Séance 5

Algorithmique II

BIO109 - Dominique Gravel

Laboratoire d'écologie intégrative | Integrative Ecology Lab [IE]

Solution

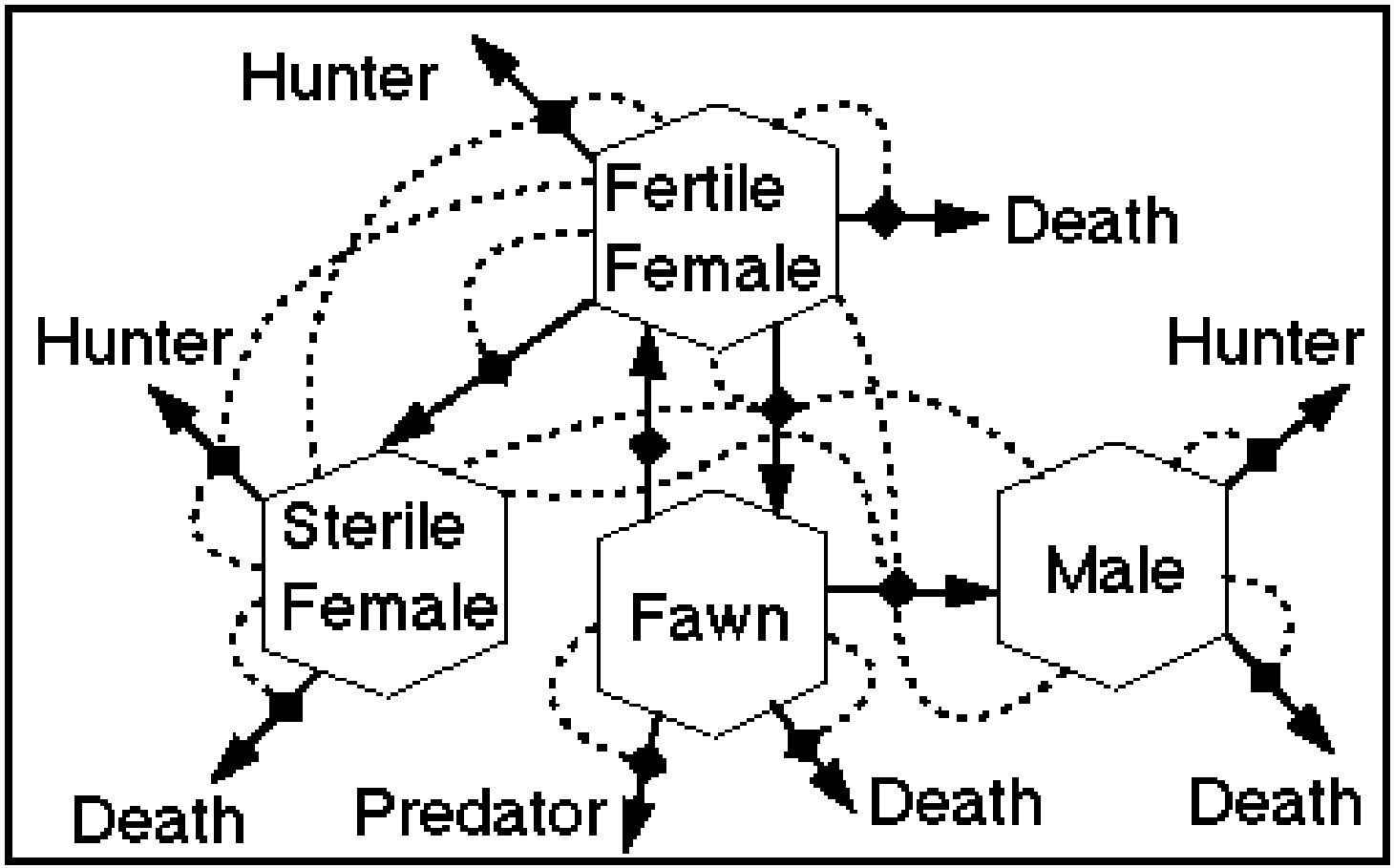
```
tri = function(x){
    # Calcul de la dimension du vecteur
    taille = length(x)
    ordre = "NON"
    # Boucle qui tourne jusqu'à ce que tout soit en ordre
    while(ordre == "NON") {
        ordre = "OUI"
        # Boucle qui passe tous les éléments en paires
        for(i in 1:(taille-1)) {
            if(x[i+1] < x[i]) {
                # Inversion des deux lettres
                x[c(i,i+1)] = x[c(i+1,i)]
                # Comme un changement a été fait, l'ordre
                # n'est pas encore garanti
                ordre = "NON"
            }
        }
    }
    return(x)
}
```

Les jeux de hasard

Les jeux de hasard

Le hasard et l'écologie

Les populations structurées par la taille

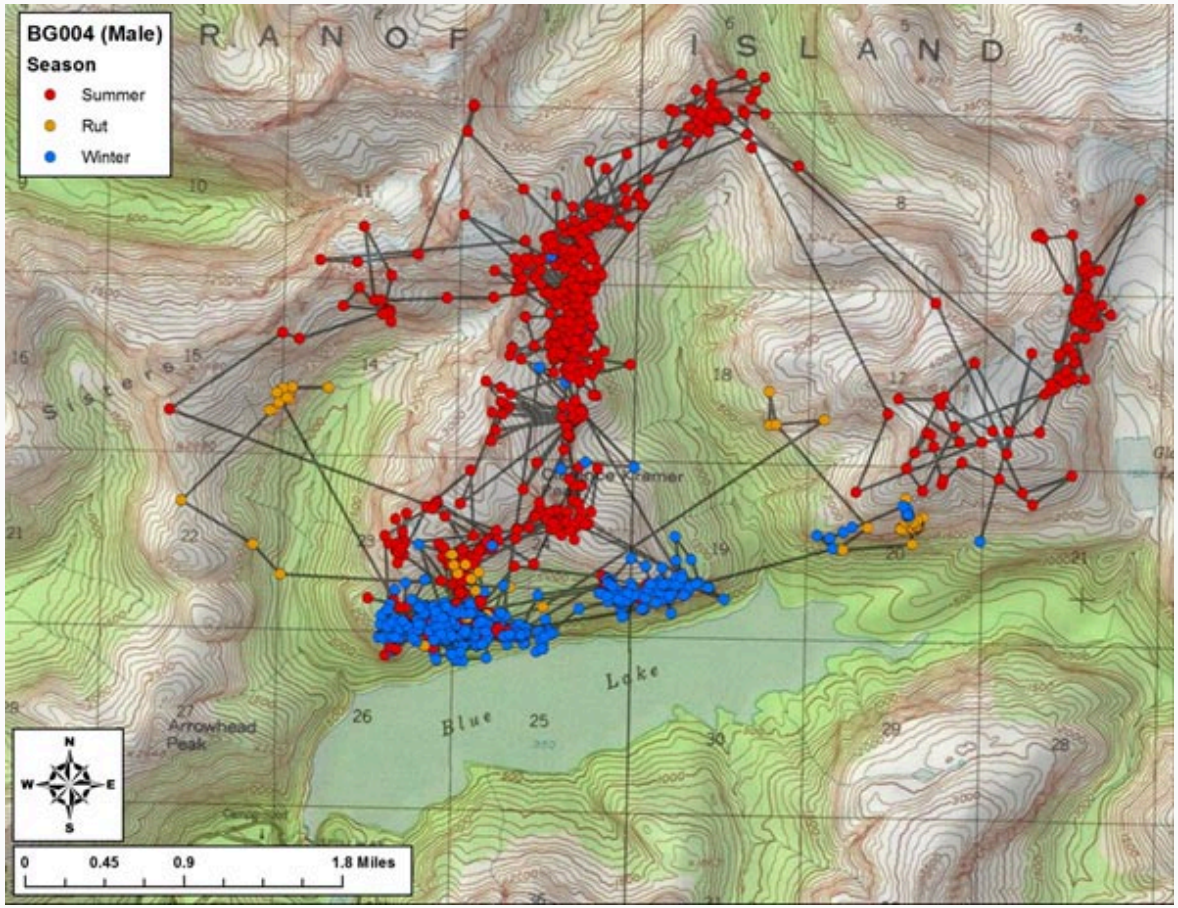


Le hasard et l'écologie

Écologie du paysage

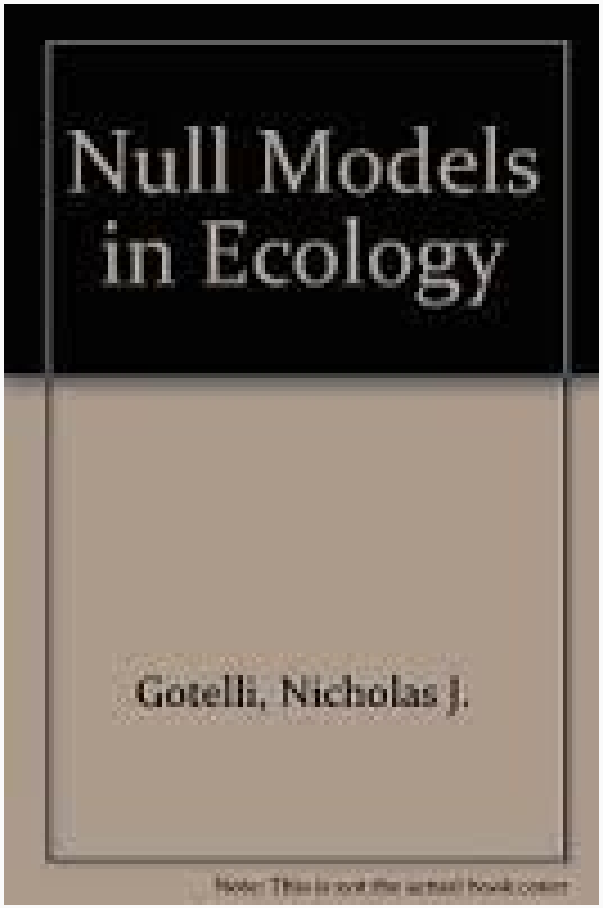
Le hasard et l'écologie

Mouvement des individus



Le hasard et l'écologie

Statistiques



Application 1: échantillonner un vecteur

Exemple : tirer une carte au hasard

Exemple : votre jeu de cartes sur R

La fonction `sample()`

```
valeurs ← c("2", "3", "4", "5", "6", "7", "8", "9", "10", "valet", "reine", "roi",
couleurs ← rep(c("pique", "trèfle", "carreau", "coeur"), each = 13)
cartes ← paste(valeurs, "-", couleurs)
tirage ← function(n, cartes) {
    sample(x = cartes, size = n, replace = FALSE)
}
tirage(3, cartes)
```

```
## [1] "5 - coeur"   "4 - carreau" "4 - coeur"
```

Exercice

Vous trouverez les lettres du scrabble dans le fichier lettres.txt. Vous pouvez les charger et programmer votre fonction qui pigera au hasard les 7 lettres pour ce jeux.

Application 2: échantillonner une loi de probabilité

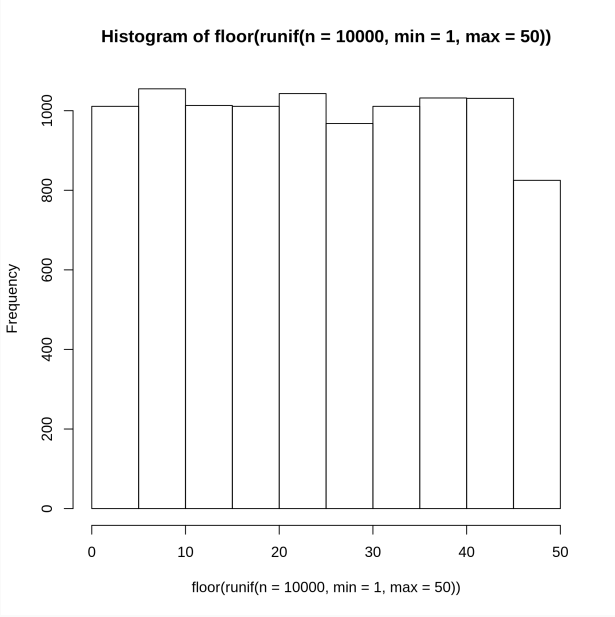
Principe

Plutôt que de tirer un élément d'une série de chiffres ou de caractères, on peut échantillonner une loi de probabilité dont les propriétés sont connues. Fort heureusement, la plupart de ces lois de probabilité sont déjà programmées. Certaines sont intuitives et déjà utilisées dans des jeux de hasard.

La lotto 6/49

La loi uniforme

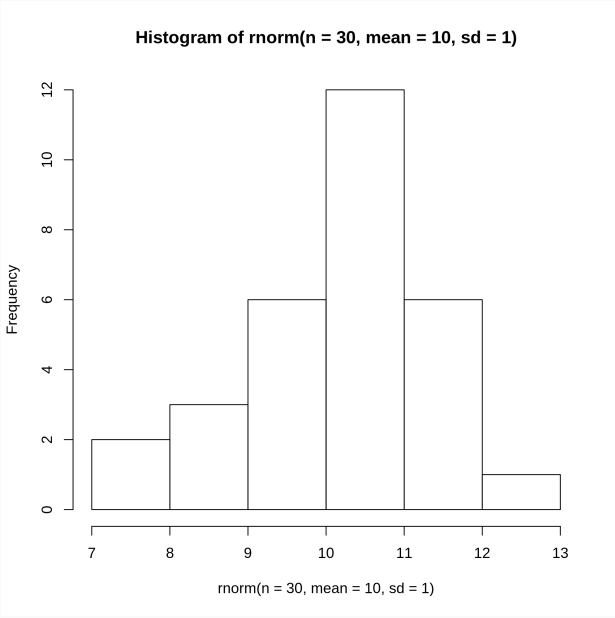

```
hist(floor(runif(n = 10000, min = 1,
  max = 50)))
```



Le jeu de la courte paille

La loi normale

```
hist(rnorm(n = 30, mean = 10, sd = 1))
```



Une pièce de monnaie

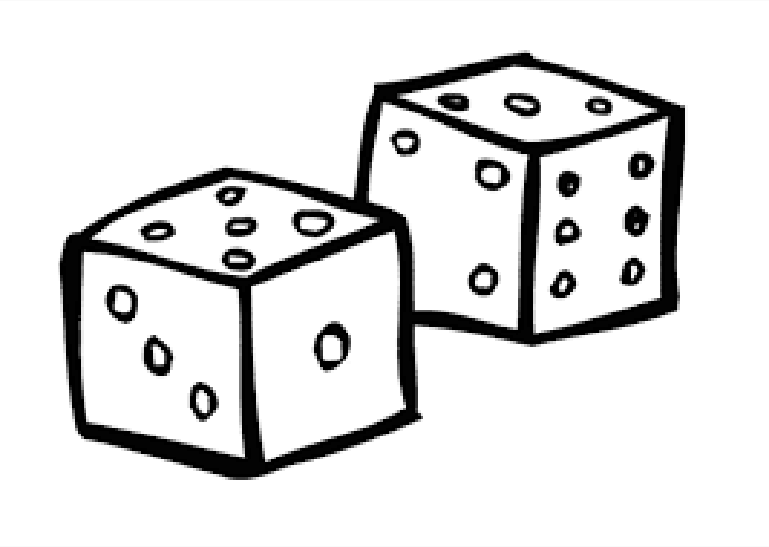
La loi binomiale


```
rbinom(n = 10, size = 1, prob = 0.5)
```

```
##  [1] 1 1 1 0 1 0 0 0 0 0
```

Les dés

La loi multinomiale



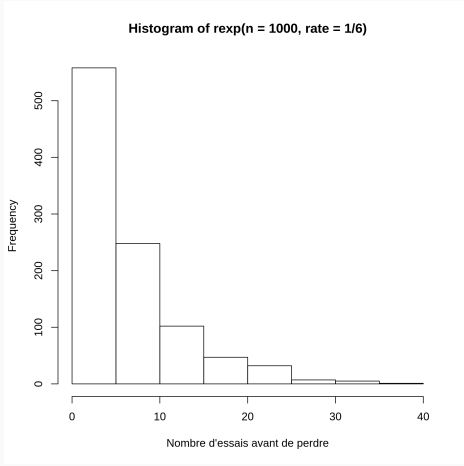
```
rmultinom(n = 3, size = 1,
  prob = rep(1/6, 6))
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    1    0
## [4,]    0    0    1
## [5,]    0    0    0
## [6,]    1    0    0
```

La roulette russe

La loi exponentielle

```
hist(rexp(n = 1000, rate = 1/6), xlab =
  "Nombre d'essais avant de perdre")
```



Exercice

Combien de fois obtient-on "pile" si on lance une pièce de monnaie 10 fois de suite ?

Solution

On peut utiliser la loi binomiale pour simuler le nombre de fois où l'on obtient pile. On obtient alors un vecteur. On peut ensuite compter le nombre de 1 dans ce vecteur (le nombre de réussites).

```
tirages ← rbinom(n = 10, size = 1, prob = 0.5)
tirages
```

```
##  [1] 1 0 1 0 0 1 1 0 1 0
```

```
sum(tirages)
```

```
## [1] 5
```

Exercice

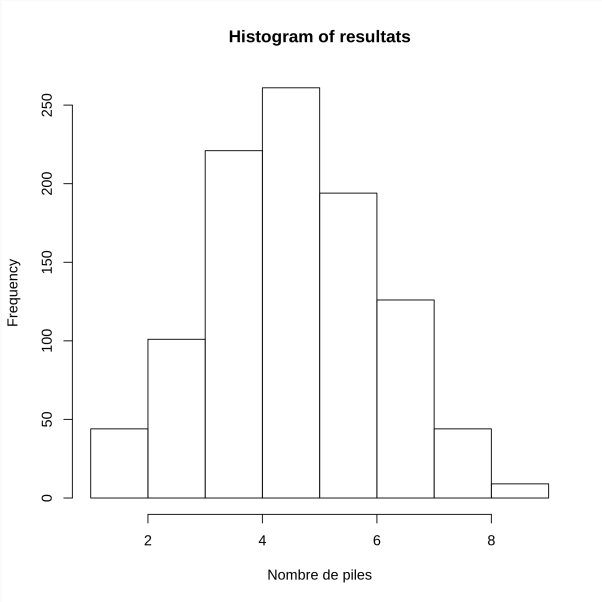
Quel est le résultat si on répète l'expérience 1000 fois ?

1. Insérez le code dans une boucle `for` pour répéter l'expérience 1000 fois.
2. Sauvez le résultat de chaque expérience dans un vecteur.
3. Utilisez la fonction `hist()` pour visualiser le résultat.

Solution

```r
resultats ← c()
for (i in 1:1000) {
  tirages ← rbinom(n = 10, size = 1, prob = 0.5)
  resultats ← c(resultats, sum(tirages))
}
```

```r
hist(resultats, xlab = "Nombre de piles")
```



Exercice

Qu'est-ce qui se passe si on ajoute `set.seed(123)` avant le tirage `rbinom` ?

Solution

```
resultats ← c()
for (i in 1:1000) {
  set.seed(123)
  tirages ← rbinom(n = 10, size = 1, prob = 0.5)
  resultats ← c(resultats, sum(tirages))
}
```

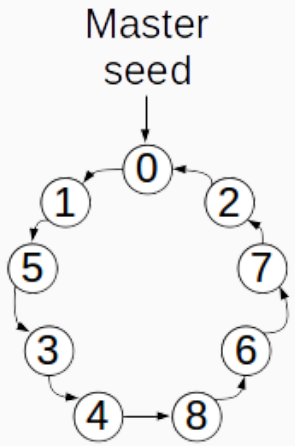
```
summary(resultats, xlab = "Nombre de piles")
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       6       6       6       6       6       6
```

Reproduire un tirage aléatoire

Les chiffres aléatoires sont générés aléatoirement... mais pas complètement !

La fonction `set.seed()` permet de reproduire des tirages aléatoires. Si on utilise la même graine, on obtient les mêmes tirages. Particulièrement utile si l'on veut reproduire les résultats d'une simulation ou d'une analyse statistique à tout coup.



Attention, il s'agit d'une simplification

Exercice

Une étude vous indique que la relation entre le nombre d'espèces de plantes et l'aire sur la placette de Sutton suit la relation suivante :

$$S \sim N(\mu, \sigma)$$

Où

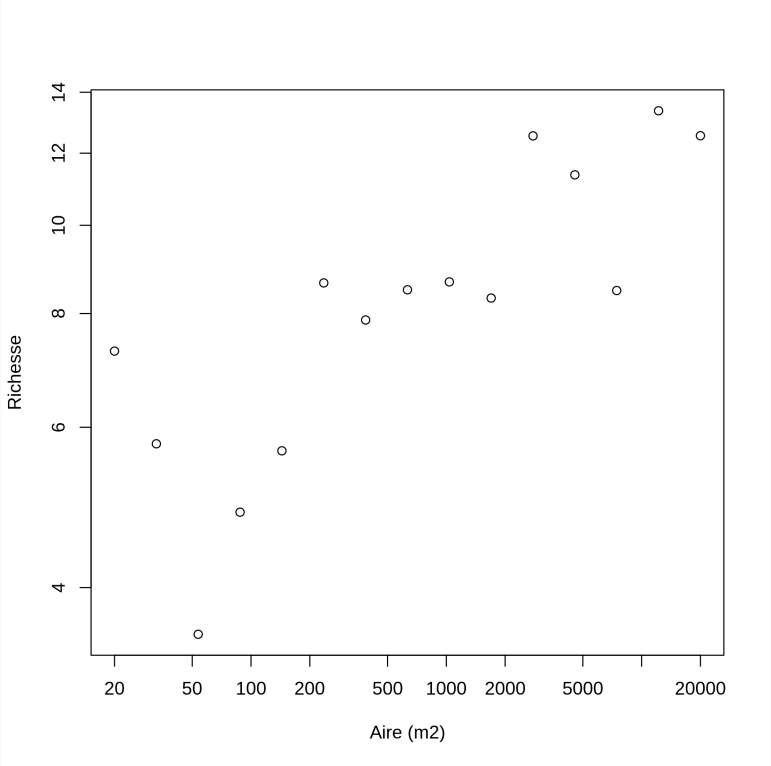
$$\mu = cA^z$$

Vous trouvez dans la littérature des valeurs de paramètres de $c = 3$, $z = 0.15$ et $\sigma = 1.5$. Illustrez des valeurs attendues sur ce modèle pour l'interval de $A = [20, 20000]$ correspondant à la dimension de la placette.

Solution

```
A = 2*10^seq(1, 4, length.out = 15)
c = 3
z = 0.15
S = rnorm(15, mean = c*A^z, sd = 1.5)

plot(A, S, xlab = "Aire (m2)", ylab = "
  log = "xy")
```



Application 3: prise de décision

Principe

L'épreuve de Bernoulli

- Une épreuve de Bernoulli est une expérience aléatoire (un tirage) avec deux issues : succès ou échec;
- Les épreuves sont indépendantes: l'issue d'une seconde épreuve ne dépend pas de la première;
- La probabilité de succès est représentée par le paramètre `p`, alors que la probabilité d'un échec est représentée par `1-p`;

Principe

L'épreuve de Bernoulli

Exemple des jeux de hasard: la pièce de monnaie

Exemples en écologie:

- la mortalité;
- la détermination du sexe;
- l'occurrence d'un incendie forestier;
- la contamination par un virus après le contact entre une personne infectée et une personne susceptible;

Simuler un tirage de Bernoulli

Lorsque $p = 0.5$, on tire une pièce de monnaie. Mais qu'en est-il si la pièce est biasée, par exemple si la face est plus lourde et donc la probabilité d'obtenir pile est de $p = 0.55$?

```
set.seed(1)
p = 0.55
alea = runif(n = 1, min = 0, max = 1)
if(alea < p) res = "pile" else res = "face"
res
```

```
## [1] "pile"
```

Un exemple vaut mille mots

- Tirez au hasard 1000 chiffres de la loi uniforme bornée entre 0 et 1
- Créez un vecteur 'res' rempli de 0 et de 1000 unités de longueur
- Pour chaque valeur de la séquence, évaluez si le chiffre est plus petit que la valeur seuil de $p = 0.2$. Si la valeur est plus petite, inscrivez 1 dans le vecteur 'res' à la position correspondante
- Combien de fois est-ce que la valeur est inférieure à p ?
- Essayez à nouveau et comparez.

Un exercice est encore plus efficace...

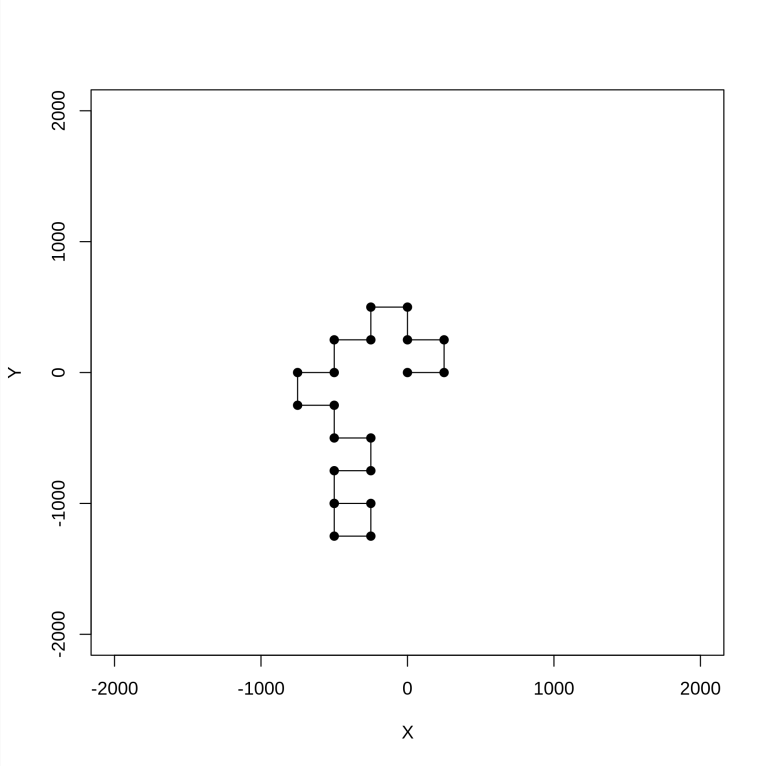
Vous observez une personne ivre à la sortie d'un bar prendre sa voiture. Vous appelez la police pour l'avertir. Estimez la distance parcourue par ce conducteur en 10 minutes, sachant que :

- Le quartier où vous vous trouvez est un plan cartésien parfait (une grille régulière) composé de blocs de 250 m;
- À chaque intersection, votre conducteur ivre tourne à gauche ou à droite avec une probabilité de 0.5;
- Il faut environ 30 secondes à votre conducteur pour parcourir la distance entre deux arrêts.

Calculez la distance parcourue en X et en Y, puis reprenez votre calcul une centaine de fois pour estimer la distance moyenne parcourue (à vol d'oiseau).

Solution

```
nsteps = 20
xy = matrix(0,nr = 21, nc = 2)
xy[1,] = c(0,0)
direction = 0
set.seed(2)
for(step in 2:(nsteps+1)) {
    # De quel côté tourner ?
    if(runif(1,0,1) < 0.5) {
      # Tourne à droite
      direction = direction + pi/2
    }
    else {
      # Tourne à gauche
      direction = direction - pi/2
    }
    # Calcul des nouvelles coordonnées
    xy[step,1] =
      xy[step-1,1] + sin(direction)*250
    xy[step,2] =
      xy[step-1,2] + cos(direction)*250
}
```



Optimisation des scripts

Optimisation

R est un langage de programmation et peut donc faire à peu près tout ce que l'on fera avec d'autres langages de programmation. Mais il a d'abord et surtout été développé pour l'analyse statistique de données. Par conséquent, il peut être assez lent pour réaliser certaines opérations.

L'optimisation de code peut être réalisée de différentes façons:

- Trouver les portions de code qui prennent le plus de temps ;
- Profiter de la structure de R (vectorielle) afin d'accéléer le calcul ;
- Écrire ses propres fonctions en C pour les portions les plus exigeantes ;

Référence

L'optimisation est un sujet avancé que nous ne couvrirons pas en détails dans le cours. Il est recommandé cependant d'aller consulter l'excellente référence sur le sujet :

> Visser, M.D., McMahon, S.M., Merow, C., Dixon, P.M., Record, S., Jongejans., E. 2015. Speeding up ecological and evolutionary computations in R; Essentials of high performance computing for biologists. PLoS Computational Biology 11: e1004140.

Calculer le temps écoulé

La fonction `system.time()` est un minimum pour tester la performance d'un code. Par exemple, on peut comparer la performance de notre fonction de tri à celle qui est native sur R.

Notre fonction de tri

```
tri ← function(x){
    # Calcul de la dimension du vecteur
    taille ← length(x)
    ordre ← "NON"
    # Boucle qui tourne jusqu'à ce que tout soit en ordre
    while(ordre == "NON") {
        ordre ← "OUI"
        # Boucle qui passe tous les éléments en paires
        for(i in 1:(taille-1)) {
            if(x[i+1] < x[i]) {
                # Inversion des deux lettres
                x[c(i,i+1)] ← x[c(i+1,i)]
                # Comme un changement a été fait, l'ordre
                # n'est pas encore garanti
                ordre ← "NON"
            }
        }
    }
    return(x)
}
```

Comparée à la fonction sort

```
x ← runif(1000)
identical(sort(x), tri(x))
```

```
## [1] TRUE
```

```
system.time(sort(x))
```

```
##    user  system elapsed
##       0       0       0
```

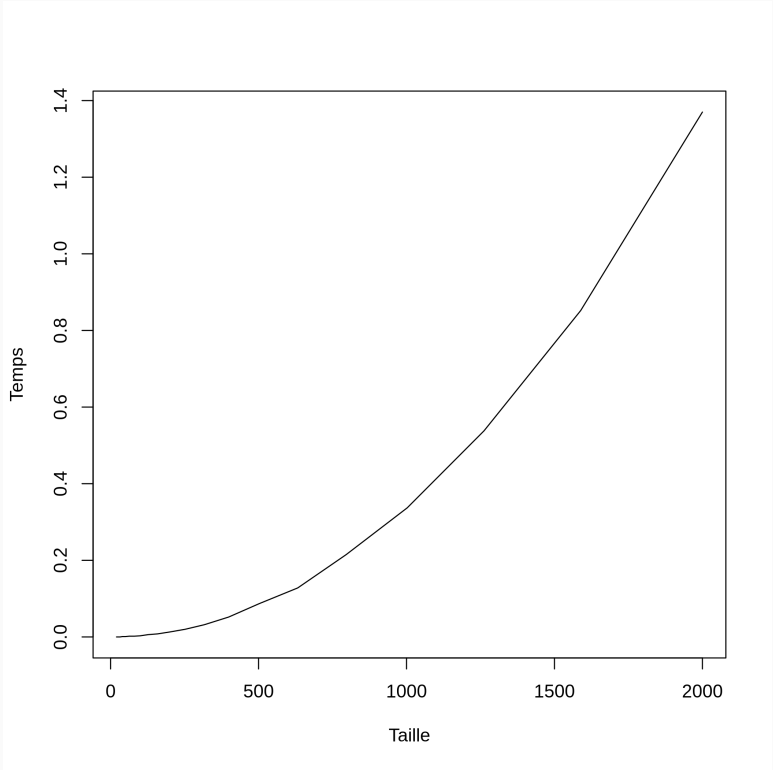
```
system.time(tri(x))
```

```
##    user  system elapsed
##   0.285   0.000   0.285
```

Calculer le temps écoulé

Certaines opérations peuvent dépendre de la taille de l'objet (ci-dessous, `res`)

```
n ← 2*10^seq(1, 3, 0.1)
res ← numeric(length(n))
for(i in 1:length(res)) {
    x ← runif(n[i])
    res[i] ← system.time(tri(x))[3]
}
plot(n, res, type = "l", xlab = "Taille
    ylab = "Temps")
```



Vectorisation

L'utilisation d'opérations vectorielles plutôt que les boucles augmente l'efficacité du code.

```
# Fonction avec boucle
f1 ← function(x) {
    taille ← length(x)
    x2 ← numeric(taille)
    for(i in 1:taille) {
    if(x[i] < 0.5) {
      x2[i] ← 1
    }
  }
}

# Fonction vectorisée
f2 ← function(x) {
    x2 ← x * 0
    x2[x < 0.5] ← 1
}
```

```
x ← runif(1000000)
system.time(f1(x))
```

```
##    user  system elapsed
##   0.066   0.000   0.067
```

```
system.time(f2(x))
```

```
##    user  system elapsed
##   0.007   0.000   0.007
```

Utilisation de fonctions natives

Certaines fonctions sur R (voir Visser et al. 2015) sont optimisées. L'exemple suivant montre la puissance de la fonction `rowSums()`.

```
f1 ← function(x) {
    res ← numeric(ncol(x))
    for(j in 1:ncol(x)) {
        for(i in 1:nrow(x)){
            res[x] ← res[j] + x[i,j]
        }
    }
}
X ← matrix(runif(100 * 100),
  nr=100, nc=100)
```

```
system.time(rowSums(X))
system.time(apply(X,2,sum))
system.time(f1(X))
```

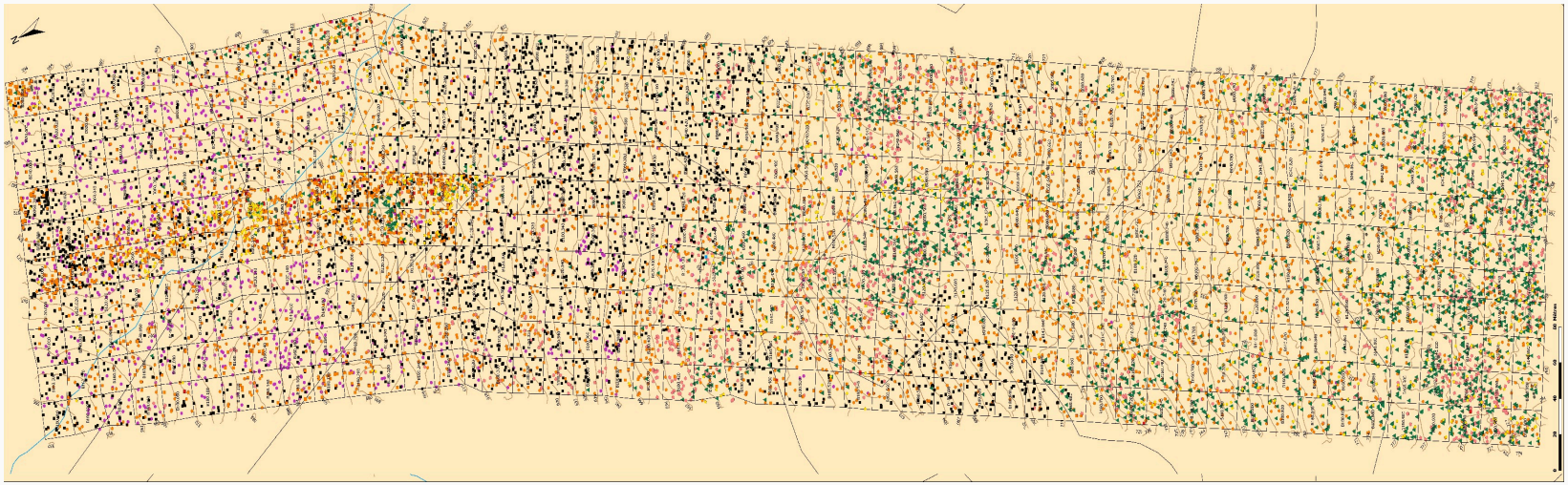
```
##    user  system elapsed
##   0.001   0.000   0.000
##    user  system elapsed
##   0.000   0.000   0.001
##    user  system elapsed
##   0.261   0.000   0.260
```

Travail final

Répartition des domaines bioclimatiques

Écotone tempéré-boréal

La distribution des arbres à Sutton



La Grive de Bicknell

Projet de session

Question de recherche

À quelle vitesse se réalisera la migration de l'érable à sucre au sein de la sapinière de montagne du massif des Montagnes vertes ?

Le type de données

```
arbres ← read.table(file = './donnees/arbres.txt', header = TRUE, sep=";")
head(arbres)
```

```
##   id_bor borx bory arbre  esp multi mort dhp
## 1    0-0    0    0 34501 acpe  FAUX FAUX  82
## 2    0-0    0    0 34502 acpe  VRAI FAUX  26
## 3    0-0    0    0 34502 acpe  VRAI FAUX  98
## 4    0-0    0    0 34503 acpe  FAUX FAUX  73
## 5    0-0    0    0 34504 acpe  FAUX VRAI  28
## 6    0-0    0    0 34506 fagr  FAUX FAUX  26
```

Objectif

Programmer un modèle de simulation stochastique qui permet de reproduire la dynamique de la végétation à la transition entre forêt tempérée et boréale

Le modèle de lotterie

En mots

- Pour chaque année de 2025 à 2100
- Pour chaque arbre dans le fichier `arbres.csv`
- On évalue si l'arbre survie (avec probabilité μ) ou meurt et est remplacé ($1 - \mu$)
- Si l'arbre meurt, on remplace son identité (l'espèce) en réalisant une lotterie parmi les semences qui se trouvent au pied de l'arbre
- On calcul l'abondance relative de chaque espèce par quadrat et sur l'ensemble de la parcelle à chaque pas de temps

Note : seules les colonnes `id_bor` et `esp` sont pertinentes pour le travail, les autres informations ne sont pas utilisées dans la simulation.

Le modèle de lotterie

La probabilité de remplacement

$$p_{i,x,t+1} = (1-m)\frac{\alpha_i N_{i,x,t}}{\sum \alpha_j N_{j,x,t}} + m\frac{\alpha_i R_{i,t}}{\sum \alpha_j R_{j,t}}$$

Où :

- i et j sont des espèces
- x est un quadrat
- t est une année
- α_i est un coefficient de recrutement
- m est le paramètre de dispersion
- $N_{i,x,t}$ est l'abondance dans le quadrat x au temps t
- $R_{i,t}$ est l'abondance relative sur la parcelle au temps t

Faire un tirage parmi plusieurs options

On pige l'espèce au hasard en utilisant `sample()` ou bien `rmultinom()` en utilisant un vecteur de probabilités p

```
esp ← c("A", "B", "C")
p ← c(0.2, 0.5, 0.3)
sample(x = esp, size = 1, prob = p)
```

```
## [1] "B"
```

Note : Le vecteur de probabilités doit toujours avoir une somme de 1

Étapes

1. Lecture du fichier `arbres.csv`
2. Fonction qui calcul l'abondance par quadrat au temps t
3. Fonction qui calcul l'abondance relative pour l'ensemble de la parcelle au temps t
4. Fonction qui réalise le tirage de l'identité de l'espèce à recruter
5. Fonction qui exécute la simulation sur 75 ans (mortalité, remplacement, calcul d'abondance)
6. Illustration de l'abondance relative des différentes espèces au fil du temps au moyen de la fonction plot

Paramètres

Vecteur des coefficients de recrutement

$\alpha = \{1, 1.01, 1.04, 1.03, 1, 1.02, 1)\}$

Probabilité de mortalité

$u = 0.01$

Probabilité de dispersion

$m = 0.1$

Rappel

On peut exécuter un script dans un autre script avec la commande `source()`

Exemple : on crée un script `main.R` avec les instructions suivantes :

```r
# Script main.R

# Lecture des fichiers
source("lecture_donnees.R")

# Charger les fonctions
source("fonctions.R")

# Exécuter la simulation
resultats ← simulation(debut = 2025, fin = 2100, u = 0.01, m = 0.1)

# Faire la figure
ma_figure(resultats)
```

Ensuite on exécute l'ensemble avec `source("main.R")`

Modalités

Remise

Le travail se réalisera en équipe de 2. Inscrire le nom des membres en commentaire sur les travaux

11 février : Séance de dépannage au D7-2007

18 février : Le travail final (pseudo code + programme) devra être remis sur Moodle avant le 18 février 23h59

Modalités

- Votre programme devra être remis dans un dossier `projet` qui contiendra vos scripts
- Le tout doit être reproductible sur un autre ordinateur sans avoir à intervenir. Assumez que le fichier de données `arbres.csv` se situe dans le même répertoire que vos scripts téléchargés à partir de Moodle

Critères d'évaluation : pseudo-code

Respect des bonnes pratiques enseignées

- Utilisation des éléments de base de la programmation (indexation, boucles, fonctions, énoncés conditionnels)
- Mise en forme des instructions et variables
- Les noms de variables sont explicites

La mise en forme adéquate d'un pseudo-code

- Une instruction par ligne
- Le code est indenté
- Le pseudo-code est divisé en blocs d'instructions cohérentes

Justesse des processus de programmation utilisés

- Les opérations répétées utilisent des boucles
- Les opérations logiques utilisent des déclarations conditionnelles

Critères d'évaluation : scripts

Respect des bonnes pratiques enseignées dans le cours (40%)

- Revoir les 10 instructions de la séance 1

Exécution des différentes étapes du code (40%)

- Lecture des données d'entrée
- Utilisation d'une fonction pour l'exécution de la simulation du modèle
- Réalisation de la simulation stochastique
- Exécution de la figure finale

Capacité d'exécuter le code de A à Z, soit de la lecture du fichier de données jusqu'à la figure, sans intervenir (20 %)

- Les données d'entrées seront dans le répertoire de travail d'où sera exécuté le code - nous utiliserons la fonction `source(main.R)` pour exécuter votre script d'analyse