Rendu MDET

2023-10-16

```
main_theme = theme_bw()+
  theme(axis.line = element_line(colour = "black"),
        axis.text.x = element_text(colour = "black", size=13),
        axis.text.y = element_text(colour = "black", size=13),
        legend.title = element_text(colour = "black", size=13),
        legend.title.align=0.5,
        legend.text = element_text(colour = "black", size=12),
        axis.title=element_text(size=20))
```

Définition du modèles

Formalisme mathématique

Modèle de compétition Lotka-Volterra :

$$\frac{dn_i}{dt} = rn_i(1 - \frac{\sum_{j=1}^{N} a(x_i, x_j)n_j}{K(x_i)})$$

$$i = 1, 2, \ldots, N$$

Définition des paramètres

Avec :

- i le nombre de phénotype
- r le taux de développement de ce phénotype
- $a(x_i, x_j)$ la fonction de compétition interspécifique entre un phénotype x_i et un phénotype x_j. L'intensité de la compétition ne dépend que de la distance entre les deux phénotypes. Si elle augmente, la compétition augmente. Donc plus les traits sont proches plus la compétition est importante. $a(x_i, x_j)$ est défini selon la formule suivante :

$$a(x_i, x_j) = e^{-0.5\frac{(x_i - x_j)^2}{\sigma_a^2}}$$

- $K(x_i)$ la fitness d'un individu qui porte le phénotype x_i. $K(x_i)$ est défini de la façon suivante :

$$K(x_i) = (K_0 - \lambda(x_i - x_0)^2)$$

Implémentation du modèle sous R

Comme cela a été fait en TD, nous n'avons pas rappelé la démarche réflexive menant au pseudo code nécessaire à l'implémentation du modèle en R.

Sous R, pour des raisons d'optimisation, la proportion de chaque phénotypes sera calculé avec la formule suivente :

$$\frac{dN}{dt} = r * N_0 * (1 - N_0 * M)$$

Avec :

- N_0 : les valeurs initiales de densité des phénotypes
- M : la matrice qui regroupe les valeurs des fonctions de compétitions a entre chaque phénotypes et de fitness K égale à :

$$M_{ij} = \begin{pmatrix} \frac{a_{11}}{K_1} & \cdots & \frac{a_{N1}}{K_1} \\ \vdots & & \vdots \\ \frac{a_{1N}}{K_N} & \cdots & \frac{a_{NN}}{K_N} \end{pmatrix}$$

Création des fonctions nécessaires à la simulation

```
fonc.K = function(x, K0 = K0, lambda = lambda, x0 = x0){return(
  max(0, 1/(K0-lambda*(x - x0)^2)))}  #Afin de ne pas obtenir de valeur négative, le maximum
entre 0 et la valeur de K est retourné.

fonc.a = function(x, y, sigma = sigma){
  return(exp(-0.5*(x-y)^2/sigma^2))
}

EDO_LK <- function(t, y, paraM){
  with(as.list(paraM), { # permet d'éviter de lister les paramètres

  Mat = A * K

  dndt <- r*y*(1-(y%*%Mat)) # Equation différentielle

  return(list(dndt))
  })
}
```

Définition des valeurs des parmètres

Voici les paramètres dont nous aurons besoin pour définir notre modèle.

```
# les valeurs des paramètres sont renseignées, elles pourront être réadaptées au besoin

N <- 8 #nombre de traits

# Pour le calcul de la fonction de compétition (a)
sigma <- 1

# Pour le calcul de la fonction de fitness (K)
K0 <- 1
lambda <- 1
x0 <- 1/2

# Pour le calcul de la valeur des trais phénotypique
xmin <- 0
xmax <- 1

#Pour le calcul de la proportion d'un phénotype x donné (= résolution de l'EDO)
r <- 0.5
```

Calcul de K et a pour chaque valeur de traits phénotypiques :

```
# création des N valeurs de trait phénotypiques
X = seq(xmin, xmax, length.out = N)

#Création des matrice K et A vides
K = matrix(NA, nrow = length(X), ncol = length(X))
A = matrix(NA, nrow = length(X), ncol = length(X))

for(i in 1:length(X)){
  for(j in 1:length(X)){
    K[i,j] =  fonc.K(x = X[j],K0 = K0, lambda = lambda, x0 = x0)
    A[i,j] = fonc.a(x = X[i], y = X[j], sigma = sigma)
  }
}
```

Simulations

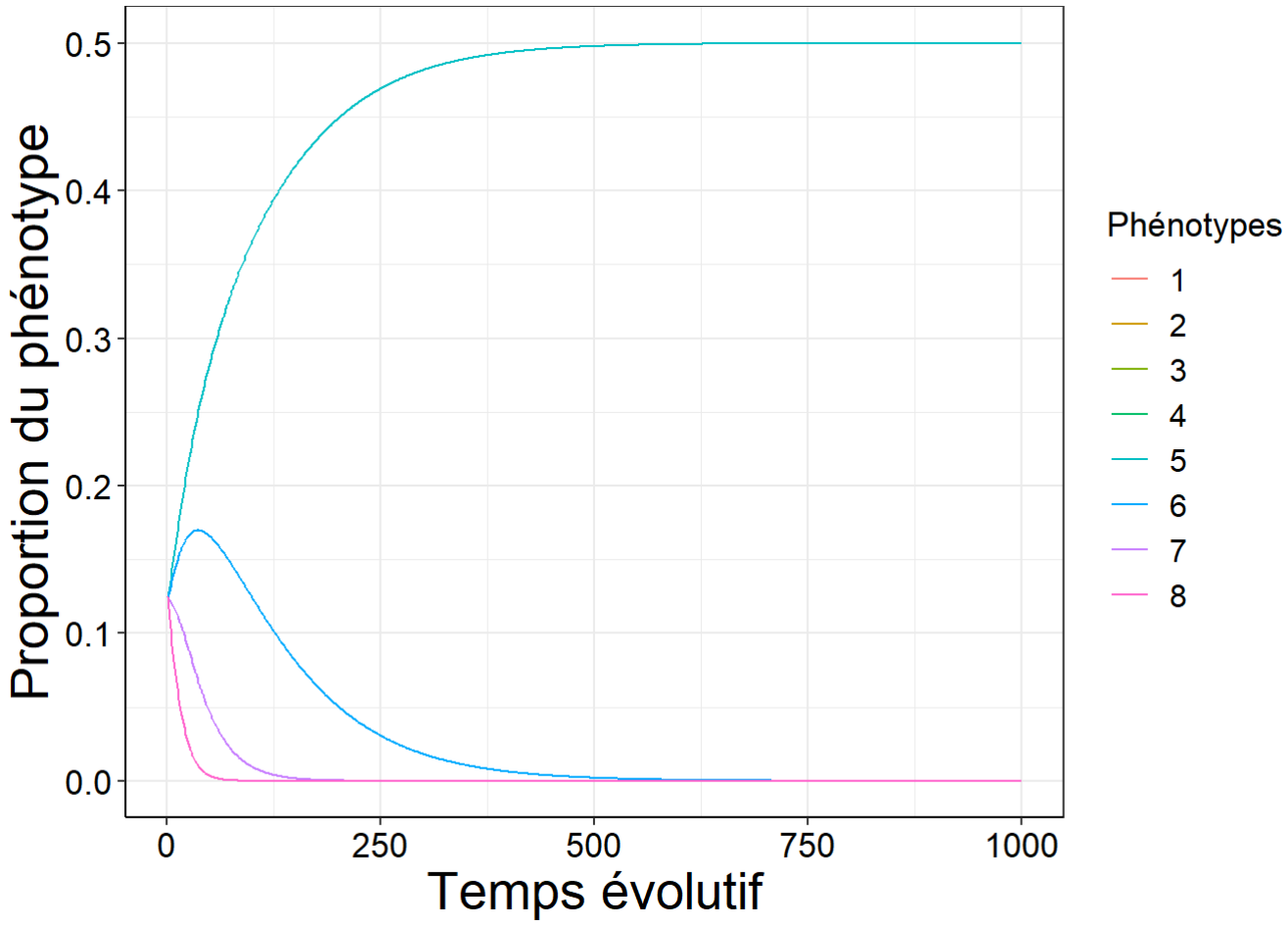
Pour simuler le comportement du modèle au cours du temps, nous avons besoin de résoudre cette équation différentielle.

Résolution de l’équation différentielle

L’équation différentielle ainsi définie est résolue par la fonction ode, le résultat, pour chaque pas de temps, est stocké dans la matrice solution.

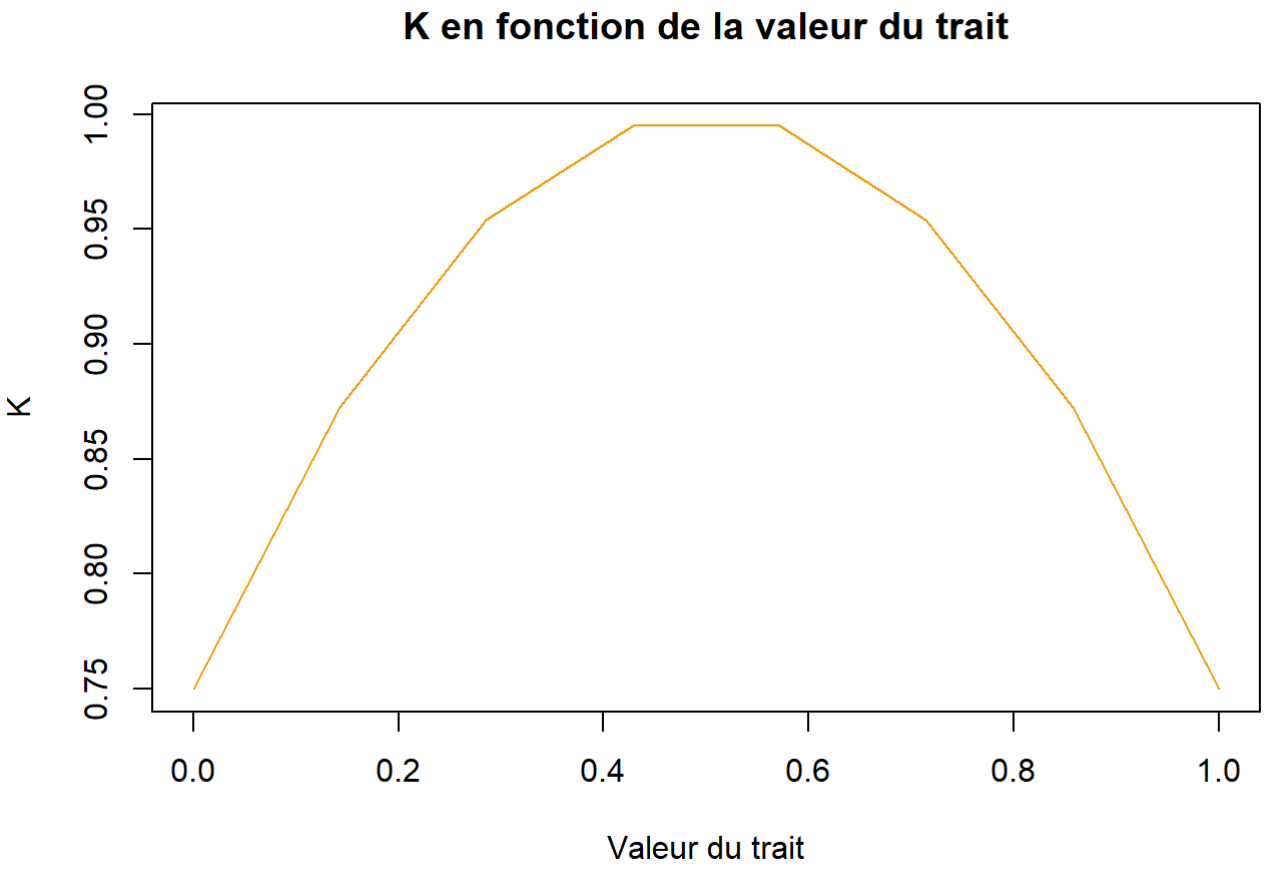
Evolution de la proportion du trait en fonction du temps

```
solution %>%
  as_tibble() %>%
  mutate_all(as.numeric) %>%
  pivot_longer(-time, names_to = "Phénotypes", values_to = "value") %>%
  ggplot() +
  geom_line(aes(x = time, y = value, color = Phénotypes))+
  xlab ("Temps évolutif")+
  ylab ("Proportion du phénotype")+
  main_theme
```



La simulation est réalisée pour 8 traits phénotypiques initiaux. Nous constatons une symétrie entre les traits qui sont distribués en mirroir selon la cloche de K (représentée ci-dessous). Cette simulation permet la fixation majoritaire de deux traits (courbe bleu claire), quatre autres disparaissent.

```
plot(x=X, y=K[1,]^(-1), type='l', col = "orange2", xlab= "Valeur du trait", ylab = "K", main
= "K en fonction de la valeur du trait")
```

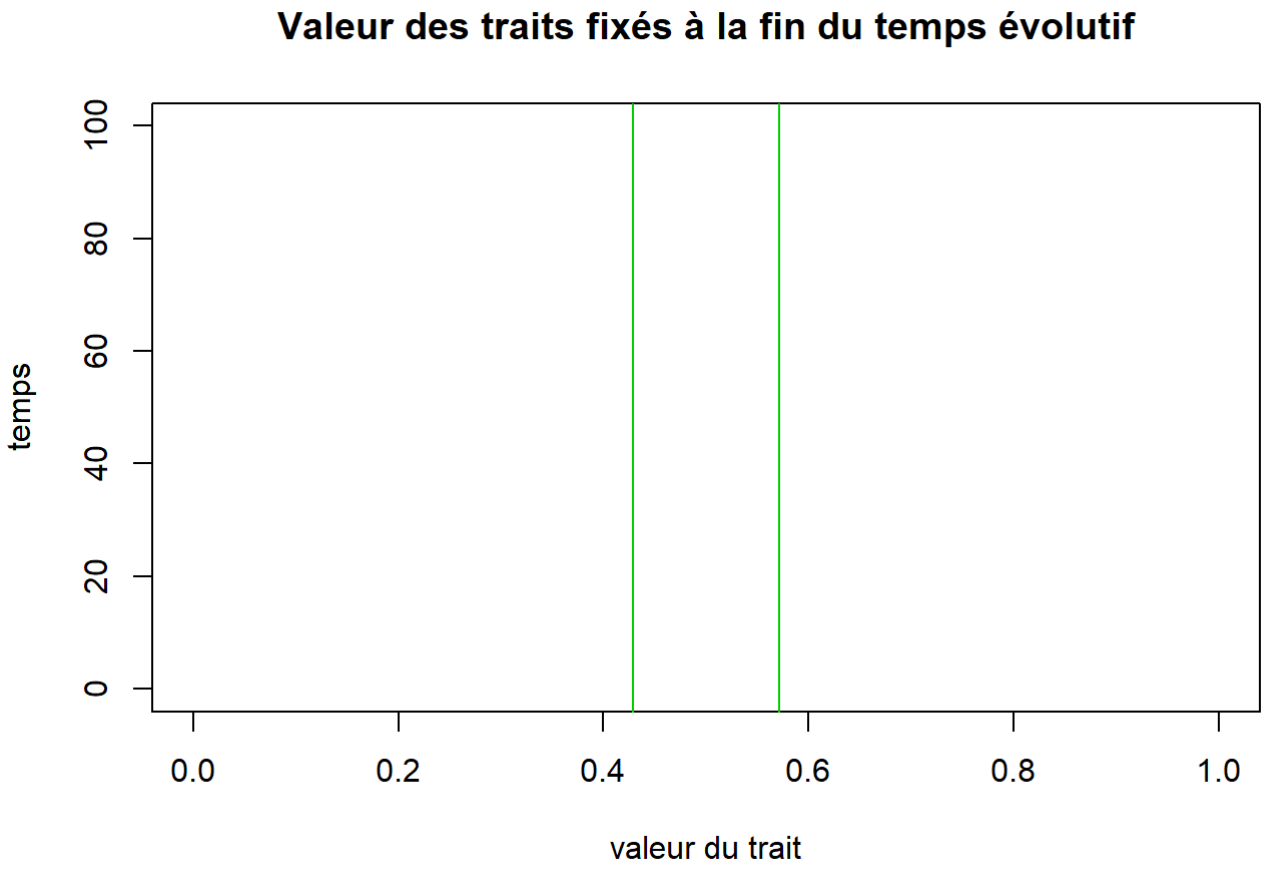


Ainsi, les valeurs de trait à 0.43 et 0.57 connaissent une même valeur de K et ont le même écart avec l'optimum à 0.5. Ayant la même différence, ils réagissent de la même façon vis-à-vis de la capacité d'accueil maximale. C'est pourquoi leur comportement est identique.

Valeur des traits fixés à la fin du temps évolutif

A la fin de la simulation, on cherche à savoir quels sont les traits qui ont été fixés et ceux qui ont disparus. Pour cela, on utilise le résultat au temps final de l'équation différentielle. Il s'agit de seuiller sur la densité des traits au sein de la population puis de représenter les seuls phénotypes ayant été conservés.

```
etatfinal = solution[nrow(solution),2:ncol(solution)] #solution de l'équation différentielle
au temps final

plot(NULL, ylim = c(0, 100), xlim= c(xmin, xmax), xlab="valeur du trait", ylab="temps", type
='l', main ="Valeur des traits fixés à la fin du temps évolutif")
for (rgtrait in 1:length(etatfinal)){
  if (etatfinal[rgtrait] > 0.1){ #seuil
    abline(v = X[rgtrait], col='green3')
  }
}
```



On retrouve bien les deux traits à 0.43 et 0.57 qui ont été fixés.

Aprofondissement de l'étude

Analyse de sensibilité du paramètre σ

On cherche à étudier l'influence de l'incertitude de σ sur le nombre de traits phénotypiques sélectionnés. σ intervient dans le calcul de compétition interspécifique entre deux traits tel que :

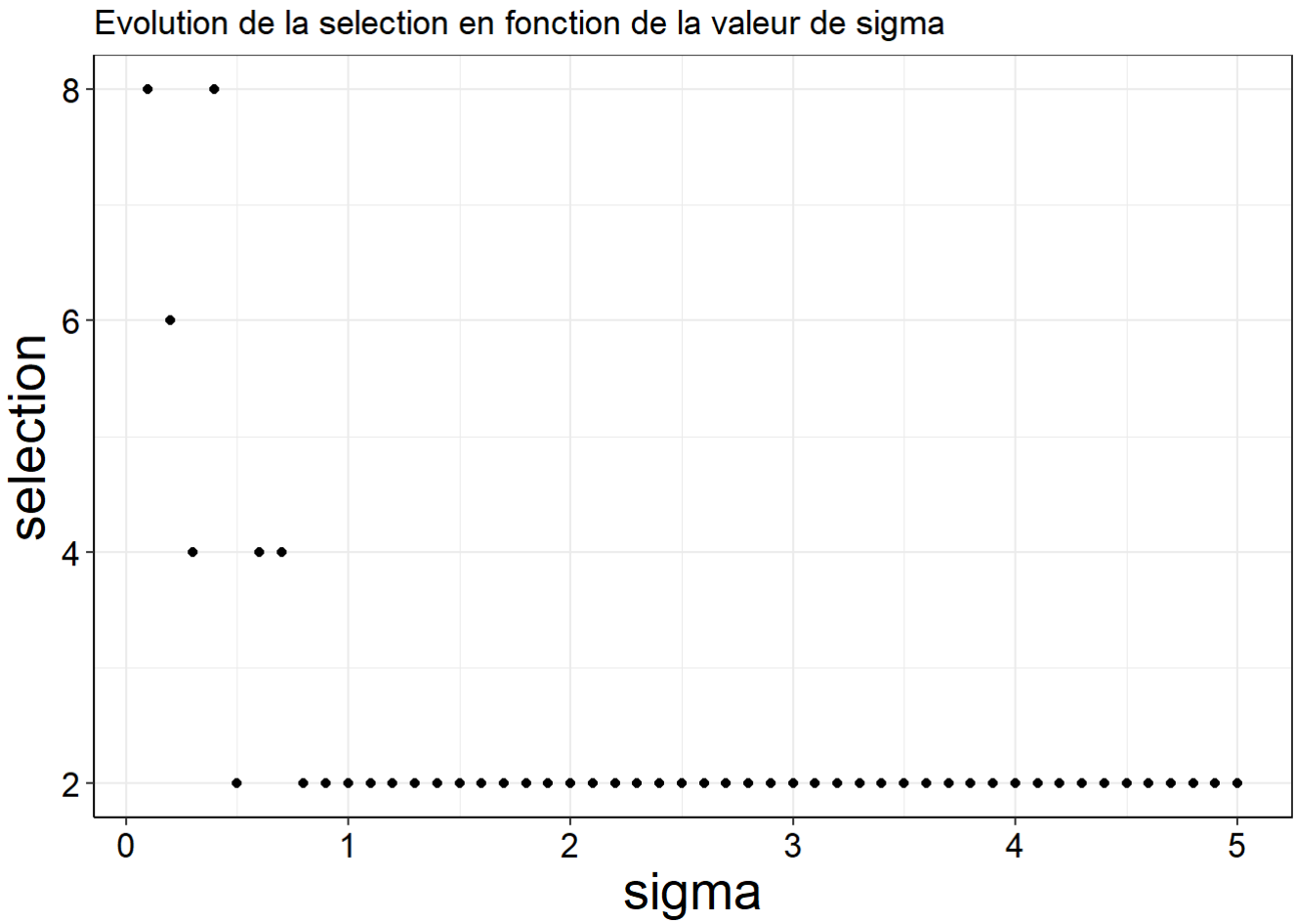
$$a(x_i, x_j) = e^{-0.5\frac{(x_i - x_j)^2}{\sigma_a^2}}$$

On sait déjà que l'intensité de la compétition va être d'autant plus grande que la distance entre les traits $(x_i - x_j)$ est petite. On cherche maintenant à déterminer, pour 8 traits donnés comment la valeur de σ influence l'intensité de la compétion interspécifique.

On commence par résoudre l'EDO pour différentes valeurs de σ :

Représentation graphique

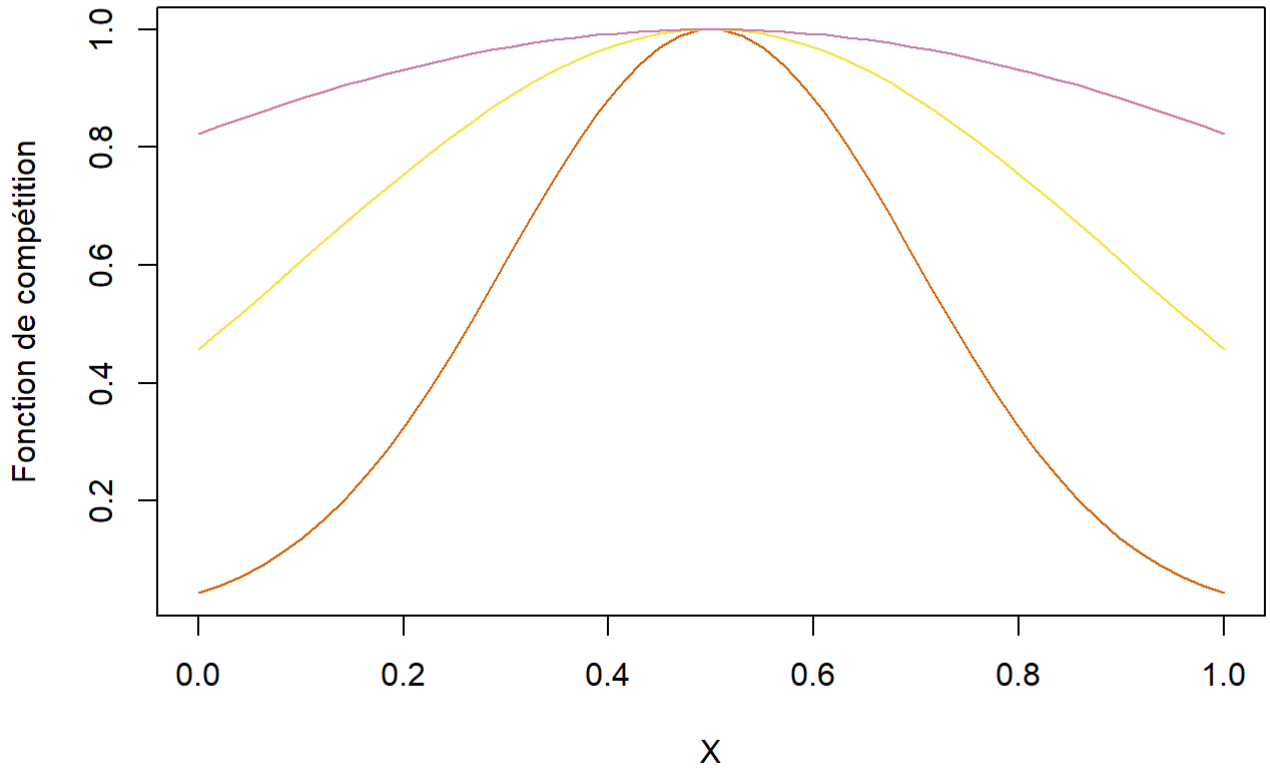
```
sel_sig = data.frame(sigma = val_sig, selection = selection)

ggplot(data = sel_sig)+
  geom_point(aes(x = sigma, y = selection))+
  ggtitle("Evolution de la selection en fonction de la valeur de sigma")+
  main_theme
```



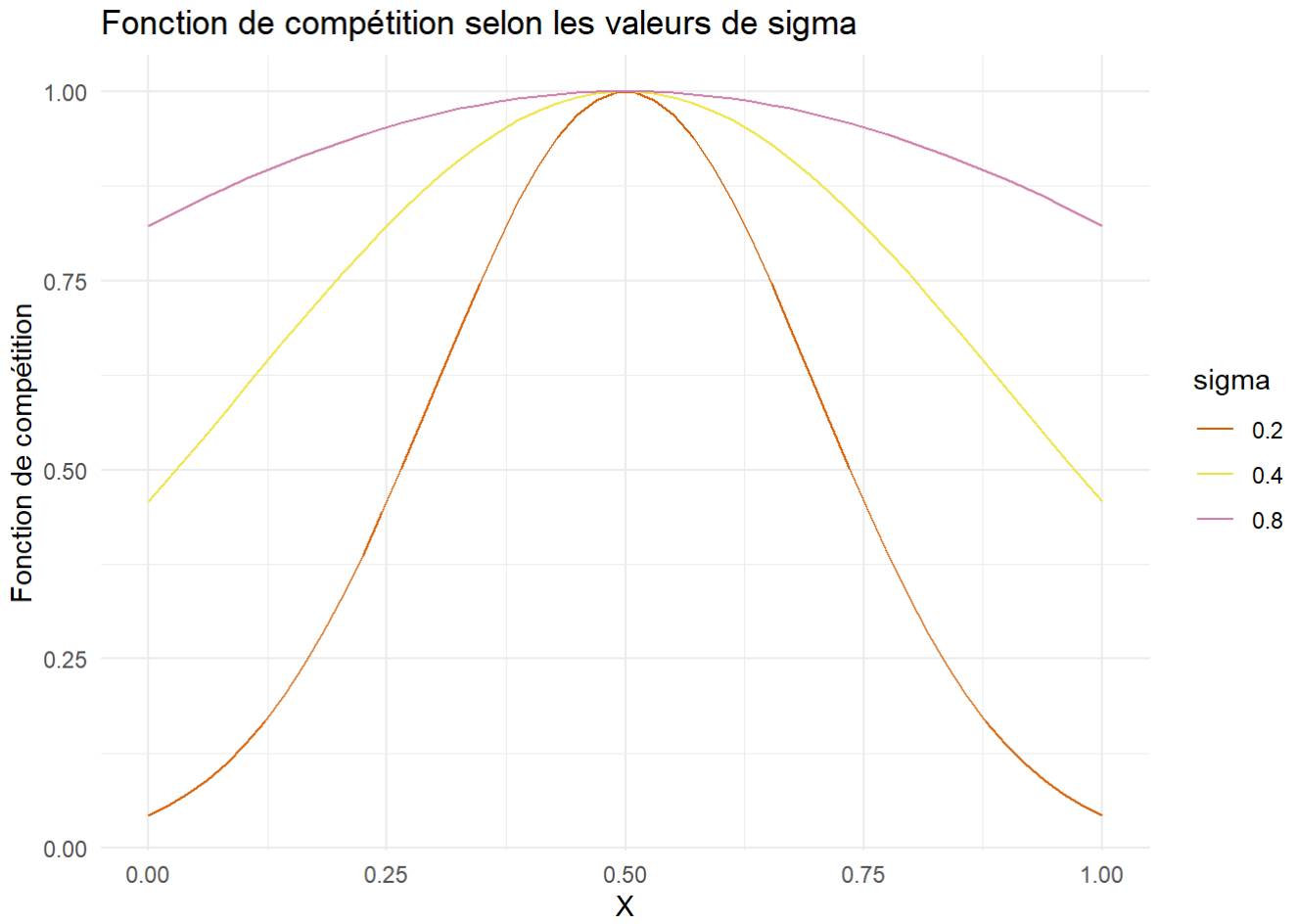
Pour 8 traits donnés il semble que plus σ augmente plus la compétition s'intensifie donc le nombre de traits sélectionnés diminue.

On peut faire la même observation en repésentant graphiquement la fonction de compétition $a(x_i, x_j)$:

```
X = seq(from = 0, to = 1, length.out = 50 )

plot(X,fonc.a(x = X, y = 0.5, sigma = 0.2), type = "l", col =  "#D55E00", ylab = ("Fonction d
e compétition"))
lines(X,fonc.a(x = X, y = 0.5, sigma = 0.4), type = "l", col = "#F0E442")
lines(X,fonc.a(x = X, y = 0.5, sigma = 0.8), type = "l", col = "#CC79A7")
```




```
X <- seq(from = 0, to = 1, length.out = 50)

data <- data.frame(
  X = rep(X, times = 3),
  Y = c(fonc.a(x = X, y = 0.5, sigma = 0.2),
        fonc.a(x = X, y = 0.5, sigma = 0.4),
        fonc.a(x = X, y = 0.5, sigma = 0.8)),
  Sigma = rep(c(0.2, 0.4, 0.8), each = length(X))
)

ggplot(data, aes(x = X, y = Y, col = factor(Sigma))) +
  geom_line() +
  scale_color_manual(values = c("#D55E00", "#F0E442", "#CC79A7")) +
  labs(y = "Fonction de compétition") +
  labs(color = "sigma")+
  ggtitle("Fonction de compétition selon les valeurs de sigma")+
  theme_minimal()
```



On peut voir que plus σ est grand plus la fonction de compétition est étendue, donc plus la compétition est importante pour un interval de traits élevé.

Introduction de Mutations et fixation des traits

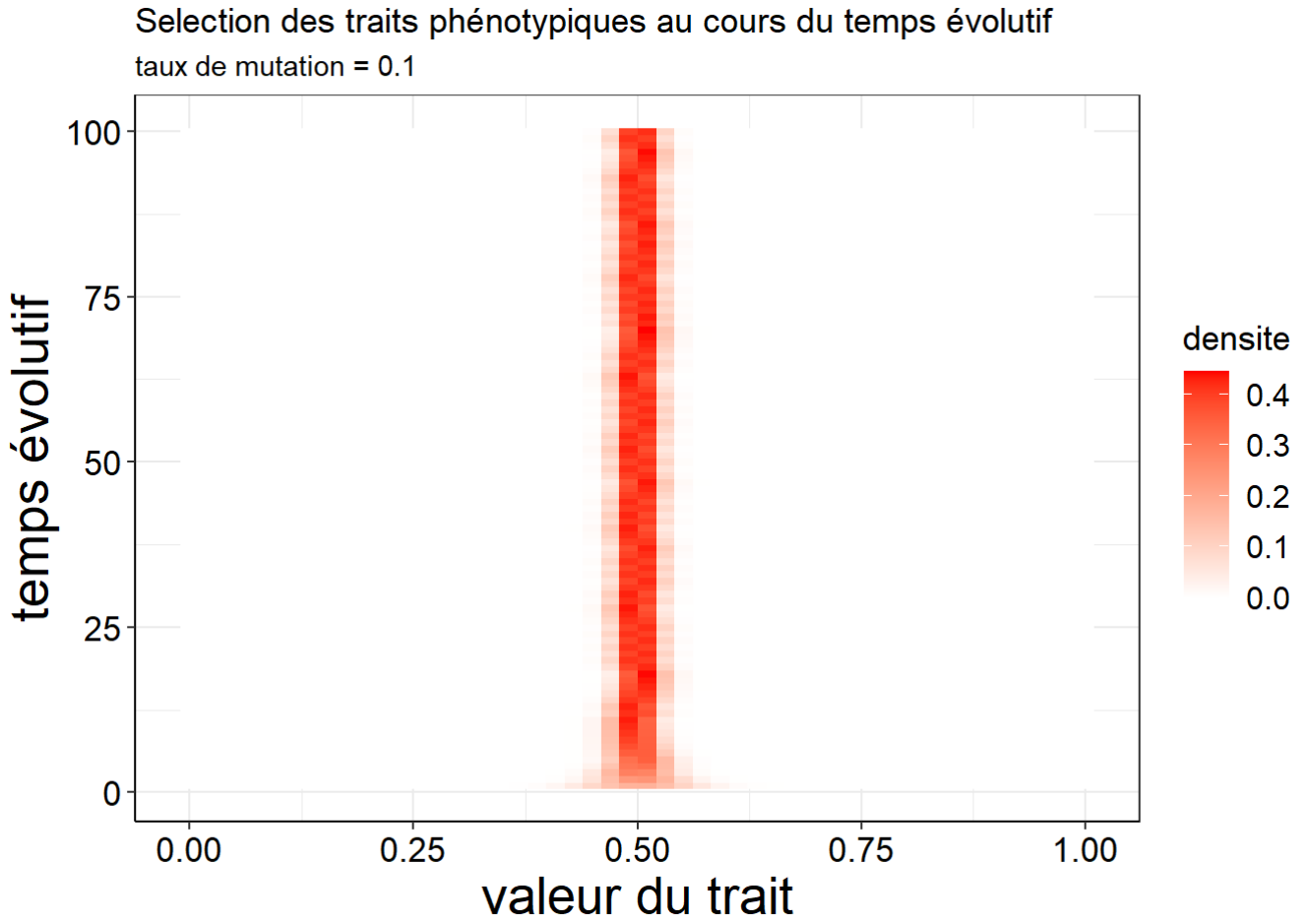
Une mutation est introduite dans le modèle. Une fraction de la population va être soumise à cette légère modification de la valeur du trait, avec une probabilité p. Nous avons effectué des mutations vers la gauche et vers la droite, c'est à dire qu'il y a un "retour en arrière" possible. Cette fois, nous travaillons sur des temps évolutifs longs, c'est à dire, M répétitions du temps t initialement choisi.

Pour tous les traits d'effectifs initiaux équivalents :

En concervant l'hypothèse qu'à l'origine, la population comporte autant d'individus que de phénotypes possible, voici-ce que l'on observe.

```r
#Initialisation des paramètres pour cette étude
sigma = 1
N = 50 #nombre de traits
M = 100 #temps évolutif répétition du temps t.
p = 0.1 #probanilité de mutation
X = seq(xmin, xmax, length.out = N) #valeur des traits
nbarre <- rep(K0/N, N) #Une même portion d'individu partagent chaque trait

#Création des matrice K et A (elles restent fixe malgré la mutation)
K = matrix(NA, nrow = length(X), ncol = length(X))
A = matrix(NA, nrow = length(X), ncol = length(X))

for(i in 1:length(X)){
  for(j in 1:length(X)){
    K[i,j] =  fonc.K(x = X[j],K0 = K0, lambda = lambda, x0 = x0)
    A[i,j] = fonc.a(x = X[i], y = X[j], sigma = sigma)
  }
}

paraM = list(r = r, K=K, A=A)

# Création de la matrice qui va contenir les valeurs des phénotypes au cours du temps évoluti
f M
nTmat = matrix(nrow=M, ncol=length(X))


for(m in 1:M){
  solution <- ode(y = nbarre, times = temps, func = EDO_LK, parms = paraM, method = "euler")
  nT = solution[nrow(solution),2:ncol(solution)]
  a = runif(1,0,1)
      if (a>0.5){ #mutation à droite
        nTmute <- nT[1:length(nT)-1]*p
        nT[1:length(nT)-1] = nT[1:length(nT)-1] - nTmute
        nT[2:length(nT)] = nT[2:length(nT)] + nTmute
      }
      else{ #mutation à gauche
        nTmute <- nT[2:length(nT)]*p
        nT[2:length(nT)] = nT[2:length(nT)] - nTmute
        nT[1:length(nT)-1] = nT[1:length(nT)-1] + nTmute
      }
  nbarre <- round(nT,3)
  nTmat[m,] <- round(nT,3)
}



## Affichage graphique

colnames(nTmat) = c(1:N) # on renome pour acceder à l'indice ensuite

nTmat = as.data.frame(nTmat) # on passe la matrice en tableau (obligatoire pour ggplot)

temps_evol <- seq(1, M, 1)
nTmat$time <- temps_evol # ajout du temps
```

```
# passage des colonnes en lignes pour permettre l'affichage sous ggplot
nTmat_long = nTmat%>%
  pivot_longer(-time, values_to = "densite", names_to = "phenotype")


nTmat_long$trait = X[ as.numeric(nTmat_long$phenotype)] # étiquette du trait

# Plots
ggplot(nTmat_long)+
  geom_raster(aes(trait, time,  fill=densite))+
  scale_fill_gradient2(low = "white" ,
                       high = "red")+
  ggtitle("Selection des traits phénotypiques au cours du temps évolutif", subtitle="taux de
mutation = 0.1")+
  xlab("valeur du trait")+
  ylab("temps évolutif")+
  main_theme
```



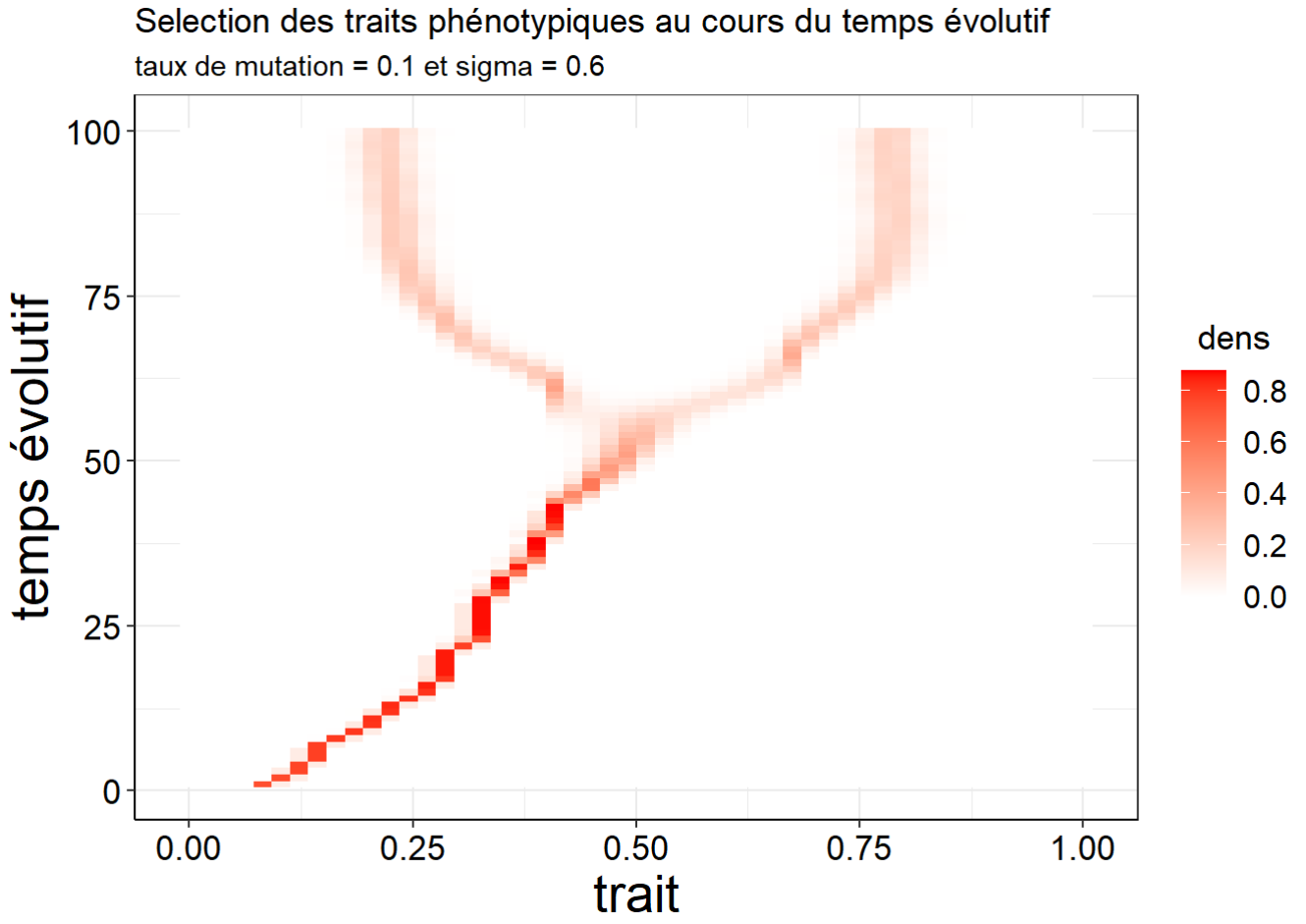
Comme précédemment, les phénotypes qui perdurent sont ceux qui présentent un bon compromis entre valeur selective et compétition, ici, avec $\sigma = 1$, les traits proches du trait maximisant K (soit 0.5) sont selectionnés.

Lorsque le trait initial est un trait de valeur selective inoptimale :

Dans ce cas-ci, afin de de mieux percevoir l'effet des mutations sur la valeur du phénotype, nous fixons la population initiale au phénotype avec une mauvaise valeur sélective. Il s'agit d'observer la fixation du trait de valeur sélective maximle au cours du temps évolutif. Nous choisissons $\sigma = 0.6$, afin d'appliquer une forte pression de compétition.

```
sigma = 0.6

z <- rep(0,N)
z[5] <- 1


nbarre <- z # cette fois, toute la population possède le trait inoptimal à l'état inital.

#Création des matrice K et A (elles restent fixe malgré la mutation)
K = matrix(NA, nrow = length(X), ncol = length(X))
A = matrix(NA, nrow = length(X), ncol = length(X))

for(i in 1:length(X)){
  for(j in 1:length(X)){
    K[i,j] =  fonc.K(x = X[j],K0 = K0, lambda = lambda, x0 = x0)
    A[i,j] = fonc.a(x = X[i], y = X[j], sigma = sigma)
  }
}

paraM = list(r = r, K=K, A=A)

# Création de la matrice qui va contenir les valeurs des phénotypes au cours du temps évoluti
f M
nTmat = matrix(nrow=M, ncol=length(X))

for(m in 1:M){
  solution <- ode(y = nbarre, times = temps, func = EDO_LK, parms = paraM, method = "euler")
  nT = solution[nrow(solution),2:ncol(solution)]
  a = runif(1,0,1)
      if (a>0.5){ #mutation à droite
        nTmute <- nT[1:length(nT)-1]*p
        nT[1:length(nT)-1] = nT[1:length(nT)-1] - nTmute
        nT[2:length(nT)] = nT[2:length(nT)] + nTmute
      }
      else{ #mutation à gauche
        nTmute <- nT[2:length(nT)]*p
        nT[2:length(nT)] =nT[2:length(nT)] - nTmute
        nT[1:length(nT)-1] = nT[1:length(nT)-1] + nTmute
      }
  nbarre <- round(nT,3)
  nTmat[m,] <- round(nT,3)
}



#### et on plot !

colnames(nTmat) = c(1:N) # on renome pour acceder à l'indice ensuite

nTmat = as.data.frame(nTmat) # on passe la matrice en tableau (obligatoire pour ggplot)

temps_evol <- seq(1, M, 1)
nTmat$time <- temps_evol # ajout du temps

# passage des colonnes en lignes pour permettre l'affichage sous ggplot
```

```
nTmat_long = nTmat%>%
  pivot_longer(-time, values_to = "dens", names_to = "phenotype")


nTmat_long$trait = X[ as.numeric(nTmat_long$phenotype)] # étiquette du trait

ggplot(nTmat_long)+
  geom_raster(aes(trait, time,  fill=dens))+
  scale_fill_gradient2(low = "white" ,
                       high = "red")+
  ggtitle("Selection des traits phénotypiques au cours du temps évolutif", subtitle="taux de
mutation = 0.1 et sigma = 0.6")+
  main_theme+
  ylab("temps évolutif")
```



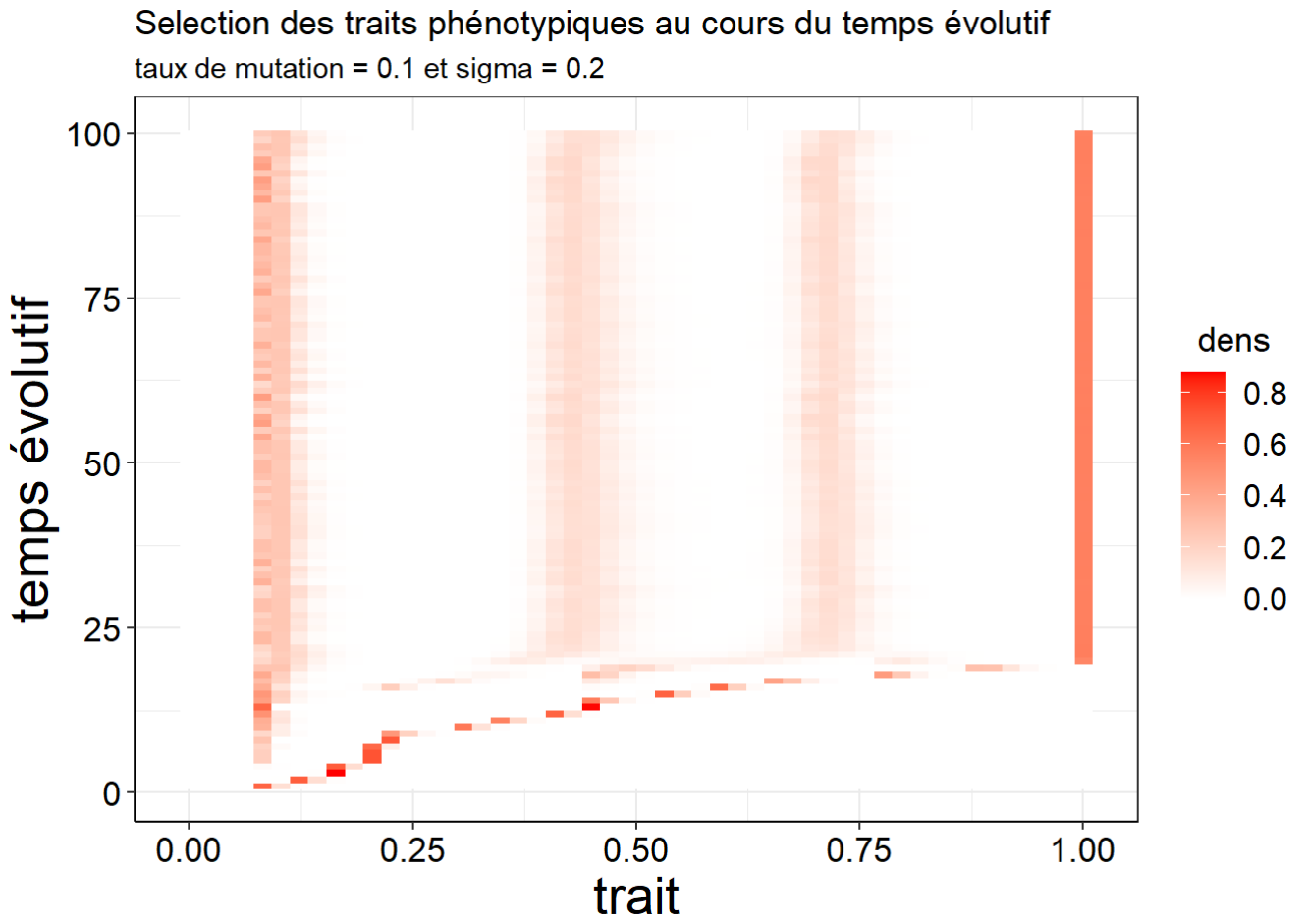
On voit clairement que le phénotype optimal, de valeur 0.5 est sélectionné au bout de 50 pas de temps évolutif M avec $t = 1000$ soit 50 000 pas de temps. De plus, aussitôt le phénotype optimal atteind, un branchement opère et deux phènotypes présentant un bon équilibre entre compétition et pression de sélection appraissent.

Influence de la valeur de sigma

Lorsque l'on modifie cette valeur de σ pour lever la pression de compétition, on constate la coexistence de deux phénotypes. Ici, $\sigma = 0.2$.

```
sigma = 0.2

z <- rep(0,N)
z[5] <- 1
nbarre <- z

#Création des matrice K et A (elles restent fixe malgré la mutation)
K = matrix(NA, nrow = length(X), ncol = length(X))
A = matrix(NA, nrow = length(X), ncol = length(X))

for(i in 1:length(X)){
  for(j in 1:length(X)){
    K[i,j] =  fonc.K(x = X[j],K0 = K0, lambda = lambda, x0 = x0)
    A[i,j] = fonc.a(x = X[i], y = X[j], sigma = sigma)
  }
}

paraM = list(r = r, K=K, A=A)

# Création de la matrice qui va contenir les valeurs des phénotypes au cours du temps évoluti
f M
nTmat = matrix(nrow=M, ncol=length(X))

for(m in 1:M){
  solution <- ode(y = nbarre, times = temps, func = EDO_LK, parms = paraM)
  nT = solution[nrow(solution),2:ncol(solution)]
  for (rgtrait in 1:length(nT)){
    if (nT[rgtrait] > 0.01){
      nTmute <- nT[1:length(nT)-1]*p
      a = runif(1,0,1)
      if (a>0.5){ #mutation à droite
        nT[1:length(nT)-1] = nT[1:length(nT)-1] - nTmute
        nT[2:length(nT)] = nT[2:length(nT)] + nTmute
      }
      #else{ #mutation à gauche
        #nT[1:length(nT)-1] = nT[1:length(nT)-1] + nTmute
        #nT[2:length(nT)] = nT[2:length(nT)] - nTmute
      #}
    }
  }
  nbarre <- nT
  nTmat[m,] <- nT
}



#### et on plot !


colnames(nTmat) = c(1:N) # on renome pour acceder à l'indice ensuite

nTmat = as.data.frame(nTmat)

temps_evol <- seq(1, M, 1)
nTmat$time <- temps_evol # ajout du temps
```

```
nTmat_long = nTmat%>%
  pivot_longer(-time, values_to = "dens", names_to = "phenotype")

nTmat_long$trait = X[ as.numeric(nTmat_long$phenotype)] # étiquette du trait

ggplot(nTmat_long)+
  geom_raster(aes(trait, time,  fill=dens))+
  scale_fill_gradient2(low = "white" ,
                       high = "red")+
  ggtitle("Selection des traits phénotypiques au cours du temps évolutif", subtitle="taux de
mutation = 0.1 et sigma = 0.2")+
  main_theme+
  ylab("temps évolutif")
```



Dans ce cas-ci, le branchement opère et, comme la pression de sélection est moins élevée, 4 phénotypes coexistent.