

Tutoriel Ecomeristem

Florian Larue & Grégory Beurier

2023-06-20

Installation du package recomeristem

Le modèle Ecomeristem existe sous deux formes, une application (QT) permettant l'introspection d'une simulation (à l'échelle de l'organe) et un package R utilisé pour l'analyse plus globale de simulations (à l'échelle de la plante/du peuplement).

Dans les faits, c'est essentiellement le package R qui est utilisé, notamment pour l'estimation de paramètres, l'idéotypage, etc.

Tout d'abord, installez le package recomeristem, disponible sur github à l'adresse suivante :

https://github.com/GBeurier/Ecomeristem_tutorial

Vérifiez bien la version de R installé sur votre machine afin de récupérer la bonne version du package recomeristem.

Copiez ensuite le dossier téléchargé et collez le dans le sous-dossier "library" de votre installation R (généralement : C:/Program Files/R/R-X.X.X/library)

Si tout s'est bien déroulé, le package recomeristem peut être chargé de la façon suivante :

```
library(recomeristem)
```

Fonctions de base du package

Le package recomeristem est un interfaçage entre le langage de programmation R et le modèle Ecomeristem à travers des fonctions de base permettant de communiquer avec le modèle (et des simulations).

1. Données d'entrée

La première étape consiste à importer les données d'entrée en utilisant les fonctions suivantes du package recomeristem :

```
PATH_TO_FOLDER <- "../data/"
PATH_TO_FILE <- "../data/vobs.txt"

weather <- recomeristem::getMeteo_from_files(PATH_TO_FOLDER)
param <- recomeristem::getParameters_from_files(PATH_TO_FOLDER)
obs <- recomeristem::get_clean_obs(PATH_TO_FILE)
```

```
# Les données météo
head(weather)
```

```
##      Temperature      Par Etp Irrigation P
## 1          15.3 12.67   0          0 0
## 2          15.1 12.52   0          0 0
## 3          15.5  8.49   0          0 0
## 4          16.8  9.38   0          0 0
## 5          18.6 12.44   0          0 0
## 6          18.3  8.24   0          0 0
```

```
# Les paramètres
head(param)
```

```
##      Name      Values
## 1  Assim_A 1.429200e+00
## 2  Assim_B 1.369200e+00
## 3 BeginDate 2.457532e+06
## 4   ETPmax 8.000000e+03
## 5   EndDate 2.457642e+06
## 6   Epsib 7.038570e+00
```

```
# (Optionnel) Les observations
head(obs)
```

```
##      app arealfel biomaero2 biominmainstem day lig      pht tillernb_1
## 1      8      NaN      NaN      NaN 21   5      NaN      2
## 2     11    118.10      NaN      NaN 28   8      NaN      3
## 3     14      NaN      NaN      NaN 35  10    40.00      NaN
## 4     15    308.83      NaN      NaN 42  11    75.06      2
## 5    NaN      NaN    29.01    8.233333 45 NaN      NaN      NaN
## 6     16      NaN      NaN      NaN 49  12   111.50      NaN
```

Alternativement, il est aussi possible d'importer vous-même les données, le modèle du package R attend en entrée des dataframes :

- Un dataframe pour la météo avec en colonnes les 5 variables attendues (PAR, T, irrig, P, ETP) et une ligne par jour (attention la première ligne doit correspondre au premier jour de simulation, qui correspond généralement à l'apparition de la première feuille). Seul le rayonnement (PAR) et la température (T) sont obligatoires pour lancer une simulation, les autres variables peuvent rester à zéro (désactivant les modules concernés)
- Un dataframe pour les paramètres avec en colonnes le nom des paramètres et leur valeur et une ligne par paramètre
- (Optionnel) Un dataframe pour les observations avec en colonnes les traits phénotypiques mesurés (dont le nom doit correspondre à une sortie du modèle Ecomeristem), une colonne "day" avec le jour de la mesure exprimé en jour depuis émergence et une ligne par date d'observation. Ces observations sont uniquement obligatoires pour l'estimation de paramètres

On verra dans l'étape 3 de cette partie du tutoriel comment lancer une simulation uniquement à l'aide de ces dataframes.

2. Initiation d'une simulation

Le seconde étape consiste à initialiser une simulation. Cette opération va créer un objet simulation qu'il sera ensuite possible de lancer pour récupérer les résultats, ou de le modifier (par exemple pour tester d'autres valeurs de paramètres). L'initialisation dans `recomeristem` se fait de la façon suivante :

```
recomeristem::init_simu(param, weather, obs, "simu1")
```

La fonction d'initialisation prend donc en argument les données d'entrées précédemment importés (paramètres, météo et observations) ainsi qu'un nom permettant d'identifier une simulation (plusieurs simulations peuvent être initiés en même temps). Cette étape est généralement utile pendant l'estimation de paramètres car elle accélère le temps d'exécution (il n'est plus nécessaire d'instancier les objets déjà créés et les informations utiles à la simulation sont modifiés à la volée).

3. Lancer une simulation

Lorsque la simulation a été initiée, elle peut être lancée à l'aide de la fonction suivante :

```
res <- recomeristem::launch_simu("simu1")
head(res)
```

```
##   app arealfel biomaero2 biominmainstem day lig      pht tillernb_1
## 1    6      NaN      NaN           NaN  21  4      NaN           3
## 2    8 59.4534      NaN           NaN  28  5      NaN           5
## 3   10      NaN      NaN           NaN  35  7 11.70694         NaN
## 4   13 254.0573      NaN           NaN  42 10 28.60080           8
## 5  NaN      NaN  29.32977      5.801408  45 NaN      NaN         NaN
## 6   15      NaN      NaN           NaN  49 11 58.82283         NaN
```

La fonction `launch_simu` renvoie un dataframe avec les résultats de simulations. La particularité de ce dataframe est qu'il ne contient que les valeurs simulés aux dates auxquels il y a une observations (et pour les traits phénotypiques observés).

Alternativement, une simulation peut être lancée sans être initiée à partir de dataframes (voir point 1. pour exemple) de la façon suivante :

```
res <- recomeristem::rcpp_run_from_dataframe(param, weather)
head(res[,1:5]) # head des 5 premières colonnes
```

```
##   app appstage arealfel      assim      biomaero
## 1    1         1      0 7.328911e-05 9.143039e-05
## 2    1         1      0 2.763594e-04 3.489091e-04
## 3    1         1      0 4.403839e-04 8.228735e-04
## 4    1         1      0 1.022354e-03 1.729167e-03
## 5    1         1      0 2.685102e-03 3.421845e-03
## 6    1         1      0 2.889575e-03 5.588454e-03
```

Dans ce cas, le dataframe retourné par la fonction `rcpp_run_from_dataframe()` est l'ensemble de la simulation (toutes les variables simulées pour l'ensemble des dates du premier jour de simulation au dernier jour).

Il est alors possible de réduire les résultats pour n'afficher que les résultats correspondant aux dates et traits du dataframe d'observations (similaire à ce qui est retourné par la fonction `launch_simu`) :

```
res <- recomeristem::rcpp_reduceResults(res, obs)
head(res)
```

```
##   app arealfel  biomaero2 biominmainstem lig      pht tillernb_1
## 1   6  19.5291  0.5110095      0.000000  3  4.702313          2
## 2   7  59.4534  2.1389935      0.000000  5  8.204627          5
## 3  10 121.0447  8.9604054      0.000000  7 11.706940          8
## 4  12 204.3030 21.0963449      2.990588  9 24.570243          8
## 5  13 254.0573 27.6512394      4.843028 10 34.984092          8
## 6  15 309.2284 34.4706876      8.135002 11 54.325518          8
```

A noter que si le fichier observations contient des colonnes (et/ou dates) qui ne sont pas présents dans le dataframe de résultats de simulation, il est aussi possible de les retirer automatiquement à l'aide de la fonction suivante :

```
obs$test <- 1234 # ajout d'une variable absente de res
head(obs)
```

```
##   app arealfel biomaero2 biominmainstem day lig      pht tillernb_1 test
## 1   8      NaN      NaN      NaN  21   5      NaN          2 1234
## 2  11  118.10      NaN      NaN  28   8      NaN          3 1234
## 3  14      NaN      NaN      NaN  35  10  40.00          NaN 1234
## 4  15  308.83      NaN      NaN  42  11  75.06           2 1234
## 5 NaN      NaN  29.01      8.233333 45 NaN      NaN          NaN 1234
## 6  16      NaN      NaN      NaN  49  12 111.50          NaN 1234
```

```
res <- recomeristem::rcpp_run_from_dataframe(param, weather)
obs <- recomeristem::rcpp_reduceVobs(obs, res)
head(obs)
```

```
##   app arealfel biomaero2 biominmainstem lig      pht tillernb_1
## 1   8      NaN      NaN      NaN   5      NaN          2
## 2  11  118.10      NaN      NaN   8      NaN          3
## 3  14      NaN      NaN      NaN  10  40.00          NaN
## 4  15  308.83      NaN      NaN  11  75.06           2
## 5 NaN      NaN  29.01      8.233333 NaN      NaN          NaN
## 6  16      NaN      NaN      NaN  12 111.50          NaN
```

Exemple d'utilisation : estimation de paramètres

Dans ce tutoriel on va voir l'utilisation du package recomeristem pour l'estimation de paramètres. Avant de commencer, chargement des packages et lecture des données :

```
# au besoin installez DEoptim (algorithme d'optimisation)
# install.packages("DEoptim")

# chargez les packages recomeristem et DEoptim
library(recomeristem)
library(DEoptim)
```

```

# lire les données
weather <- recomeristem::getMeteo_from_files(PATH_TO_FOLDER)
param <- recomeristem::getParameters_from_files(PATH_TO_FOLDER)
obs <- recomeristem::get_clean_obs(PATH_TO_FILE)

# lire les données d'estimation
estimParam <- read.csv(paste0(PATH_TO_FOLDER,"estimparam.csv"), sep=";")
bounds <- as.data.frame(estimParam[,c(2,3)]) # récupérer les bornes inf et sup
paramNames <- as.vector(estimParam[,1]) # récupérer les noms des paramètres

head(estimParam)

```

```

##              Param Lower Upper
## 1              Ict    0.5   2.5
## 2             MGR_init    6.0  14.0
## 3          phyllo_init   30.0  40.0
## 4 leaf_length_to_IN_length    0.1   0.2

```

L'estimation de paramètres consiste à trouver les valeurs de quelques paramètres génotypiques qui minimisent les écarts entre les simulations et les observations. L'algorithme d'optimisation (ici DEoptim) fournit à chaque itération un set de paramètres à tester (valeurs entre les bornes définies dans "bounds", pour les paramètres définies dans "paramNames")

1. Mise en place de l'optimisation

La première étape consiste donc à mettre en place l'optimisation :

```

# Définir une fonction qui calcule l'écart entre simulation et observations
error_fn <- function(obs, sim) {
  nmse <- ((obs - sim)/obs)^2
  nrmse <- sum((colSums(nmse, na.rm=T))/(colSums(!is.na(nmse))),na.rm=T)
  return(nrmse)
}

# Définir la fonction de coût qui reçoit des valeurs de paramètres en entrée
# et retourne l'erreur associée
isInit <- FALSE # permet de vérifier si une simulation est déjà initiée
fitness_fn <- function(p) {
  if(!isInit) { # si la simulation n'est pas encore initiée, l'initier
    recomeristem::init_simu(param, weather, obs, "sim1")
    isInit <- TRUE
  }

  # lancer la simu en modifiant les valeurs de paramètres
  sim <- recomeristem::launch_simu("sim1", paramNames, p)
  # calculer l'erreur
  error <- error_fn(obs, sim)
  # retourner l'erreur
  return(error)
}

```

2. Lancer l'optimisation

A partir de là, l'étape suivante est de lancer l'optimisation à l'aide de la fonction DEoptim :

```
res <- DEoptim(fn = fitness_fn, lower = bounds[,1], upper = bounds[,2],
              control = DEoptim.control(itermax = 20))
```

```
## Iteration: 1 bestvalit: 6.180086 bestmemit: 2.346237 13.516748 30.301173 0.151449
## Iteration: 2 bestvalit: 5.934880 bestmemit: 2.314472 13.277718 30.387412 0.163181
## Iteration: 3 bestvalit: 3.938402 bestmemit: 2.234655 13.551624 30.790689 0.187912
## Iteration: 4 bestvalit: 3.931097 bestmemit: 2.234655 13.551624 30.790689 0.190048
## Iteration: 5 bestvalit: 3.931097 bestmemit: 2.234655 13.551624 30.790689 0.190048
## Iteration: 6 bestvalit: 3.931097 bestmemit: 2.234655 13.551624 30.790689 0.190048
## Iteration: 7 bestvalit: 3.931097 bestmemit: 2.234655 13.551624 30.790689 0.190048
## Iteration: 8 bestvalit: 3.931097 bestmemit: 2.234655 13.551624 30.790689 0.190048
## Iteration: 9 bestvalit: 3.931097 bestmemit: 2.234655 13.551624 30.790689 0.190048
## Iteration: 10 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
## Iteration: 11 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
## Iteration: 12 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
## Iteration: 13 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
## Iteration: 14 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
## Iteration: 15 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
## Iteration: 16 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
## Iteration: 17 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
## Iteration: 18 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
## Iteration: 19 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
## Iteration: 20 bestvalit: 2.926974 bestmemit: 2.429409 13.983609 30.831851 0.190401
```

Ici pour l'exemple on ne lance que 20 itérations (itermax = 20), en réalité il faut un nombre bien plus important d'itérations pour trouver un optimum (dans les travaux récents, plutôt autour de 2500-5000 itérations).

DEoptim renvoie une liste avec plusieurs éléments. D'abord l'élément optim qui résume la valeur de chaque paramètre trouvé, la plus petite erreur trouvée, le nombre d'évaluations (à chaque itération plusieurs évaluations ont lieu) et le nombre d'itérations avant l'arrêt de l'algorithme. A noter que d'autres critères d'arrêts sont disponibles et que le nombre d'itération peut donc être plus faible que le paramètre itermax.

```
head(res$optim)
```

```
## $bestmem
##      par1      par2      par3      par4
## 2.4294094 13.9836085 30.8318514 0.1904014
##
## $bestval
## [1] 2.926974
##
## $nfeval
## [1] 42
##
## $iter
## [1] 20
```

Ensuite l'élément member qui résume dans le sous-élément bestmemit le meilleur candidat à chaque itération et dans le sous-élément pop les valeurs de paramètres pour chaque évaluation.

```
head(res$member$bestmemit)
```

```
##      par1      par2      par3      par4
## 1 2.430621 10.37112 30.79069 0.1375535
## 2 2.346237 13.51675 30.30117 0.1514489
## 3 2.314472 13.27772 30.38741 0.1631813
## 4 2.234655 13.55162 30.79069 0.1879115
## 5 2.234655 13.55162 30.79069 0.1900477
## 6 2.234655 13.55162 30.79069 0.1900477
```

3. Récupérer les résultats de l'estimation

Les résultats de l'estimation sont donc stockés dans la variable `res`, il suffit alors de remplacer les nouvelles valeurs de paramètres dans le dataframe d'origine et de relancer la simulation pour obtenir les résultats de simulation de l'optimum trouvé par l'algorithme.

```
# au besoin installer le package ggplot2
# install.packages(ggplot2)
```

```
library(ggplot2)
```

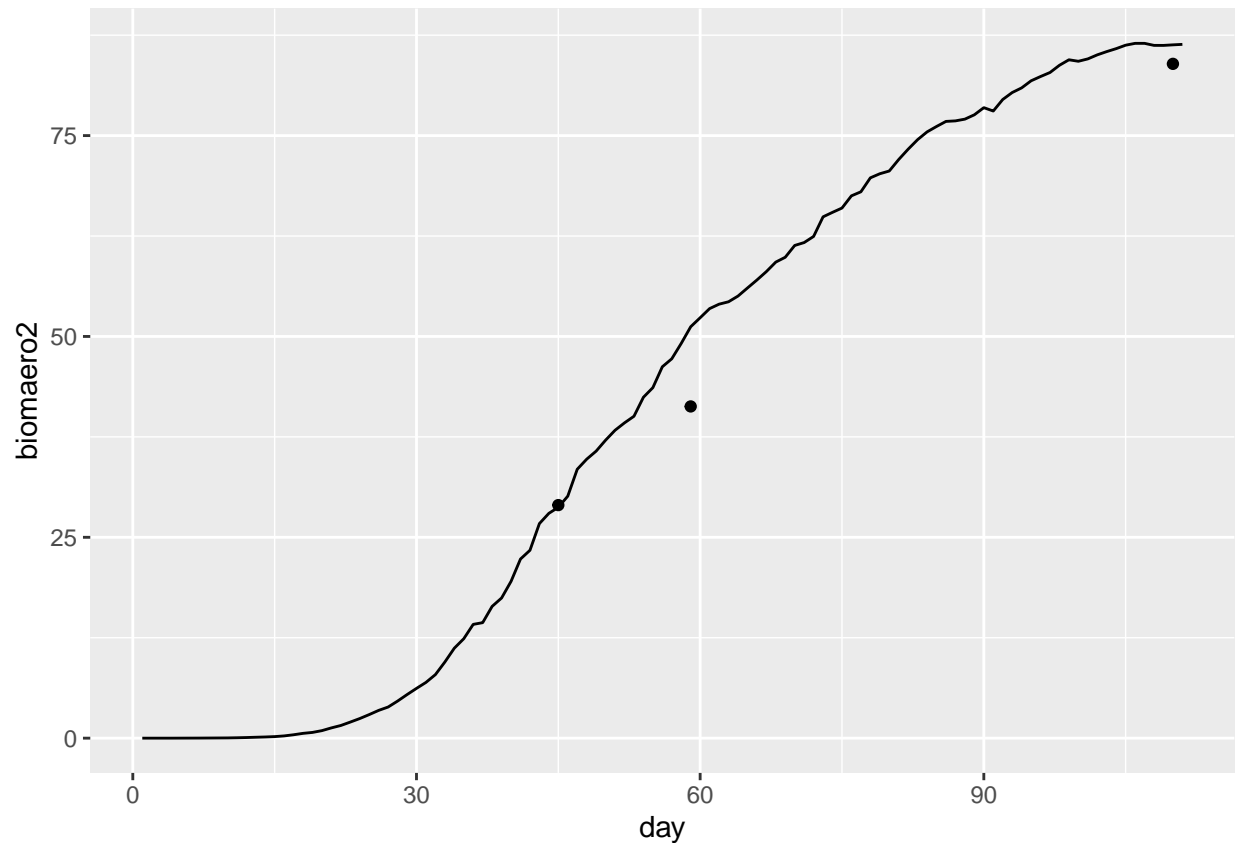
```
finalParam <- data.frame(Param = paramNames, Values = res$optim$bestmem,
                          Lower = bounds[,1], Upper = bounds[,2])
head(finalParam)
```

```
##              Param      Values Lower Upper
## par1              Ict  2.4294094   0.5   2.5
## par2              MGR_init 13.9836085   6.0  14.0
## par3              phyllo_init 30.8318514  30.0  40.0
## par4 leaf_length_to_IN_length 0.1904014   0.1   0.2
```

```
# remplacer dans le dataframe param original les nouvelles valeurs de paramètres
param[match(paramNames, param$Name), "Values"] <- res$optim$bestmem
```

```
# relancer la simulation avec les nouvelles valeurs de paramètres
finalSim <- recomeristem::rcpp_run_from_dataframe(param, weather)
finalSim$day <- as.numeric(row.names(finalSim))
```

```
# tracer un graphe avec la simulation d'un trait (ici la biomasse aérienne)
# et ajouter les points d'observations
plt <- ggplot(finalSim, aes(x = day, y=biomaero2)) + geom_line() +
  geom_point(data = obs, aes(x = day, y=biomaero2))
print(plt)
```

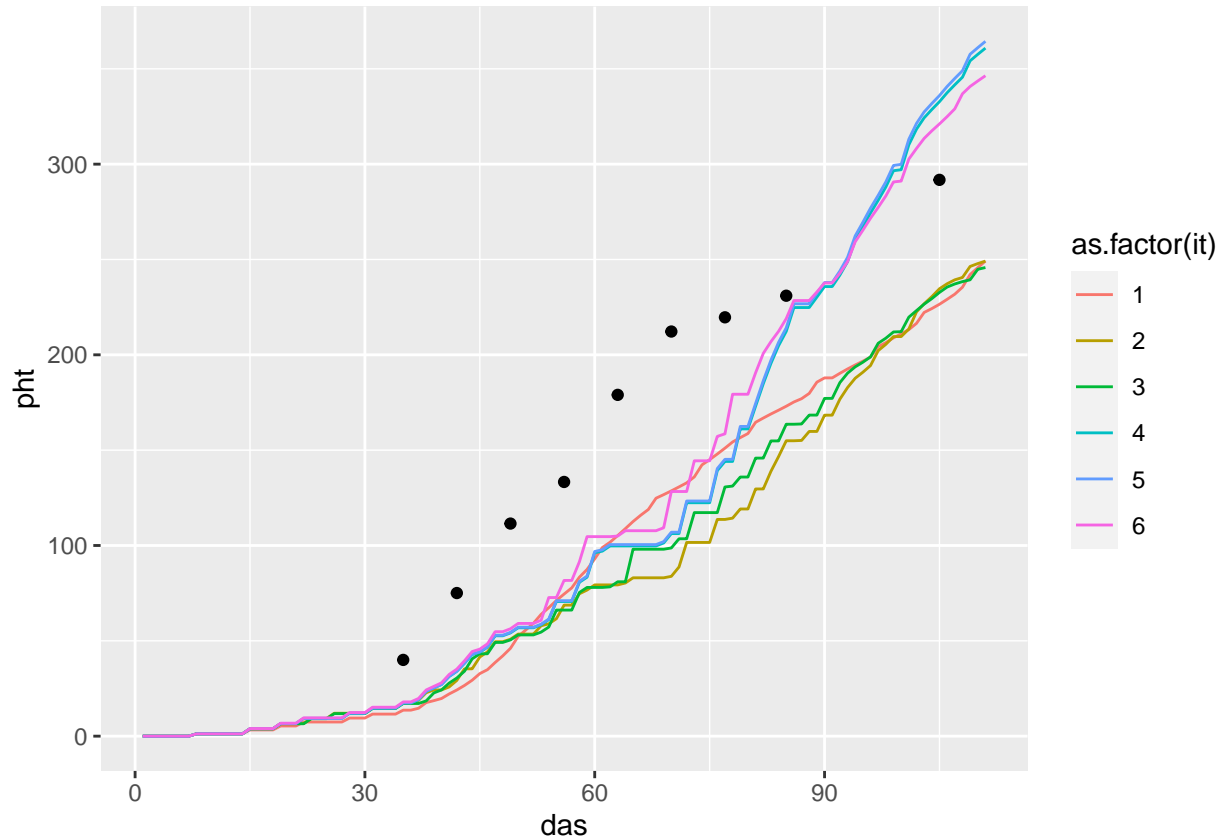


De façon similaire, on peut également regarder l'évolution de l'estimation en traçant le graphe du meilleur candidat à chaque itération, pour l'exemple on regarde ici un autre trait simulé/mesuré (pht = la hauteur de la plante) :

```
# récupérer les meilleurs candidats (uniques) sur l'ensemble des itérations
bestmems <- unique(res$member$bestmemit)

# construire un dataframe de résultats pour l'ensemble de ces candidats
bestRes <- data.frame()
for(i in 1:nrow(bestmems)) {
  param[match(paramNames, param$Name), "Values"] <- bestmems[i,]
  bestSim <- recomeristem::rcpp_run_from_dataframe(param, weather)
  bestRes <- rbind(bestRes, data.frame(das = as.numeric(row.names(bestSim)),
                                       pht = bestSim$pht, it = i))
}

# tracer le graphe
plt2 <- ggplot(bestRes, aes(x=das, y=pht, col=as.factor(it))) +
  geom_line() + geom_point(data = obs, aes(x = day, y=pht), inherit.aes = F)
print(plt2)
```

La dernière itération n'est pas spécialement le meilleur candidat pour le trait qu'on regarde ici (pht). Ce qui amène à un point important de l'estimation de paramètres présenté dans ce tutoriel : l'erreur est calculé sur un ensemble de traits phénotypiques (une RMSE normalisée, sur toutes les dates de mesures et tous les traits dans le fichier vobs.txt). De fait, si un candidat a une erreur globale plus faible, il sera sélectionné même si cela implique d'augmenter l'erreur sur un trait donné. Une solution envisageable est de faire de l'optimisation multi-critère. Cela dépasse le cadre de ce tutoriel mais pour les personnes intéressés, voir par exemple :

Tamaki, H., Kita, H., & Kobayashi, S. (1996, May). Multi-objective optimization by genetic algorithms: A review. In *Proceedings of IEEE international conference on evolutionary computation* (pp. 517-522). IEEE

Konak, A., Coit, D. W., & Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability engineering & system safety*, 91(9), 992-1007.

D'autres pistes d'améliorations peuvent être explorées avant de passer à l'optimisation multi-critère : en modifiant la fonction de calcul de l'erreur pour mettre de poids sur l'une ou l'autre variable prioritaire par exemple.

Lorsque les paramètres pour un génotype donné ont été estimés, il est alors possible d'utiliser le modèle Ecomeristem pour d'autres analyses tels que présentés pendant le séminaire.