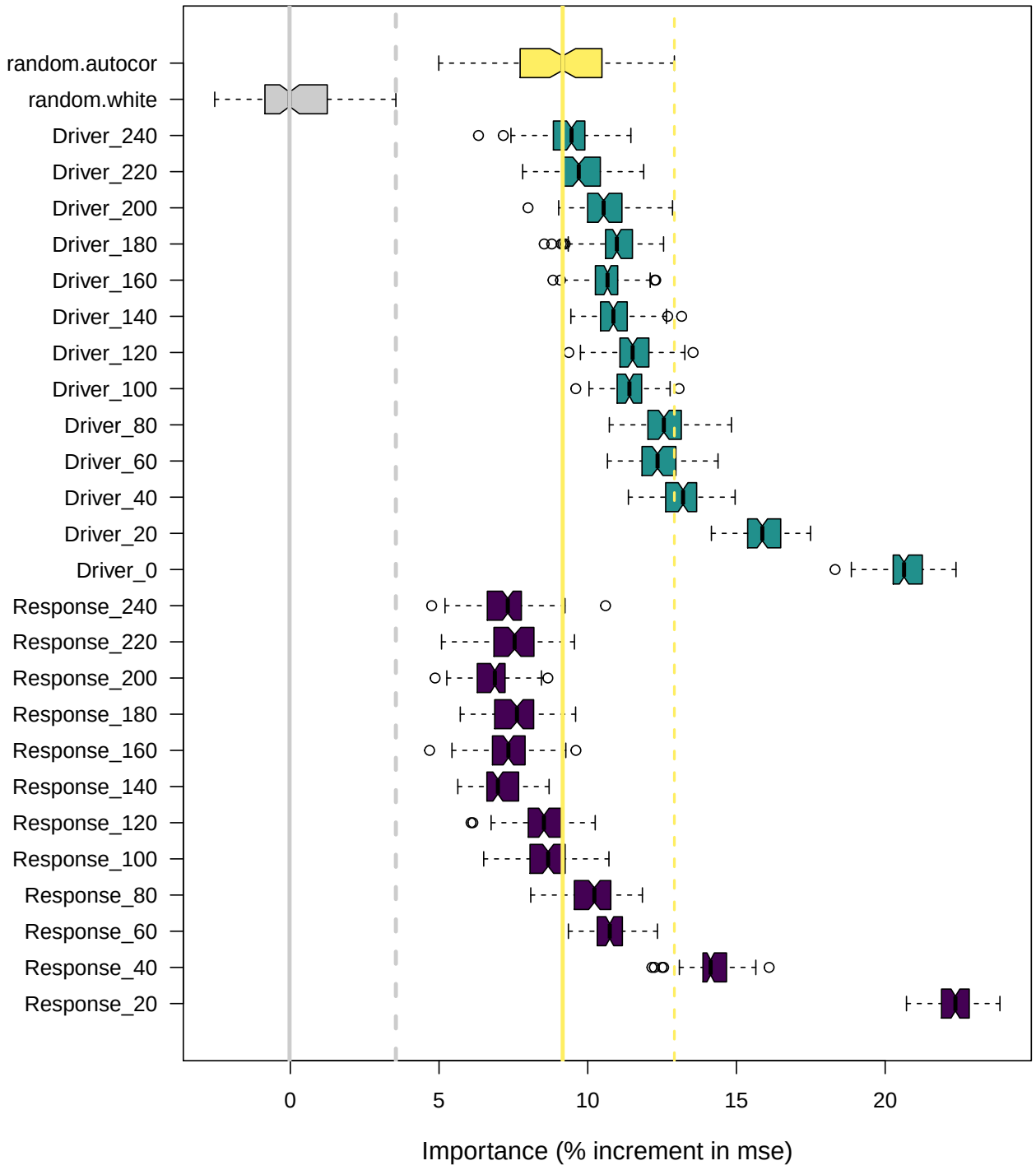

Figure 9: Importance of predictors in relation to the importance of a white noise variable (gray), and a temporally structured random variable (yellow). Solid lines represent the medians of the random variables, while dashed lines represent their maximum importance across 100 model runs.

The boxplot in **Figure 9** shows the relative importance of the random variables, and suggests that the variable representing random noise is not useful to identify importance scores arising by chance.

On the other hand, the variable based on autocorrelated random walks (marked in yellow in the plot) tells a different story. Importance values below the yellow solid line have a probability higher than 0.5 of being the result of chance. Importance values between the yellow solid and dashed lines have probabilities between 0.5 and 0 and are the result of a random association between a predictor and the response, while beyond the dashed line the results can be confidently defined as non-random. Note that **Figure 8**, when compared with **Figure 7**, also shows that adding random variables to a Random Forest model does not change the importance scores of other variables in the model.

3 Analyzing ecological memory with Random Forest

So far we have explained how to organize the simulated pollen curves in lags, and how to fit Random Forest models on the lagged data to evaluate variable importance. However, further steps are required to quantify ecological memory patterns:

- Extract and aggregate variable importance scores, and organize them in ecological memory components (endogenous, exogenous, and concurrent).
- Plot the pattern to facilitate interpretation.
- Extract ecological memory features from these components, namely **memory strength** (maximum difference in relative importance between each component (endogenous, exogenous, and concurrent) and the median of the random component), **memory length** (proportion of lags over which the importance of a memory component is above the median of the random component), and **dominance** (proportion of the lags above the median of the random term over which a memory component has a higher importance than the other component).

The function **computeMemory** fits as many Random Forest models as indicated by the argument *repetitions* on a lagged dataset, and on each iteration includes a random variable in the model. The function **plotMemory** gets the output of **computeMemory** and plots it, while the function *extractMemoryFeatures* computes the features of each ecological memory component. The simplified workflow is shown below.

```
#computes ecological memory pattern
memory.pattern <- computeMemory(lagged.data = sim.lags,
                                drivers = "Driver",
                                random.mode="autocorrelated",
                                repetitions=30,
                                response="Response")

#computing memory features
```

```
memory.features <- extractMemoryFeatures(analysis.output=memory.pattern,
                                         exogenous.component="Driver",
                                         endogenous.component="Response")

#plotting the ecological memory pattern
plotMemory(memory.pattern)
```

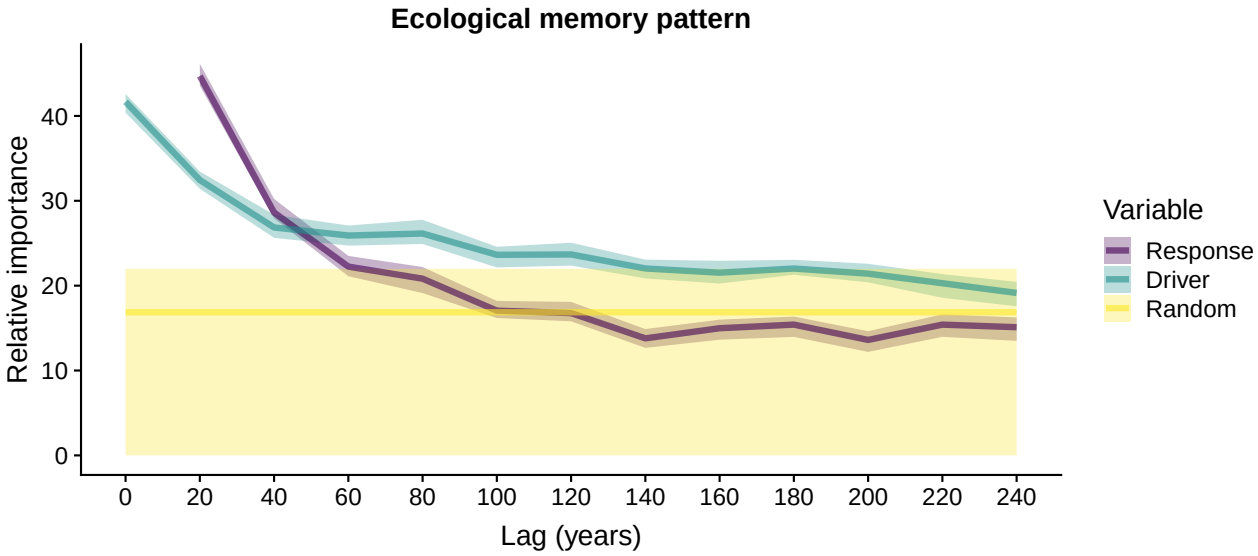

Figure 10: Ecological memory pattern of taxon 6, 1cm dataset. The variable Response represents the endogenous memory, the first lag of the variable Suitability represents the concurrent effect of environmental conditions, and the rest of the lags of Suitability represent the exogenous memory component. Relative importance values below the yellow line are considered non-significant.

Table 4: Features of the ecological memory pattern shown in Figure 10 by using the extractMemoryFeatures function.

strength.endogenous
strength.exogenous
strength.concurrent
length.endogenous
length.exogenous
dominance.endogenous
dominance.exogenous

In order to analyze the ecological memory patterns of 16 virtual taxa across the 5 levels of data quality

(Annual, 1cm, 2cm, 6cm, and 10cm), we integrated the functions above into a larger function named *runExperiment*. The code below runs an artificial simple example with only two virtual taxa (1 and 6 in *parameters*), and two dataset types ("1cm" and "10cm"). Only 30 repetitions are generated to quicken execution, which is not nearly enough to achieve consistent results.

```
#running experiment
E1 <- runExperiment(simulations.file = simulation,
                    selected.rows = c(1, 6), #2 species only
                    selected.columns = c(2, 5), #1cm dataset only
                    parameters.file = parameters,
                    parameters.names = c("maximum.age",
                                         "fecundity",
                                         "niche.A.mean",
                                         "niche.A.sd"),
                    sampling.names = c("1cm", "10cm"),
                    driver.column = "Driver.A",
                    response.column = "Pollen",
                    time.column = "Time",
                    lags = lags,
                    repetitions = 30)

#E1 is a list of lists
#first list: names of experiment output
E1$names

#second list, first element
i <- 1 #change to see other elements
#ecological memory pattern
E1$output[[i]]$memory
#pseudo R-squared across repetitions
E1$output[[i]]$R2
#predicted pollen across repetitions
E1$output[[i]]$prediction
#variance inflation factor of input data
E1$output[[i]]$multicollinearity
```

Experiment results can be plotted with the function *plotExperiment* shown below.

```
plotExperiment(experiment.output=E1,
                    parameters.file=parameters,
```

```
                    experiment.title="Toy experiment",
                    sampling.names=c("1cm", "10cm"),
                    legend.position="bottom",
                    R2=TRUE)
```

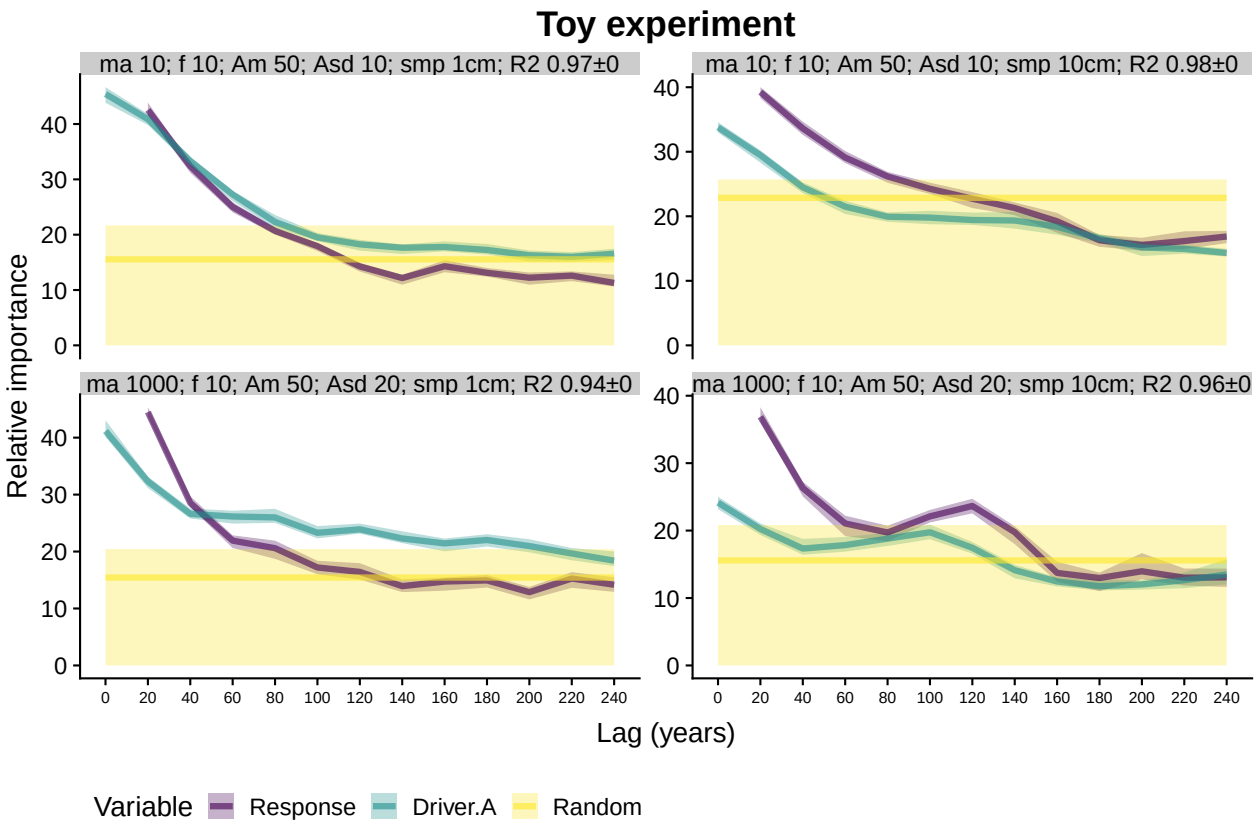

Figure 11: Ecological memory patterns of 2 virtual taxa (one per row, taxa 1 and 6) at two different sampling resolutions (one per column: 1cm and 10cm). Abbreviations in plot title: ma - maximum age, f - fecundity, Am - niche mean, Asd - niche breadth, smp - sampling resolution, R2 - pseudo R-squared.

The experiment data can be organizes as a single table, joined with the data available in the *parameters* dataframe to facilitate further analyses.

```
E1.df <- experimentToTable(experiment.output=E1,
                           parameters.file=parameters,
                           sampling.names=c("1cm", "10cm"),
                           R2=TRUE)
```

Table 5: First rows of the experiments table.

median	sd	min	max	Variable	Lag	R2mean	R2sd	VIFmean	VIFsd	label	maximum.age	reproductive.age	fecundity	growth.rate	driver.A.weight	driver.B.weight	niche.A.mean	niche.A.sd	niche.B.mean	niche.B.sd	sampling
42.6	0.70	0	0	Response	20	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
32.4	0.55	0	0	Response	40	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
25.0	0.54	0	0	Response	60	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
20.7	0.43	0	0	Response	80	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
17.9	0.45	0	0	Response	100	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
14.3	0.55	0	0	Response	120	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
12.2	0.59	0	0	Response	140	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
14.3	0.67	0	0	Response	160	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
13.1	0.50	0	0	Response	180	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
12.2	0.71	0	0	Response	200	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
12.6	0.61	0	0	Response	220	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
11.3	0.70	0	0	Response	240	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
45.4	0.89	0	0	Driver.A	0	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
40.9	0.62	0	0	Driver.A	20	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
33.3	0.51	0	0	Driver.A	40	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
27.3	0.56	0	0	Driver.A	60	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
22.3	0.75	0	0	Driver.A	80	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
19.5	0.53	0	0	Driver.A	100	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
18.3	0.67	0	0	Driver.A	120	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
17.6	0.56	0	0	Driver.A	140	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
17.8	0.67	0	0	Driver.A	160	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
17.2	0.60	0	0	Driver.A	180	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
16.3	0.68	0	0	Driver.A	200	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
16.0	0.61	0	0	Driver.A	220	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
16.6	0.69	0	0	Driver.A	240	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	20	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	40	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	60	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	80	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	100	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	120	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	140	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	160	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	180	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	200	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	220	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	240	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
15.5	4.87	0	0	Random	0	0.97	0	80.19685	72.96249	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	1cm
39.2	0.59	0	0	Response	20	0.98	0	459.73097	367.61615	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	10cm
33.6	0.65	0	0	Response	40	0.98	0	459.73097	367.61615	S10A50-5_f10	10	4	10	1.5	1	0	50	10	50	5	10cm

Finally, ecological memory features can be extracted from the experiment with *extractMemoryFeatures* in order to facilitate further analyses, as shown below.

```
E1.features <- extractMemoryFeatures(analysis.output=E1.df,
                                     exogenous.component="Driver.A",
                                     endogenous.component="Response")
```

Table 6: Features of the ecological memory patterns produced by the example experiment.

label	S10A50-5_f10	S10A50-5_f10	S1000A50-20_f10	S1000A50-20_f10
strength.endogenous	27.1	16.3	29.1	21.3
strength.exogenous	25.4	6.6	16.9	4.6
strength.concurrent	29.9	10.9	25.8	8.5
length.endogenous	0.417	0.417	0.500	0.583
length.exogenous	1.000	0.167	1.000	0.500
dominance.endogenous	0.083	0.417	0.167	0.583
dominance.exogenous	0.917	0.000	0.833	0.000
maximum.age	10	10	1000	1000
fecundity	10	10	10	10
niche.mean	50	50	50	50
niche.sd	10	10	20	20
sampling	1cm	10cm	1cm	10cm