TD de simulation d'équations différentielles stochastiques

Pierre Gloaguen

27/03/2020

1 Un modèle de croissance logistique

On s'intéresse à un modèle de dynamique de population plus réaliste que le modèle de croissance malthusienne vu en cours.

On note $X(t)$ la taille de la population au temps t. On suppose que le processus $\{X(t)\}_{t\geq 0}$ satisfait l'équation différentielle stochastique suivante:

$$\mathrm{d}X(t) = rX(t)\mathrm{d}t + \sigma X(t)\mathrm{d}B(t)$$

$$\mathrm{d}X(t) = rX(t)\left(1 - \frac{X(t)}{K}\right)\mathrm{d}t + \sigma X(t)\mathrm{d}B(t),$$

où r, K, σ sont trois paramètres **strictement positifs**.

1. Identifiez la fonction de dérive et la fonction de diffusion de cette EDS.

La dérive est donnée par

$$f(x) = rx(1 - \frac{x}{K})$$

La diffusion est donnée par

$$g(x) = \sigma x$$

2. Codez ces deux fonctions dans le logiciel R. Ces deux fonctions prendront comme argument une taille de population `x` et leurs paramètres (donc r et K pour la dérive, et σ pour la diffusion).

```
get_drift <- function(# Creation de la fonction get_drift
  x, r, K # Arguments de la fonction
  )
  {
  r * x * (1 - x / K)
}
get_diffusion <- function(x, sigma){ # Creation de la fonction get_diffusion
  sigma * x
}
```

3. En reprenant le squelette de fonction de simulation d'une EDS vu en cours, écrire une fonction de simulation de cette EDS, qui prendra en plus comme argument les paramètres r, K et σ.

```
simulate_log_growth <- function(x0, times, r, K, sigma){
  n_points <- length(times) # Nombre de points de simulations
  output <- rep(NA, n_points) # Initialisation du vecteur final
  output[1] <- x0 # Initialisation
  for(k in 2:n_points){ # Itération
    h <- times[k] - times[k - 1] # Pas de temps (doit être petit!)
    moyenne_euler <- output[k - 1] + # x
     get_drift(output[k - 1], r, K) * h # f(x, t) * h
```

```r
    variance_euler <- get_diffusion(output[k - 1], sigma)^2 * h
    output[k] <- rnorm(n = 1, # 1 simulation de loi normale
                       mean = moyenne_euler, # Moyenne
                       sd = sqrt(variance_euler) # Ecart-type
                       )
  }
  # Renvoi sous forme de tibble pour faciliter la visualisation
  resultat <- tibble(t = times, # Temps de simulation
                     x_t = output, # Simulation de l'EDS
                     x0 = x0, # Point initial
                     r = r, K = K, sigma = sigma) # Parametres utilises
  # ggplot dessine des tableaux
  return(resultat)
}
```

4. Pour le vecteur de temps suivant:

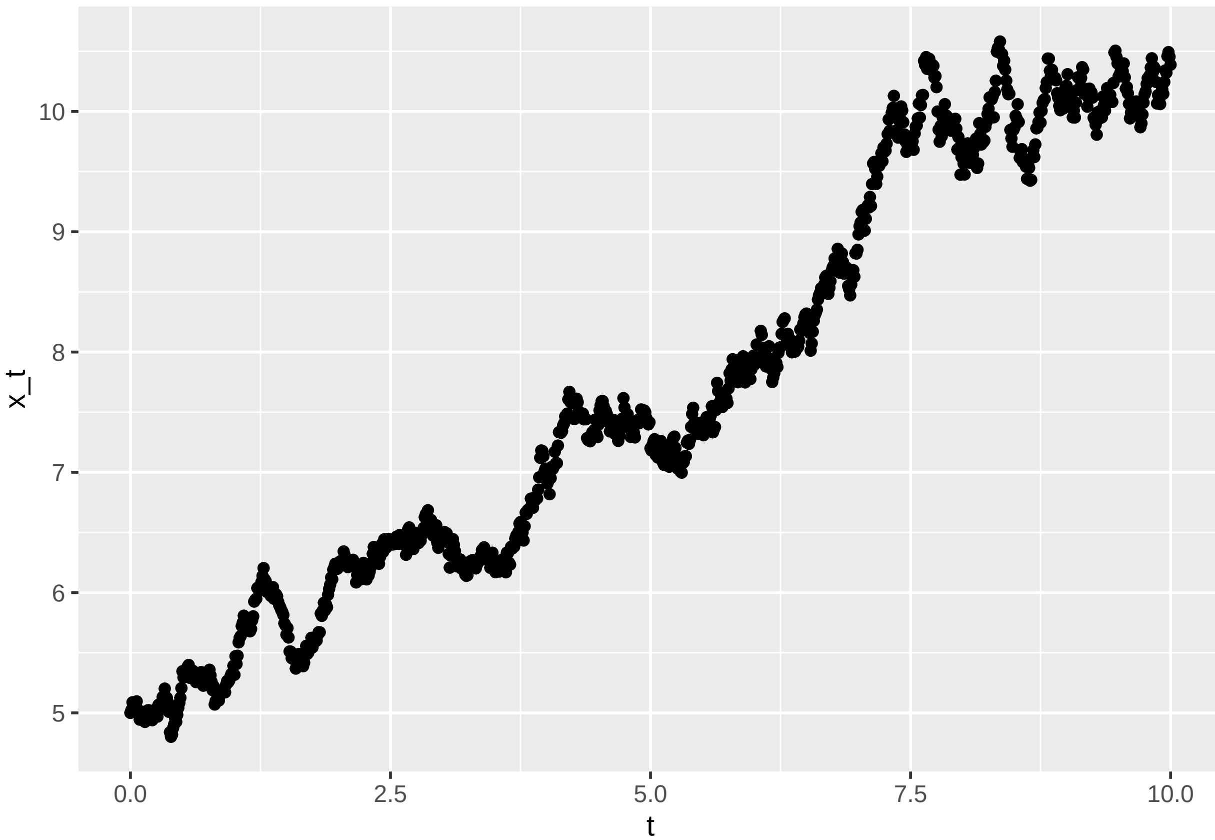
```r
# 0, 0.01, 0.02, ... 9.99, 10 (aille 1001)
my_times <- seq(from = 0, to = 10, by = 0.01) # "by" donne le pas de temps h
```

simulez le processus pour

```
- $r = 0.1, 0.5$, $K = 50, 500$, et $\sigma = 0.1, 3$. Dans chaque cas, vous ferez varier le point de d
Vous ferez plusieurs simulations afin d'évaluer la variabilité du processus.
```

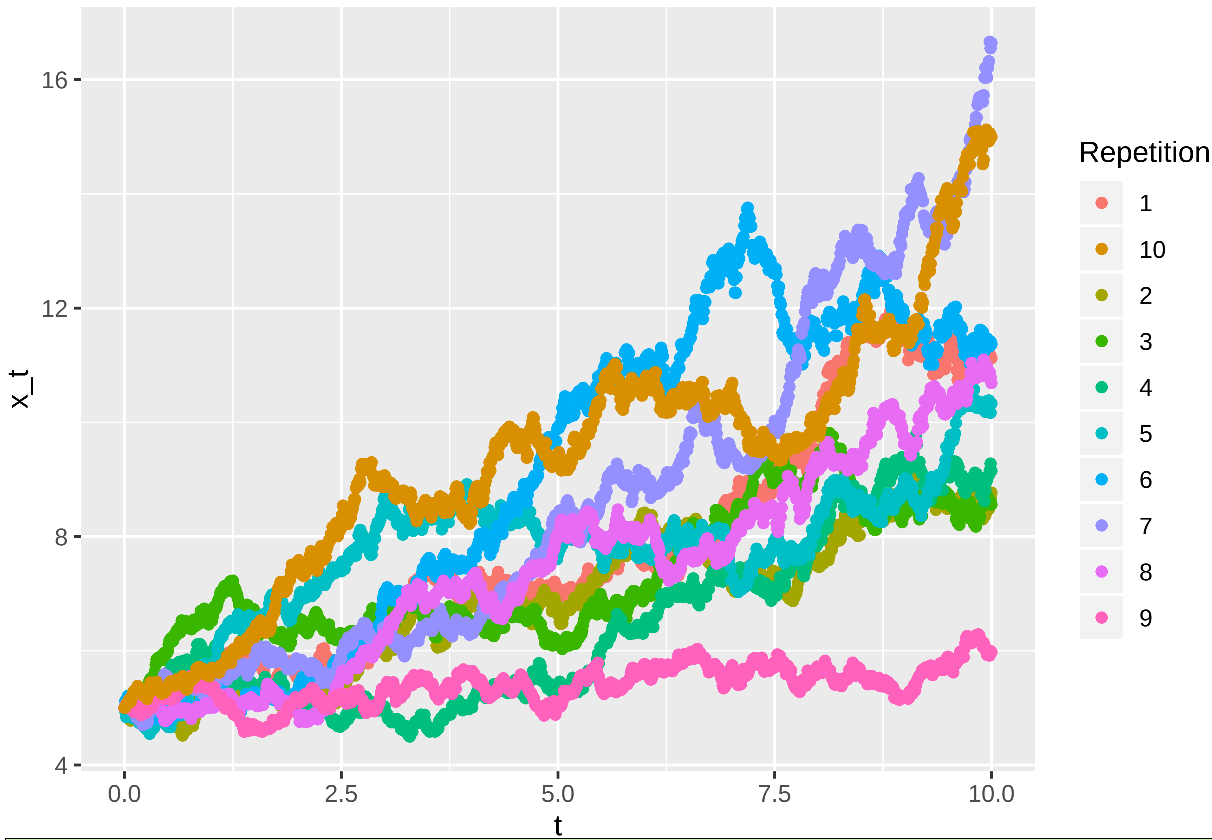
On commence par une première simulation

```r
premiere_simulation <- simulate_log_growth(x0 = 5, times = my_times,
                                           r = 0.1, K = 50, sigma = 0.1)
ggplot(premiere_simulation) + # Tableau à visualiser
  aes(x = t, y = x_t) +
  geom_point()
```



On peut ensuite en faire 10 grâce à la fonction `rerun`.

```
# rerun est une  Fonction du package purrr
dix_simulations <- rerun(10, # On fait tourner 10 fois
  simulate_log_growth(x0 = 5, times = my_times, # Le code précdent
                      r = 0.1, K = 50, sigma = 0.1)
  )  %>% # Le résultat est une liste ensuite, on aggrege tout dans un tableau
  bind_rows(.id = "Repetition") # On colle les tibble les uns sous les autres
# On garde en mémoire de quel tableau il s'agit dans la colonne repetition
ggplot(dix_simulations) + # Tableau à visualiser
  aes(x = t, y = x_t, color = Repetition) +
  geom_point()
```



Pour un jeu de paramètres, on peut ainsi voir la variabilité.

Maintenant, en étendant un peu le code, on peut regarder ce qui se passe pour différentes combinaisons de paramètres. On teste 16 configurations de paramètres différentes:

```
grille_parametres <- expand.grid(r = c(0.1, 0.5),
                                 K = c(50, 500),
                                 sigma = c(0.1, 3)) %>% # D'abord les paramètres
  as_tibble() %>%
  full_join(tibble(K = rep(c(50, 500), each = 2)) %>%
              mutate(x0 = c(0.1, 2) * K),
            by = "K") # puis x0
```

```
# A tibble: 16 x 4
       r     K sigma    x0
   <dbl> <dbl> <dbl> <dbl>
 1   0.1    50   0.1     5
 2   0.1    50   0.1   100
 3   0.5    50   0.1     5
 4   0.5    50   0.1   100
 5   0.1   500   0.1    50
 6   0.1   500   0.1  1000
 7   0.5   500   0.1    50
 8   0.5   500   0.1  1000
 9   0.1    50   3       5
10   0.1    50   3     100
11   0.5    50   3       5
12   0.5    50   3     100
13   0.1   500   3      50
```

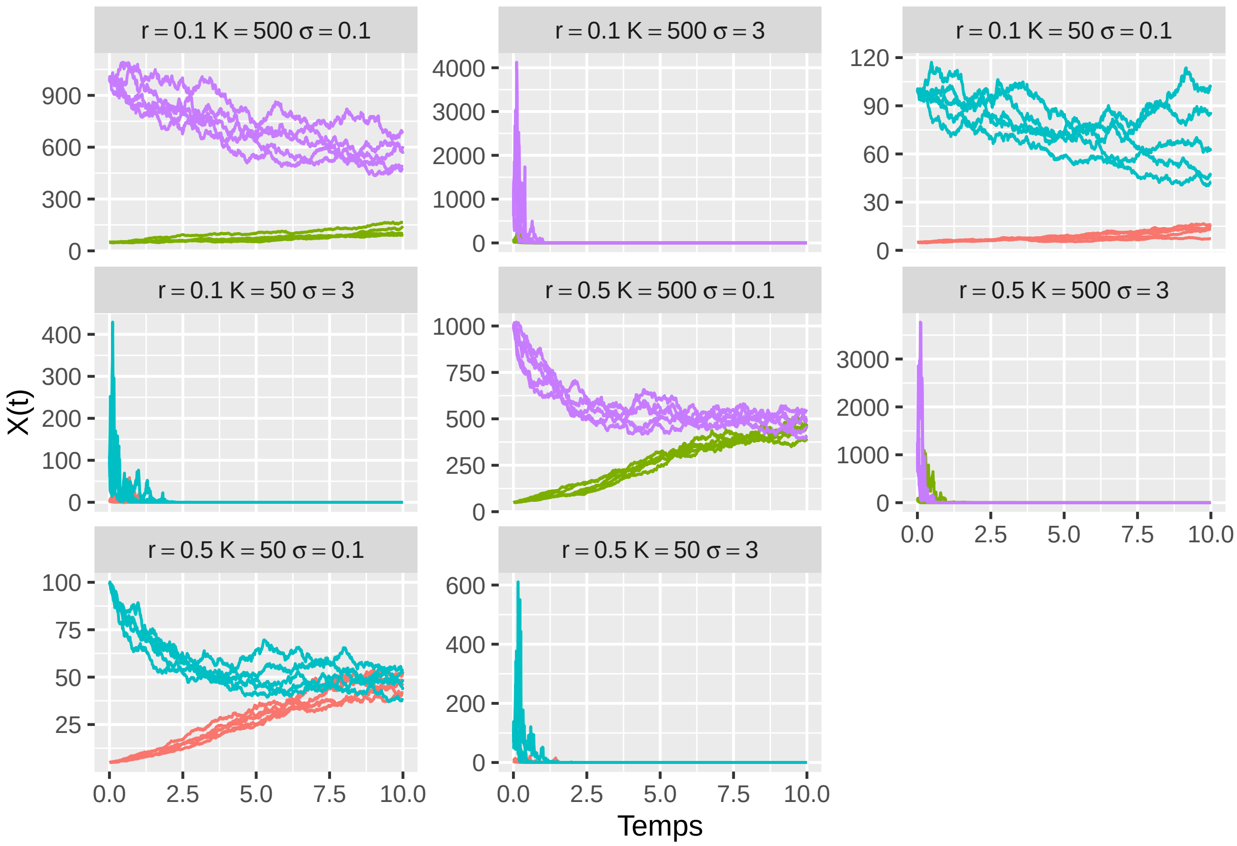
```
14   0.1    500    3     1000
15   0.5    500    3       50
16   0.5    500    3     1000
```

Pour chaque jeu de paramètres, on fait 5 simulations. Le jeu de donnees commence à être conséquent!

```r
ensemble_simus <- rerun(5, #  Pour chaque jeu de paramètres, 5 répétitions
                        pmap_dfr(.l = grille_parametres, # Liste des arguements
                                 .f = simulate_log_growth, # Fonction appliquée
                                 times = my_times))  %>% # Argument supplémentaire
  bind_rows(.id = "Replicat") # On concatène les 5 répétitions, on
# conserve l'info réplicat grâce à la colonne réplicat
```

```
# A tibble: 80,080 x 7
   Replicat     t   x_t    x0     r     K sigma
   <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 1         0     5        5   0.1    50   0.1
 2 1         0.01  4.99     5   0.1    50   0.1
 3 1         0.02  5.00     5   0.1    50   0.1
 4 1         0.03  5.03     5   0.1    50   0.1
 5 1         0.04  4.96     5   0.1    50   0.1
 6 1         0.05  5.02     5   0.1    50   0.1
 7 1         0.06  5.11     5   0.1    50   0.1
 8 1         0.07  5.05     5   0.1    50   0.1
 9 1         0.08  5.13     5   0.1    50   0.1
10 1         0.09  5.07     5   0.1    50   0.1
# ... with 80,070 more rows
```

```r
ensemble_simus %>%
  mutate(r = paste("r ==", r), # On renomme (simplement pour l'esthetisme)
         K = paste("K ==", K),
         sigma = paste("sigma ==", sigma)) %>%
  unite(col = "Parametre", r, K, sigma, sep = "~") %>% # Une seule colonne
  ggplot() +
  aes(x = t, y = x_t, color = factor(x0)) +
  labs(x = "Temps", y = "X(t)") +
  geom_line(aes(group = interaction(Replicat, x0))) +
  facet_wrap(. ~ Parametre, scales = "free_y", labelle = label_parsed) +
  theme(legend.position = "none")
```



5. Quelle interprétation faites vous des 3 paramètres?

Les simulations précédentes permettent de voir que:

- r est le taux de croissance intrinsèque (quand la population est faible). Il est lié à la vitesse d'atteinte de la population d'équilibre. - K est la capacité d'accueil, c'est la population "idéale" (on peut le voir comme une quantité d'energie disponible). - σ est un paramètre d'intensité de l'aléa. On voit que quand il est trop grand, la population va s'éteindre!

6. **Validité du schéma d'Euler**: Dans l'exercice précédent, pour $r = 0.5, K = 500$ et $\sigma = 0.1$, faites 500 simulations et conservez les points finaux (c'est à dire la valeur du processus au temps 10). Tracez la distribution de ces points à l'aide d'un histogramme (ou de la fonction `density`, ou `geom_density`). Refaites cet exercice pour un `by = 0.001`, `by = 1` et `by = 2` dans le vecteur `my_times`. Comparez les différents histogrammes. Que remarquez vous? D'après vous, quel est le bon pas de temps pour simuler ce processus?

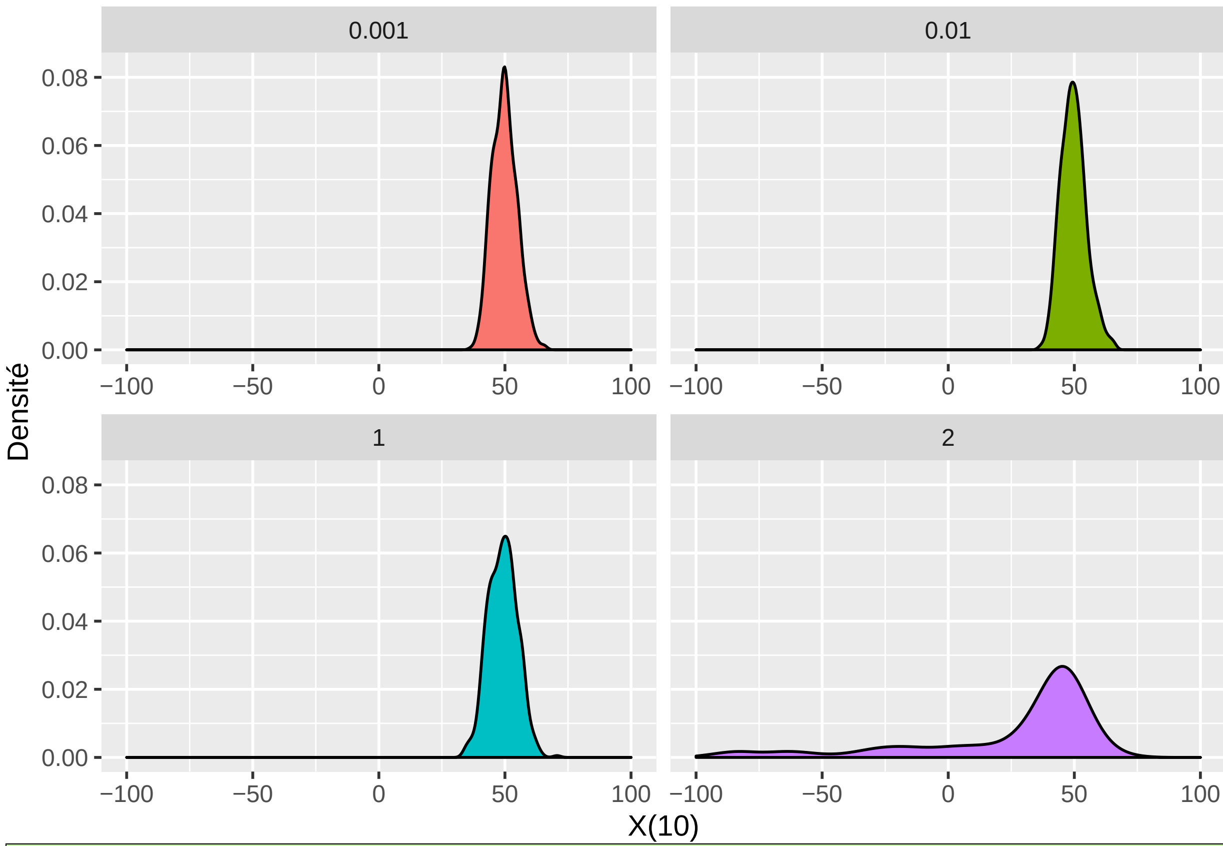
```r
pas_temps <- c(0.001, 0.01, 1, 2)
# Pour chaque pas de temps
final_points <- map_dfr(pas_temps,
        function(delta_t){
          sim_times <- seq(0, 10, delta_t)
          rerun(500,
                simulate_log_growth(x0  = 100, times = sim_times, # On simule
                                    r = 0.5, K = 50, sigma = 0.1) %>%
                  dplyr::filter(t == 10) %>% # On ne garde que le dernier point
                  dplyr::select(x_t) %>% # On ne garde que la colonne x_t
                  # (on se fiche du reste)
                  rename(x_10 = x_t) # On renomme en x_10 pour etre clair
                ) %>%
```

```r
            bind_rows() %>% # On aggrege tout dans un seul tableau
            mutate(time_step = paste(delta_t)) # On garde en memoire le pas
          # de temps dans la colonne time_step
        })
final_points # on visualise le resultat!
```

```
## # A tibble: 2,000 x 2
##     x_10 time_step
##    <dbl> <chr>
##  1  49.6 0.001
##  2  48.7 0.001
##  3  44.1 0.001
##  4  48.9 0.001
##  5  55.7 0.001
##  6  47.2 0.001
##  7  56.9 0.001
##  8  58.4 0.001
##  9  44.0 0.001
## 10  52.6 0.001
## # ... with 1,990 more rows
```

```r
ggplot(final_points) +
  aes(x = x_10) +
  geom_density(aes(fill = time_step)) +
  facet_wrap(.~time_step, scale = "free_x") +
  theme(legend.position = "none") +
  lims(x = c(-100, 100)) +
  labs(y = "Densité", x = "X(10)")
```

```
## Warning: Removed 171 rows containing non-finite values (stat_density).
```



Si on prend comme densité de référence celle obtenue avec le pas de temps 0.001, on voit que celle de 0.01 est similaire. De même, pour un pas de 1, on a une distribution assez proche (bien qu'un peu moins piquée). Donc ces deux pas de temps semblent adéquats pour bien simuler. Cependant, le pas de temps de 2 donne des résultats complètement aberrants.

2 EDS en deux dimensions, un modèle proie prédateur stochastique

On s'intéresse à la taille de deux populations en interaction (une population de proie et une population de prédateurs). Au temps t, la taille de la population est donnée par le vecteur $Z = \begin{pmatrix} X(t) \\ Y(t) \end{pmatrix}$. On suppose que le processus $\left\{ \begin{pmatrix} X(t) \\ Y(t) \end{pmatrix} \right\}_{t\geq 0}$ satisfait l'équation différentielle stochastique en deu dimensions

$$\begin{aligned} \mathrm{d}X(t) &= X(t)\left(a_{10} - a_{11}X(t) - a_{12}Y(t)\right)\mathrm{d}t + \sigma_1 X(t)\mathrm{d}B_1(t) \\ \mathrm{d}Y(t) &= Y(t)\left(-a_{20} + a_{21}X(t) - a_{22}Y(t)\right)\mathrm{d}t + \sigma_2 Y(t)\mathrm{d}B_2(t) \end{aligned}$$

où $\{a_{ij}, \sigma_i\}_{1\leq i\leq 2,\ 0\leq j\leq 2}$ sont paramètres **positifs**, et $\{B_1(t)\}_{t\geq 0}$ et $\{B_2(t)\}_{t\geq 0}$ sont deux mouvements Browniens **indépendants**.

1. Identifiez les différentes fonctions de dérive et de diffusion et codez les dans `R`. Elles dépendront d'un vecteur z de longueur 2 et des différents paramètres inconnus.

```
get_drift_2d <- function(z, a1_vector, a2_vector){
  x <- z[1];
```

```
  y <- z[2];
  drift_x <- x * sum(a1_vector * c(1, -x, -y))
  drift_y <- y * sum(a2_vector * c(-1, x, -y))
  return(c(drift_x, drift_y))
}
```

```
get_diffusion_2d <- function(z, sigma1, sigma2){
  c(sigma1, sigma2) * z
}
```

2. Interpréter les différents paramètres.

- a_{10} et a_{20} sont deux paramètres de taux de croissance intrinsèque. Comment évolue la population en absence de compétiteurs et de congénères? - a_{11} et a_{22} sont les impacts des congénères sur le taux de reproduction (si on a trop de congénère, la ressource est trop faible et la population diminue). - a_{12} et a_{21} sont les paramètres d'influence d'une espèce sur l'autre. Plus de proies entraine plus de prédateurs, plus de prédateurs entraine moins de proies.

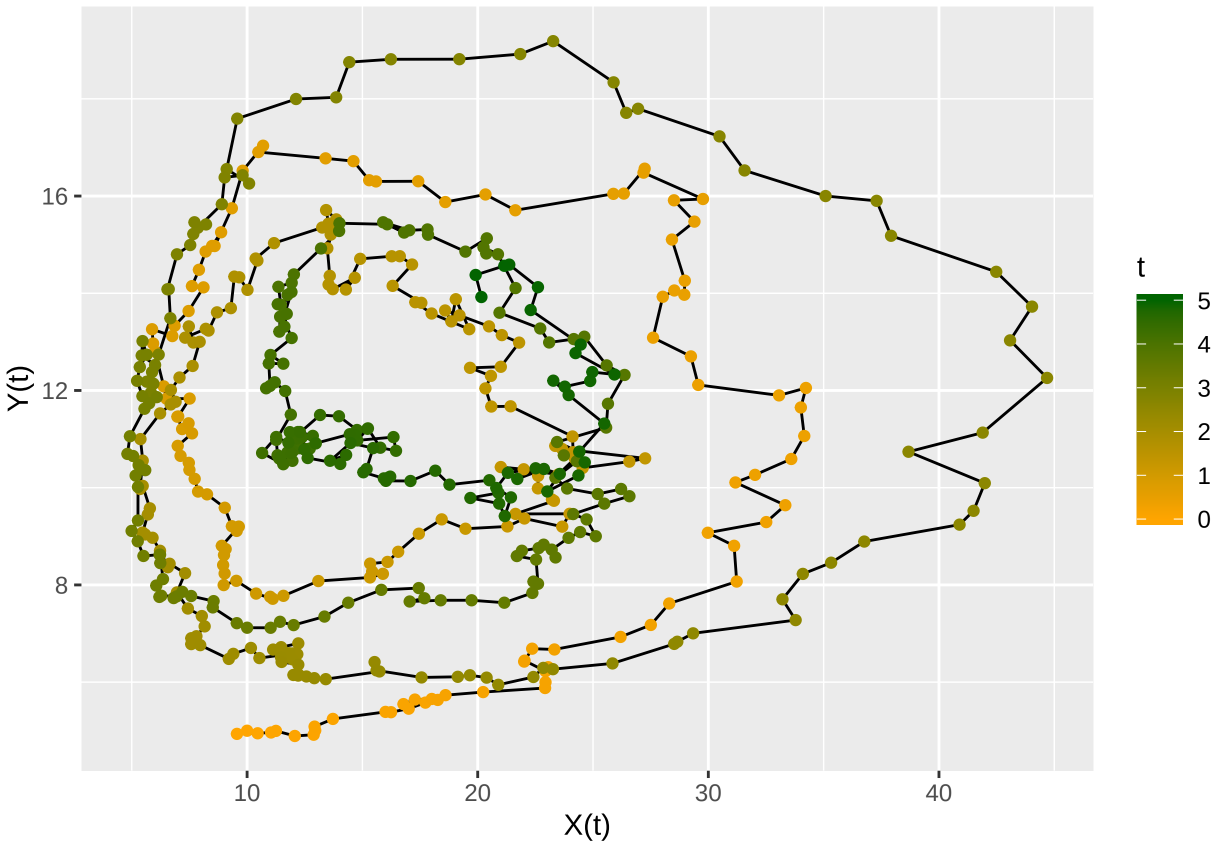
3. Ecrivez la fonction de simulation du modèle.

```
simulate_prey_predator_sde <- function(times, z0,
                                       a10, a11, a12,
                                       a20, a21, a22,
                                       sigma1, sigma2){
  n_points <- length(times) # Nombre de points de simulations
  # Cette fois, la sortie est une matrice, chaque ligne correspond
  # a une population
  output  <- matrix(NA, ncol = n_points, nrow = 2) # Initialisation de la matrice
  output[, 1] <- z0
  a1_vector <- c(a10, a11, a12) # Pour l'utiliser dans la fonction get_drift
  a2_vector <- c(a20, a21, a22)
  for(k in 2:n_points){ # Itération
    h <- times[k] - times[k - 1] # Pas de temps (doit être petit!)
    moyenne_euler <- output[, k - 1] + # c'est un vecteur
      get_drift_2d(output[, k - 1], a1_vector, a2_vector) * h # f(x, t) * h
    variance_euler <- get_diffusion_2d(output[, k - 1], sigma1, sigma2)^2 * h
    # Comme les deux browniens sont indépendants, on peut utiliser rnorm
    # Sinon, il faudrait mixtools::rmvnorm (normale multivariee)
    output[, k] <- rnorm(n = 2, # 2 simulations (c'est un vecteur)
                         mean = moyenne_euler, # Moyenne
                         sd = sqrt(variance_euler) # Ecart-type
    )
  }
  tibble(t = times,
         x_t = output[1, ],
         y_t = output[2, ])
}
```

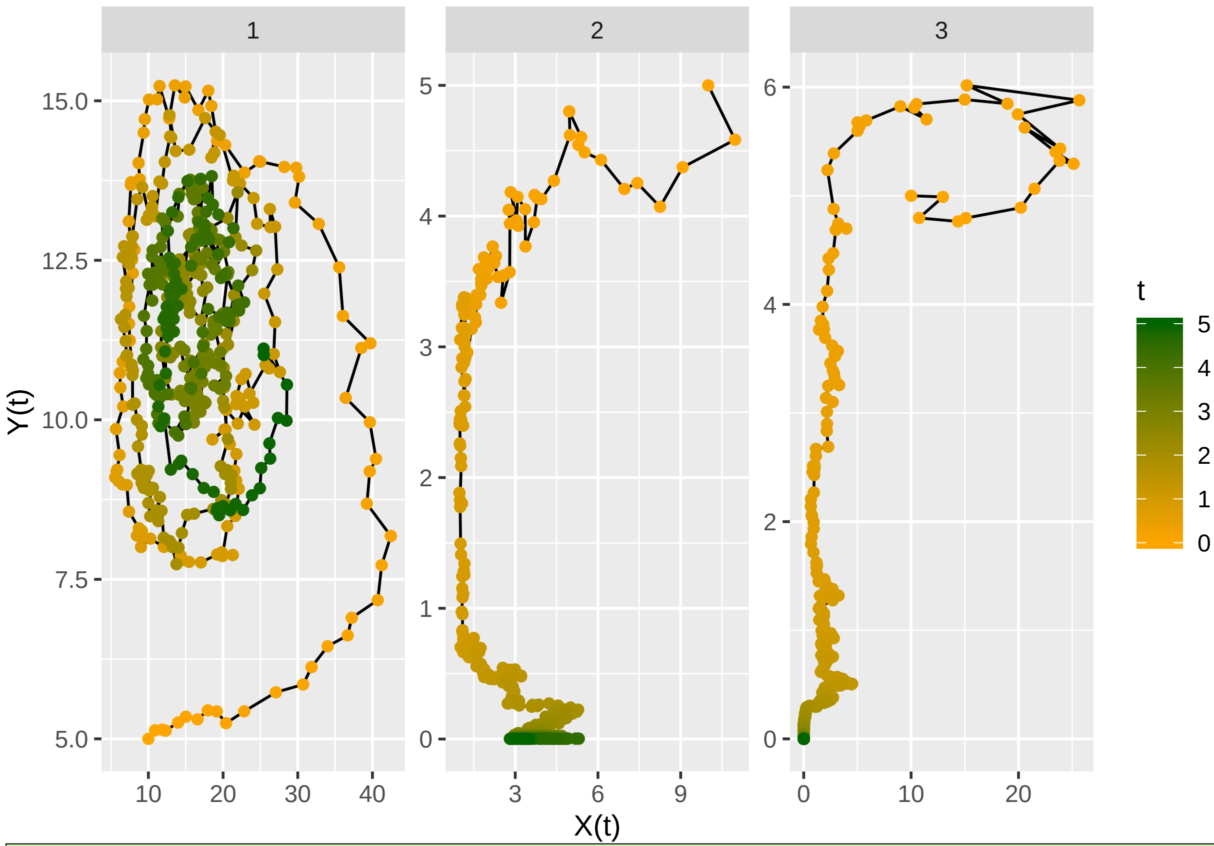
1. Pour les jeux de paramètres suivants (correspondant à 3 scenarios), simulez le processus pour $z(0) = (10, 5)$ et des temps allant de 0 à 5 (pas de temps de 0.01). Que constatez vous? Essayez d'intrerpréter les résultats

```
my_z0 <- c(10, 5)
my_times <- seq(0, 5, by = 0.01)
premiere_simu_2d <- simulate_prey_predator_sde(times = my_times,
                                               z0 = my_z0,
```

```r
                                                   a10 = 12, a11= 0.05, a12 = 1,
                                                   a20 = 2,  a21 = 0.2, a22 = 0.1,
                                                   sigma1 = 0.5, sigma2 = 0.2)
ggplot(premiere_simu_2d, aes(x = x_t, y = y_t)) +
  geom_path() +
  geom_point(aes(color = t)) +
  scale_color_gradient(low = "orange", high = "darkgreen") +
  labs(x = "X(t)", y = "Y(t)")
```




```r
grille_parametres_2d <- tibble(a10 = c(12, 4, 0.2),
                               a11 = c(0.05, 1, 0.05),
                               a12 = c(1, 1, 1),
                               a20 = c(2, 2, 2),
                               a21 = c(0.2, 0.2, 0.2),
                               a22 = c(0.1, 0.1, 0.1),
                               sigma1 = c(0.5, 0.5, 2),
                               sigma2 = c(0.2, 0.5, 0.2))
pmap_dfr(grille_parametres_2d,
         simulate_prey_predator_sde,
         times = seq(0, 5, 0.01), z0 = my_z0,
         .id = "Scenario") %>%
  ggplot(aes(x = x_t, y = y_t)) +
  geom_path() +
  geom_point(aes(color = t)) +
  scale_color_gradient(low = "orange", high = "darkgreen") +
  labs(x = "X(t)", y = "Y(t)") +
  facet_wrap(.~Scenario, scale = "free")
```



On constate qu'il y a 3 scenarios, un scenario où les deux populations persistent, un scenario où les prédateurs disparaissent mais les proies persistent, et un scenario où les deux populations disparaissent.