Fiche TD avec le logiciel R : tdRBM

UE Ecologie et Gestion des Populations - TD Structures Temporelles

Quels sont les mécanismes de la stratégie de fructification appelée « masting » ?

A. Siberchicot, M.-C. et S. Venner & J.R. Lobry

1 Définir le voisinage d'un arbre : explorer la fonction `voisinsMoore()`

À ce jour, R met à la disposition des utilisateurs plus de 12000 paquets (*packages*) que l'on peut charger dans son environnement de travail. Chaque paquet contient lui-même un ensemble de fonctions et de jeux de données, développés par des personnes comme vous et moi et mis à la disposition de la communauté R.

La fonction `voisinsMoore()` n'appartient à aucun paquet. Elle a été créée spécialement pour ce TD et son code est dans le fichier `fonctions.R` que vous pouvez charger de la façon suivante :

```
source("https://pbil.univ-lyon1.fr/R/donnees/tdRBM/fonctions.R")
```

Vous avez maintenant dans votre environnement de travail la fonction `voisinsMoore()` qui est définie, ainsi que toutes les fonctions utilisées dans ce TD :

```
ls()
```

```
[1] "calculeIndicMasting"  "plot.voisins"         "plotDynMasting"
[4] "plotDynMasting1"      "plotDynMastingPollen" "simu"
[7] "voisinsMoore"
```

La fonction `voisinsMoore()` permet de déterminer pour chaque arbre d'une population, les individus-arbres qui constituent son voisinage. La population d'arbre est modélisée par une grille où chaque arbre a pour coordonnée le numéro de sa ligne et le numéro de sa colonne dans la grille. Dans le cadre du masting, on considérera toujours une grille avec autant de colonnes que de lignes, carrée donc.

La fonction `voisinsMoore()` contient le nom du mathématicien américain E.F. Moore (1925-2003) pionnier de l'informatique et des systèmes automatisés. Il a notamment proposé une méthode pour définir le voisinage de chaque cellule dans une grille : ici, un arbre est dans le voisinage d'un autre si la distance qui les sépare dans la grille est inférieure ou égale à une valeur D, exprimée avec comme unité le nombre de cellules. Par exemple, on peut calculer le voisinage pour une distance de Moore de 1 dans une population modélisée par une grille de 5×5.

```
voisins_5_1 <- voisinsMoore(dimR = 5, dimC = 5, dist = 1)
```

La fonction `voisinsMoore()` a donc trois paramètres d'entrée qu'il faut renseigner pour l'exécuter :

1° `dimR`, le nombre de lignes (*rows*) dans la grille qui modélise la population d'arbres ;

2° `dimC`, le nombre de colonnes (*columns*) dans la grille qui modélise la population d'arbres ;

3° `dist`, la distance D que l'on considère pour le calcul du voisinage.

On décide de sauvegarder/stocker/enregistrer le résultat dans une variable que l'on nomme avec un nom assez explicite pour se souvenir ce qu'il y a dedans : `voisins_5_1`. Vous auriez pu l'appeler `toto` mais si vous passez votre code à un ami ou que vous revenez vous-même sur ce code dans plusieurs semaines, il y a de grande chance pour que vous ne sachiez plus ce qu'il y a dedans ! Maintenant, explorons cet objet `voisins_5_1`. `voisins_5_1` est une liste :

```
class(voisins_5_1)
[1] "list"
is.list(voisins_5_1)
[1] TRUE
```

On voit qu'il y a autant d'éléments dans la liste `voisins_5_1` qu'il y a de lignes dans la grille d'arbres considérée au départ :

```
length(voisins_5_1) # retourne le nombre d'éléments dans la liste
[1] 5
```

Chaque élément de l'objet `voisins_5_1` (qui est une liste) est lui-même une liste :

```
class(voisins_5_1[[1]]) # retourne la classe du premier élément de la liste voisins_5_1
[1] "list"
is.list(voisins_5_1[[1]])
[1] TRUE
```

Cette liste contient autant d'éléments qu'il y a de colonnes dans la grille d'arbres considérée au départ :

```
length(voisins_5_1[[1]])
[1] 5
```

Chaque élément de chaque élément de l'objet `voisins_5_1` est une matrice :

```
class(voisins_5_1[[1]][[1]])
[1] "matrix" "array"
is.list(voisins_5_1[[1]][[1]])
[1] FALSE
is.matrix(voisins_5_1[[1]][[1]])
[1] TRUE
```

Cette matrice a :

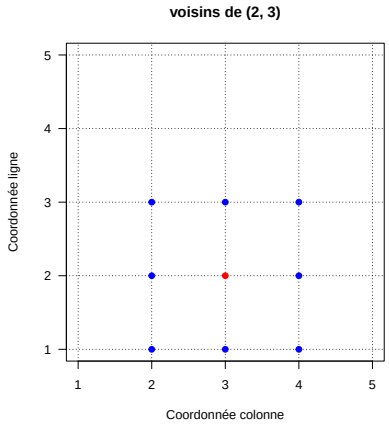
1° autant de lignes qu'il y a d'arbres dans le voisinage de Moore d'un individu-arbre, dans la population d'arbres considérée au départ ;

2° deux colonnes, nommées `numLigne` et `numColonne`.

```
dim(voisins_5_1[[1]][[1]])
[1] 8 2
dim(voisins_5_1[[2]][[1]])
[1] 8 2
dim(voisins_5_1[[1]][[2]])
[1] 8 2
colnames(voisins_5_1[[1]][[1]])
[1] "numLigne"   "numColonne"
voisins_5_1[[1]][[1]]
     numLigne numColonne
[1,]        1          2
[2,]        1          5
[3,]        2          1
[4,]        2          2
[5,]        2          5
[6,]        5          1
[7,]        5          2
[8,]        5          5
```

La fonction `plot.voisins()` permet de visualiser le voisinage d'un individu donné. Par exemple, pour l'arbre à la 2e ligne et à la 3e colonne :

```
plot.voisins(voisins_5_1, 2, 3)
```



1.1 Questions :

1° Dans une population simulée par une grille de 5 arbres par 5 arbres, combien de voisins, au sens de Moore et pour une distance de 1, possède un individu-arbre ?

2° Dans cette même population, donnez les coordonnées de tous les voisins de l'arbre (1) situé à la 3e ligne et à la 3e colonne puis de l'arbre (2) situé à la 2e ligne et à la 5e colonne.

3° Vérifiez avec la fonction `plot.voisins()` chez ces deux arbres que les coordonnées de leurs voisins correspondent bien aux cellules colorées en bleu.

4° Pour l'arbre situé en 2e ligne / 5e colonne, quel mécanisme fait que les cellules apparaissent en bleu dans la 1re colonne? Quel est l'intérêt de définir ainsi le voisinage ?

5° Considérons maintenant une population de 10 arbres par 10. Combien de voisins a un individu-arbre dans cette population en considérant un voisinage de Moore de 1 ? Un voisinage de Moore de 2 ?

1.2 Réponses :

```
# Question 1
nrow(voisins_5_1[[1]][[1]])
```

```
[1] 8
```

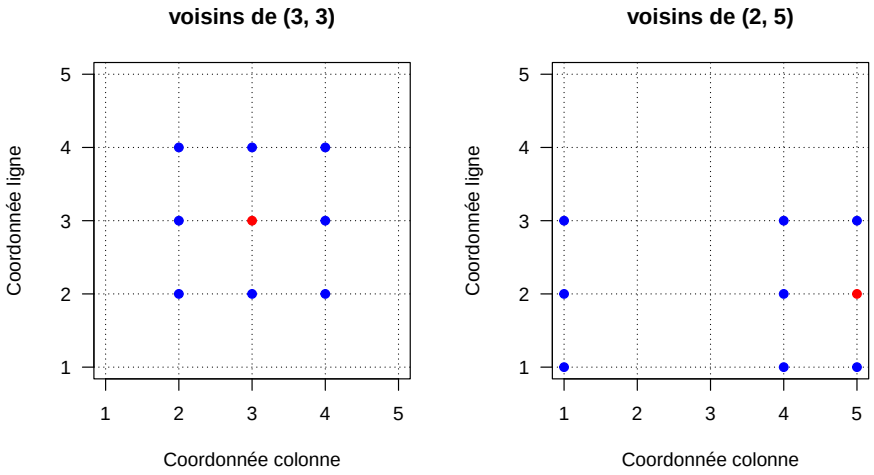
```
# Question 2
voisins_5_1[[3]][[3]]
```

```
     numLigne numColonne
[1,]        2          2
[2,]        2          3
[3,]        2          4
[4,]        3          2
[5,]        3          4
[6,]        4          2
[7,]        4          3
[8,]        4          4
```

```
voisins_5_1[[2]][[5]]
```

```
     numLigne numColonne
[1,]        1          1
[2,]        1          4
[3,]        1          5
[4,]        2          1
[5,]        2          4
[6,]        3          1
[7,]        3          4
[8,]        3          5
```

```
# Question 3
par(mfrow = c(1, 2))
plot.voisins(voisins_5_1, 3, 3)
plot.voisins(voisins_5_1, 2, 5)
# 4
# C'est le principe du tore (voir ici l'article sur Wikipédia)
```




```
# Question 5
voisins_10_1 <- voisinsMoore(dimR = 10, dimC = 10, dist = 1)
nrow(voisins_10_1[[1]][[1]])
```

```
[1] 8
```

```
voisins_10_2 <- voisinsMoore(dimR = 10, dimC = 10, dist = 2)
nrow(voisins_10_2[[1]][[1]])
```

```
[1] 24
```

2 Simuler la production de pollen et de glands d'une population de chênes : explorer la fonction `simu()`

De la même manière que la fonction `voisinsMoore()`, la fonction `simu()` a été créée spécifiquement pour ce TD et son code est contenu dans le fichier `fonctions.R` que vous avez déjà chargé. La fonction `simu()` possède de nombreux arguments qui peuvent être étudiés pour simuler différentes populations d'arbres. Les arguments de la fonction `simu()` sont :

```
args(simu)
```

```
function (dc, size, voisins, sigmaEnvNoise, syEnvNoise, cyclesnumber,
    timememory, meteo = FALSE, pollcross = 0)
NULL
```

1° `dc` est le coefficient de déplétion c'est-à-dire `coût fruit = (dc + 1)*coût fleur`. Il est strictement positif et peut varier de 0 à l'infini, en pratique jusqu'à 25.

2° `size` est la taille de la grille, c'est-à-dire le nombre d'individus-arbres en ligne (ou en colonne, puisque la grille considérée est carrée). Attention, plus la population simulée (la grille) est grande, plus le temps de simulation sera long.

3° `voisins` est la structure qui définit le voisinage préalablement calculée pour tous les arbres d'une grille par la fonction `voisinsMoore()`. Attention, la taille de la population étudiée doit être cohérente entre `voisins` et l'argument `size`. Vous ne pouvez pas simuler une population que vous définissez à 10 par 10 (`size = 10`) et donner dans l'argument voisins une structure calculée sur une population de taille différente (`voisins_5_1` par exemple).

4° `sigmaEnvNoise` définit la racine carrée de la variance environnementale totale. Ce paramètre influence notamment l'écart au gain annuel moyen d'énergie par photosynthèse pour un arbre.

5° `syEnvNoise` définit l'effet Moran (coefficient de synchronie entre les arbres dans leur gain d'énergie). Il peut varier entre 0 et 1.

6° `cyclesnumber` est le nombre d'années (l'année étant le pas de temps) pendant lesquelles la simulation de la population d'arbres est réalisée. Attention, plus ce paramètre est grand, plus le temps de calcul sera évidemment élevé.

7° `timememory` est le nombre d'années que l'on conserve dans les sorties de résultats. En effet, même si la simulation est effectuée sur 2000 ans (`cyclesnumber = 2000`), on ne retiendra que les `timememory` dernières années de la simulation pour caractériser le masting. On procède ainsi pour éviter d'inclure les premières années dont les résultats dépendent étroitement des conditions initiales de simulation, lesquelles sont en partie arbitraires.

8° `meteo` est un booléen (`TRUE` ou `FALSE`) qui permet d'indiquer si on veut ou non intégrer l'influence des conditions météorologiques dans le modèle. Ces conditions sont en réalité susceptibles d'affecter la diffusion du pollen produit et donc le succès de pollinisation des fleurs femelles. Pour une question de simplicité, cependant, on ne tiendra pas compte dans ce travail de TD de cet effet (`meteo = FALSE`).

9° `pollcross` vaut 0, 1, 2 ou 3. Ces valeurs correspondent à une fonction de pollinisation croisée, celle qui relie la probabilité pour une fleur femelle de donner un fruit (variable `y`) à la quantité de pollen produite par les voisins (variable `x`) de l'arbre considéré. Précisément :

   - ⋆ `pollcross = 0` considère qu'il n'y a pas de pollinisation croisée ; le pollen n'est pas limitant.
   - ⋆ `pollcross = 1` modélise une pollinisation croisée de la forme : `y = x`.
   - ⋆ `pollcross = 2` modélise une pollinisation croisée de la forme : `y = 1 / (1 + 200 * exp(-22*x))`.
   - ⋆ `pollcross = 3` modélise une pollinisation croisée de la forme : `y = 1 / (1 + 1000 * exp(-12*x))`.

2.1 Questions :

À titre d'exemple et toujours en considérant une population modélisée par une grille de 5 par 5, on simule la dynamique de production pollinique et fruitière de 25 arbres (5 * 5) pendant 1000 années, en ne retenant que les 10 dernières, avec `dc = 2, sigmaEnvNoise = 0.2 et syEnvNoise = 0.5`. Le résultat est stocké dans un objet nommé `simu1`.

```
simu1 <- simu(dc = 2, size = 5, voisins = voisins_5_1, sigmaEnvNoise = 0.2,
              syEnvNoise = 0.5, cyclesnumber = 1000, timememory = 10,
              meteo = FALSE, pollcross = 0)
```

1° Que contient l'objet `simu1` ? Quelle est sa taille ?

2° Que contient le premier élément de `simu1` ? Quelle est sa taille ? A quoi correspondent les lignes de cet élément ? Et ses colonnes ?

3° Que contient le deuxième élément de `simu1` ? Quelle est sa taille ? A quoi correspondent les lignes de cet élément ? Et ses colonnes ?

2.2 Réponses

```
# Question 1
str(simu1) ; class(simu1) ; length(simu1) ; names(simu1) ; dim(simu1$outputFruits);
# Question 2
class(simu1$outputPollen) ; dim(simu1$outputPollen) ; nrow(simu1$outputPollen) ;
ncol(simu1$outputPollen) ;
# Question 3
class(simu1$outputFruits) ; dim(simu1$outputFruits) ; nrow(simu1$outputFruits) ;
ncol(simu1$outputFruits) ;
```

3 Les indicateurs du masting : explorer la fonction `calculeIndicMasting()`

La fonction `calculeIndicMasting()` est également une fonction définie dans le fichier `fonctions.R`. Cette fonction ne prend qu'un argument en entrée :

```
args(calculeIndicMasting)
```

```
function (matTempsArbres)
NULL
```

L'argument `matTempsArbres` doit être un des deux éléments issus d'une simulation effectuée avec la fonction `simu()`. À partir des résultats de simulation de production pollinique et fruitière d'une population d'arbres, cette fonction calcule trois grandeurs statistiques qui sont classiquement utilisées pour mesurer le masting :

1° le coefficient de variation individuel (CVi), qui est la moyenne sur tous les arbres de leur coefficient de variation individuel de production fruitière (le coefficient de variation est égale à la division de l'écart-type par la moyenne) ;

2° la synchronie (Sy) qui est la moyenne des coefficients de corrélation de Pearson de la production fruitière entre tous les arbres 2 à 2 ;

3° le coefficient de variation populationnel (CVp) qui est le coefficient de variation des moyennes annuelles de production fruitière de tous les arbres.

3.1 Questions :

1° En reprenant l'exemple précédent, comment obtenir les indicateurs du masting sur les productions polliniques ?

2° En reprenant l'exemple précédent, comment obtenir les indicateurs du masting sur les productions fruitières ?

3° Quel est le niveau de synchronie des dynamiques polliniques de la population étudiée ? Quel est celui des dynamiques fruitières ?

4° Lequel des trois indicateurs statistiques calculés par la fonction calculeIndicMasting résume le mieux les caractéristiques du masting ?

3.2 Réponses :

```
# Question 1.
statPollen <- calculeIndicMasting(simu1$outputPollen)
# Question 2.
statFruits <- calculeIndicMasting(simu1$outputFruits)
# Question 3.
statPollen$sy # ou statPollen[3]
statFruits$sy # ou statFruits[3]
# Question 4.
```

4 Représenter graphiquement le masting : explorer les fonctions `plotDynMasting()` et `plotDynMastingPollen()`

Les fonctions `plotDynMasting()` et `plotDynMastingPollen()` sont également définies dans le fichier `fonctions.R`. Ces fonctions graphiques vous permettent de représenter facilement les dynamiques individuelles du pollen ou des fruits, sur les années retenues d'une simulation. Elles prennent en entrée deux arguments, une matrice des dynamiques individuelles (un des deux éléments issus d'une simulation effectuée avec la fonction `simu()`) et le nombre de trajectoires individuelles que l'on souhaite représenter :

```
args(plotDynMasting)
```

```
function (matTempsArbres, nbArbres)
NULL
```

```
args(plotDynMastingPollen)
```

```
function (matTempsArbres, nbArbres)
NULL
```

4.1 Questions :

1° Représentez les dynamiques fruitières de tous les arbres de la population simulée dans l'exemple précédent.

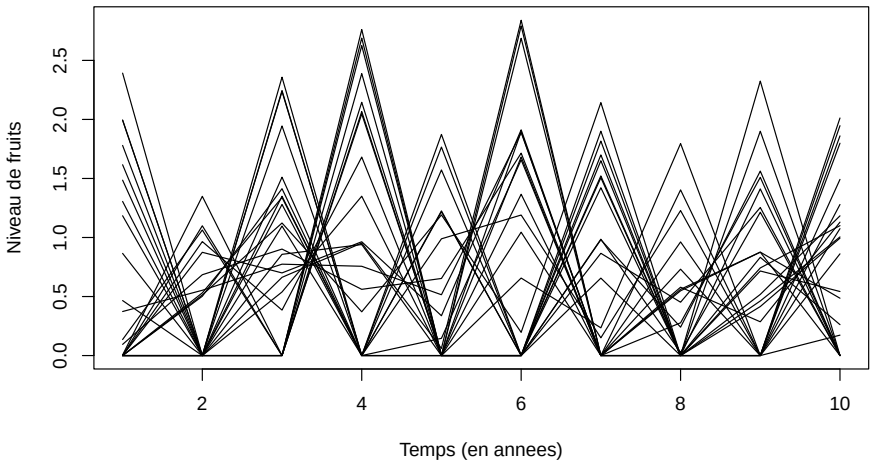
2° Est-ce que le graphique est le même que celui de votre voisin de TD ? Expliquez pourquoi ?

3° Représentez les dynamiques polliniques de 10 arbres issus de la même population d'arbres.

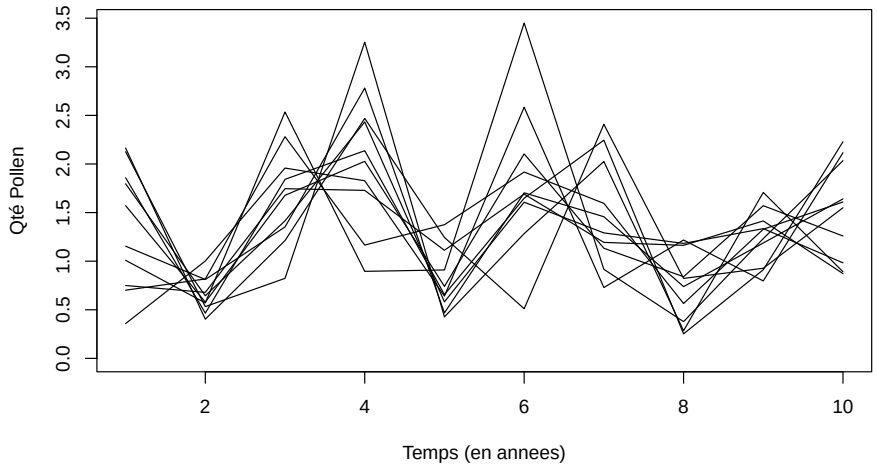
4° Relancez exactement la même opération qu'à la question précédente. Est-ce que vous observez les mêmes graphes ? A votre avis, pourquoi ?

4.2 Réponses :

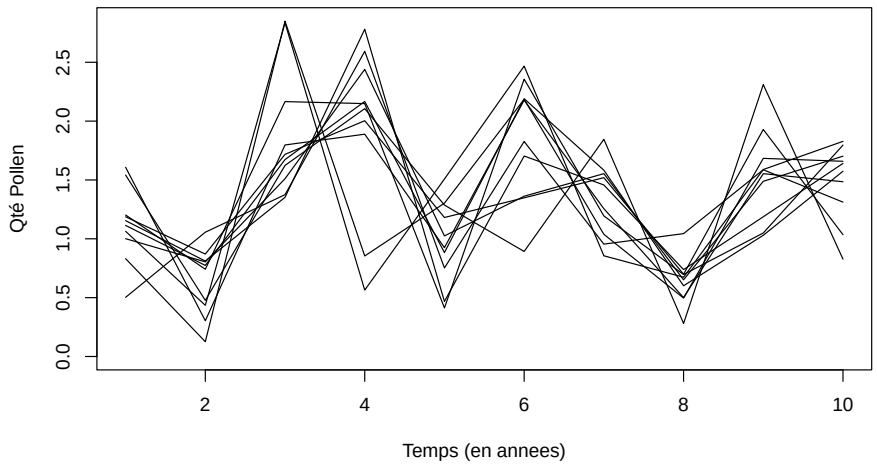
```
# Question 1
plotDynMasting(simu1$outputFruits, ncol(simu1$outputFruits))
# Question 2: non
```




```
# Question 3
plotDynMastingPollen(simu1$outputPollen, 10)
```




```
# Question 4
plotDynMastingPollen(simu1$outputPollen, 10)
```



5 Résumé de la marche à suivre pour modéliser une population caractérisée par un phénomène de masting

Voici les étapes présentées dans les paragraphes précédents et l'ordre nécessaire pour réaliser une étude de simulation du masting :

1° définir le voisinage de tous les individus d'une population (→ fonction `voisinsMoore()`) ;

2° simuler une population d'arbres caractérisée par le masting (→ fonction `simu()`) ;

3° Analyse statistique des résultats (→ fonction `calculeIndicMasting()`) et fonctions graphiques.