# Tutoriel Samara

Florian Larue & Grégory Beurier & Michael Dingkuhn

2023-06-20

## Installation du package samara

Le modèle Samara existe sous deux formes, une application (QT) et un package R.

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

https://github.com/GBeurier/Samara_Tutorial

Vérifiez bien la version de R installé sur votre machine afin de récupérer la bonne version du package samara. Le fichier récupéré est un dossier compressé qu'il est possible d'installer directement depuis RStudio. Dans la fenêtre en bas à droite, dans l'onglet "packages", cliquez sur "install". Dans la fenêtre qui vient d'apparaitre, sélectionnez "Package Archive File (.zip; .tar.gz)" dans la première liste déroulante. Sélectionnez le dossier que vous venez de télécharger sur github et cliquez ensuite sur "installer".

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

```
library(samara)
```

## Fonctions de base du package

Le package samara est un interfaçage entre le langage de programmation R et le modèle Samara 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 du modèle. Samara attend en entrée deux dataframes :

- Un dataframe de données météo : date, température et humidité relative (min, max, moyenne), précipitations, vent, rayonnement, ensoleillement, evapotranspiration de référence. Les variables sont en colonnes et chaque ligne est une date.
- Un dataframe avec les paramètres : espèce, variétaux, sol et d'itinéraire technique. Les paramètres sont en colonnes et la valeur sur la première ligne.

Pour comparer les simulations avec des observations, nous allons également charger un dataframe avec des mesures faites au champ.

```r
PATH_TO_FOLDER <- "Data/"

weather <- read.csv(paste0(PATH_TO_FOLDER, "weather.csv"), sep=";")
param <- read.csv(paste0(PATH_TO_FOLDER, "param.csv"), sep=";")
obs <- read.csv(paste0(PATH_TO_FOLDER, "obs.csv"), sep=";")
```

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

```
##   wscode weatherdate tmin tmax  tmoy rhmin rhmax rhmoy rainfall windtot
## 1      5  31/05/2014 28.4 39.0 33.70    43    73  58.0      0.0       3
## 2      5  01/06/2014 25.2 39.0 32.10    41    81  61.0     16.4       2
## 3      5  02/06/2014 21.0 35.5 28.25    53    89  71.0      0.0       3
## 4      5  03/06/2014 26.0 37.7 31.85    37    85  61.0      0.9       1
## 5      5  04/06/2014 23.5 37.1 30.30    56    93  74.5      0.0       3
## 6      5  05/06/2014 27.0 28.7 27.85    84    96  90.0     32.5       4
##   radiation sunshine  eto
## 1        20      8.8 -999
## 2        20      9.6 -999
## 3        20      5.5 -999
## 4        20      8.2 -999
## 5        20      9.6 -999
## 6        20      8.8 -999
```

```r
# Les paramètres
head(param[,1:10])
```

```
##   X variety cropcode internodelengthmax leaflengthmax coeffleafwlratio   slamin
## 1 1      NA       NA           47.54431      879.4403          0.10359 0.001266
##   coeffleafdeath tilability coefftillerdeath
## 1       0.004866   0.105128         0.207589
```

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

```
##   obsplantdate nbjas    lai leavesnumber plantheight drymatabovegroundpop
## 1      2456847    28 1.0875       9.8750     19.1525              371.7475
## 2      2456860    41 0.9650      14.5625     50.0325             1258.9100
## 3      2456874    55 2.3875      17.9575     97.0100             1666.2850
## 4      2456888    69 3.3800      22.4375    151.1850             3767.4550
## 5      2456902    83 2.7550      20.2500    165.9500            11289.4433
## 6      2456932   113     NA           NA          NA                   NA
##   grainyieldpop drymatabovegroundpopfin
## 1            NA                     NA
## 2            NA                     NA
## 3            NA                     NA
## 4            NA                     NA
## 5            NA                     NA
## 6        3995.5                22372.5
```

## 2. Initiation d'une simulation

La 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 samara se fait de la façon suivante :

```
samara::init_sim_idx_simple(1, param, weather)
```

La fonction d'initialisation prend donc en argument les données d'entrées précédemment importées (paramètres, météos) ainsi qu'un entier permettant d'identifier une simulation (plusieurs simulations peuvent être initiées en même temps).

## 3. Lancer une simulation

Lorsque la simulation a été initiée, elle peut être lancée à l'aide de la fonction run_sim_idx en précisant l'identifiant de la simulation à lancer :

```
res <- samara::run_sim_idx(1)
head(res[,1:5])
```

```
##   ObsPlantDate NbJAS ApexHeight Assim AssimNotUsed
## 1      2456810    -9          0     0            0
## 2      2456811    -8          0     0            0
## 3      2456812    -7          0     0            0
## 4      2456813    -6          0     0            0
## 5      2456814    -5          0     0            0
## 6      2456815    -4          0     0            0
```

La fonction run_sim_idx renvoie un dataframe avec les résultats de simulations.

Il est possible de modifier une simulation initiée à l'aide de la fonction update_sim_idx qui prend en argument l'identifiant d'une simulation ainsi que les valeurs et les noms des paramètres à modifier :

```
# on lance une simulation en modifiant le paramètre phyllo_init à la valeur 35
samara::update_sim_idx(1, 40, c("phyllo"))
res <- samara::run_sim_idx(1)
head(res[,1:5])
```

```
##   ObsPlantDate NbJAS ApexHeight Assim AssimNotUsed
## 1      2456810    -9          0     0            0
## 2      2456811    -8          0     0            0
## 3      2456812    -7          0     0            0
## 4      2456813    -6          0     0            0
## 5      2456814    -5          0     0            0
## 6      2456815    -4          0     0            0
```

Ke dataframe retourné par la fonction run_sim_idx() 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 correspondants aux dates et traits du dataframe d'observations :

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

```
##   drymatabovegroundpop drymatabovegroundpopfin grainyieldpop       lai nbjas
## 1              305.422                    0.00         0.000 0.3365088    28
## 2             1283.888                    0.00         0.000 0.9969556    41
## 3             3485.811                    0.00         0.000 1.8288444    55
## 4             5879.721                    0.00         0.000 2.8621106    69
## 5             8550.889                    0.00      1897.269 3.1965677    83
## 6                0.000                11198.49         0.000 0.0000000   113
##   obsplantdate plantheight
## 1      2456847    332.2420
## 2      2456860    497.0908
## 3      2456874    834.7461
## 4      2456888   1169.7416
## 5      2456902   1289.3977
## 6      2456932      0.0000
```

A noter que si le fichier observations contient des colonnes (et/ou dates) qui ne sont pas présentes 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)
```

```
##   obsplantdate nbjas    lai leavesnumber plantheight drymatabovegroundpop
## 1      2456847    28 1.0875       9.8750     19.1525              371.7475
## 2      2456860    41 0.9650      14.5625     50.0325             1258.9100
## 3      2456874    55 2.3875      17.9575     97.0100             1666.2850
## 4      2456888    69 3.3800      22.4375    151.1850             3767.4550
## 5      2456902    83 2.7550      20.2500    165.9500            11289.4433
## 6      2456932   113     NA           NA          NA                    NA
##   grainyieldpop drymatabovegroundpopfin test
## 1            NA                     NA 1234
## 2            NA                     NA 1234
## 3            NA                     NA 1234
## 4            NA                     NA 1234
## 5            NA                     NA 1234
## 6        3995.5                22372.5 1234
```

```
obs <- samara::rcpp_reduceVobs(obs, res)
head(obs)
```

```
##   drymatabovegroundpop drymatabovegroundpopfin grainyieldpop    lai nbjas
## 1              371.7475                     NA            NA 1.0875    28
## 2             1258.9100                     NA            NA 0.9650    41
## 3             1666.2850                     NA            NA 2.3875    55
## 4             3767.4550                     NA            NA 3.3800    69
## 5            11289.4433                     NA            NA 2.7550    83
## 6                    NA                22372.5        3995.5     NA   113
##   obsplantdate plantheight
## 1      2456847     19.1525
```

```
## 2      2456860     50.0325
## 3      2456874     97.0100
## 4      2456888    151.1850
## 5      2456902    165.9500
## 6      2456932          NA
```

## Exemple d'utilisation : estimation de paramètres

Dans ce tutoriel on va utiliser le package samara pour l'estimation de paramètres. Avant de commencer, chargement des packages et lecture des données :

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

# 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
paramNames <- tolower(paramNames)
head(estimParam)
```

```
##                param  lower  upper
## 1 InternodeLengthMax  29.50  86.00
## 2      LeafLengthMax 232.00 910.00
## 3         TilAbility   0.10   0.40
## 4   CoeffPanicleMass   0.11   0.44
```

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 définis dans "paramNames")

### 1. Mise en place de l'optimisation

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

```r
# 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 associé
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
    samara::init_sim_idx_simple(1, param, weather)
    isInit <- TRUE
```

```
  }

  # Modifier les paramètres et lancer la simulation
  samara::update_sim_idx(1, p, paramNames)
  sim <- samara::run_sim_idx(1)
  # réduire le dataframe de résultats pour correspondre aux dimensions de obs
  sim <- samara::rcpp_reduceResults(sim, obs)
  # calculer l'erreur
  error <- error_fn(obs, sim)
  # retourner l'erreur
  return(error)
}
```

## 2. Lancer l'optimisation

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: 17.677190 bestmemit:   43.775244  264.759608    0.275895    0.343757
## Iteration: 2 bestvalit: 16.583645 bestmemit:   30.367106  261.106600    0.180730    0.372469
## Iteration: 3 bestvalit: 15.215903 bestmemit:   43.142319  236.048922    0.215187    0.243973
## Iteration: 4 bestvalit: 15.215903 bestmemit:   43.142319  236.048922    0.215187    0.243973
## Iteration: 5 bestvalit: 15.215903 bestmemit:   43.142319  236.048922    0.215187    0.243973
## Iteration: 6 bestvalit: 14.280462 bestmemit:   34.182512  236.094222    0.188323    0.310852
## Iteration: 7 bestvalit: 14.280462 bestmemit:   34.182512  236.094222    0.188323    0.310852
## Iteration: 8 bestvalit: 13.745101 bestmemit:   29.726142  236.094222    0.188323    0.310852
## Iteration: 9 bestvalit: 13.394648 bestmemit:   32.885875  245.298315    0.342924    0.369793
## Iteration: 10 bestvalit: 13.394648 bestmemit:   32.885875  245.298315    0.342924    0.369793
## Iteration: 11 bestvalit: 13.394648 bestmemit:   32.885875  245.298315    0.342924    0.369793
## Iteration: 12 bestvalit: 13.060121 bestmemit:   39.287091  237.712463    0.379300    0.382854
## Iteration: 13 bestvalit: 12.300918 bestmemit:   29.726142  236.094222    0.347465    0.290503
## Iteration: 14 bestvalit: 11.825230 bestmemit:   30.367106  232.144245    0.363304    0.372469
## Iteration: 15 bestvalit: 11.825230 bestmemit:   30.367106  232.144245    0.363304    0.372469
## Iteration: 16 bestvalit: 11.825230 bestmemit:   30.367106  232.144245    0.363304    0.372469
## Iteration: 17 bestvalit: 11.825230 bestmemit:   30.367106  232.144245    0.363304    0.372469
## Iteration: 18 bestvalit: 11.825230 bestmemit:   30.367106  232.144245    0.363304    0.372469
## Iteration: 19 bestvalit: 11.825230 bestmemit:   30.367106  232.144245    0.363304    0.372469
## Iteration: 20 bestvalit: 11.825230 bestmemit:   30.367106  232.144245    0.363304    0.372469
```

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 avec Samara, 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
##  30.3671063 232.1442448   0.3633043   0.3724691
##
## $bestval
## [1] 11.82523
##
## $nfeval
## [1] 840
##
## $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 37.55381 309.1884 0.3345839 0.3270408
## 2 43.77524 264.7596 0.2758954 0.3437566
## 3 30.36711 261.1066 0.1807302 0.3724691
## 4 43.14232 236.0489 0.2151868 0.2439729
## 5 43.14232 236.0489 0.2151868 0.2439729
## 6 43.14232 236.0489 0.2151868 0.2439729
```

## 3. Récuperer 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)
```

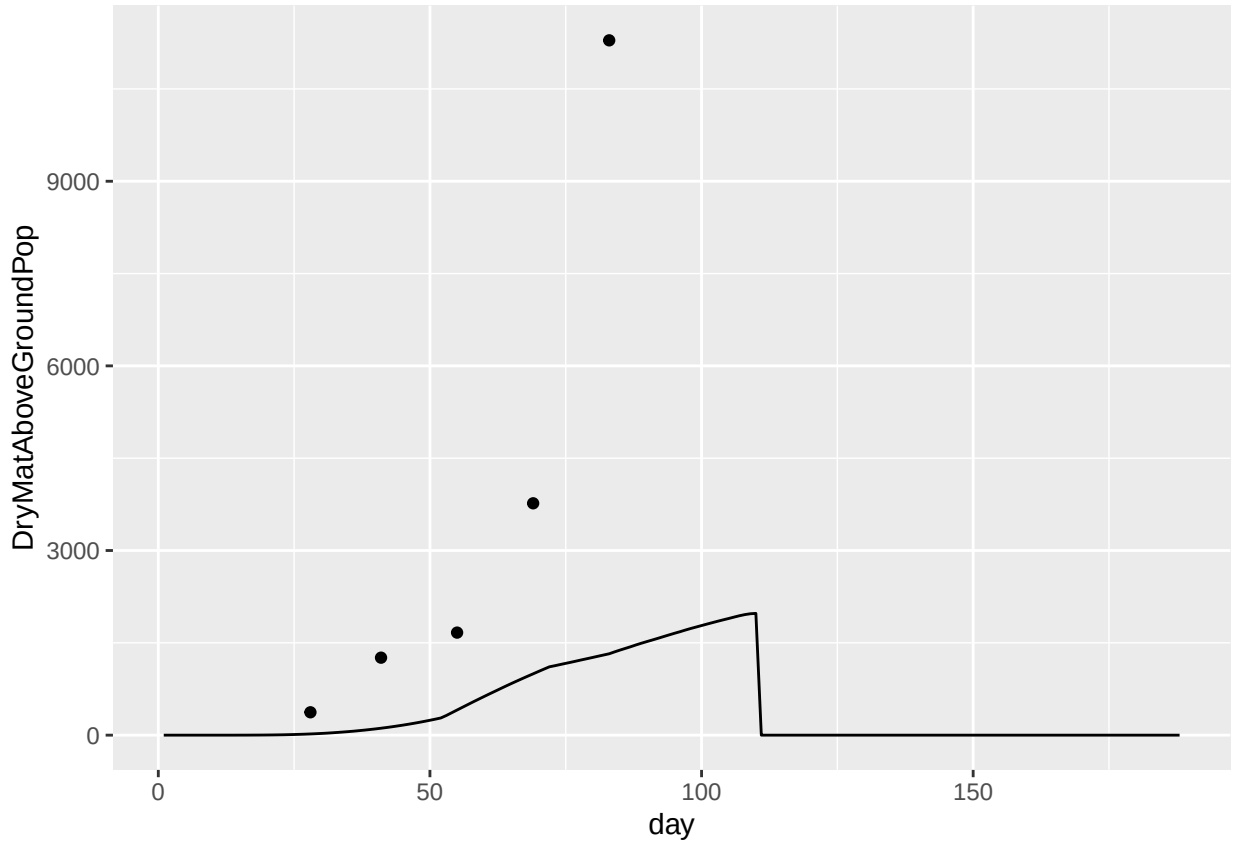
```
##                   Param      Values  Lower  Upper
## par1 internodelengthmax  30.3671063  29.50  86.00
## par2      leaflengthmax 232.1442448 232.00 910.00
## par3         tilability   0.3633043   0.10   0.40
## par4   coeffpaniclemass   0.3724691   0.11   0.44
```

```
# remplacer dans le dataframe param original les nouvelles valeurs de paramètres
param[1,match(paramNames, colnames(param))] <- res$optim$bestmem

# relancer la simulation avec les nouvelles valeurs de paramètres
samara::init_sim_idx_simple(2, param, weather)
```

```r
finalSim <- samara::run_sim_idx(2)
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=DryMatAboveGroundPop)) + geom_line() +
  geom_point(data = obs, aes(x = nbjas, y=drymatabovegroundpop))
print(plt)
```



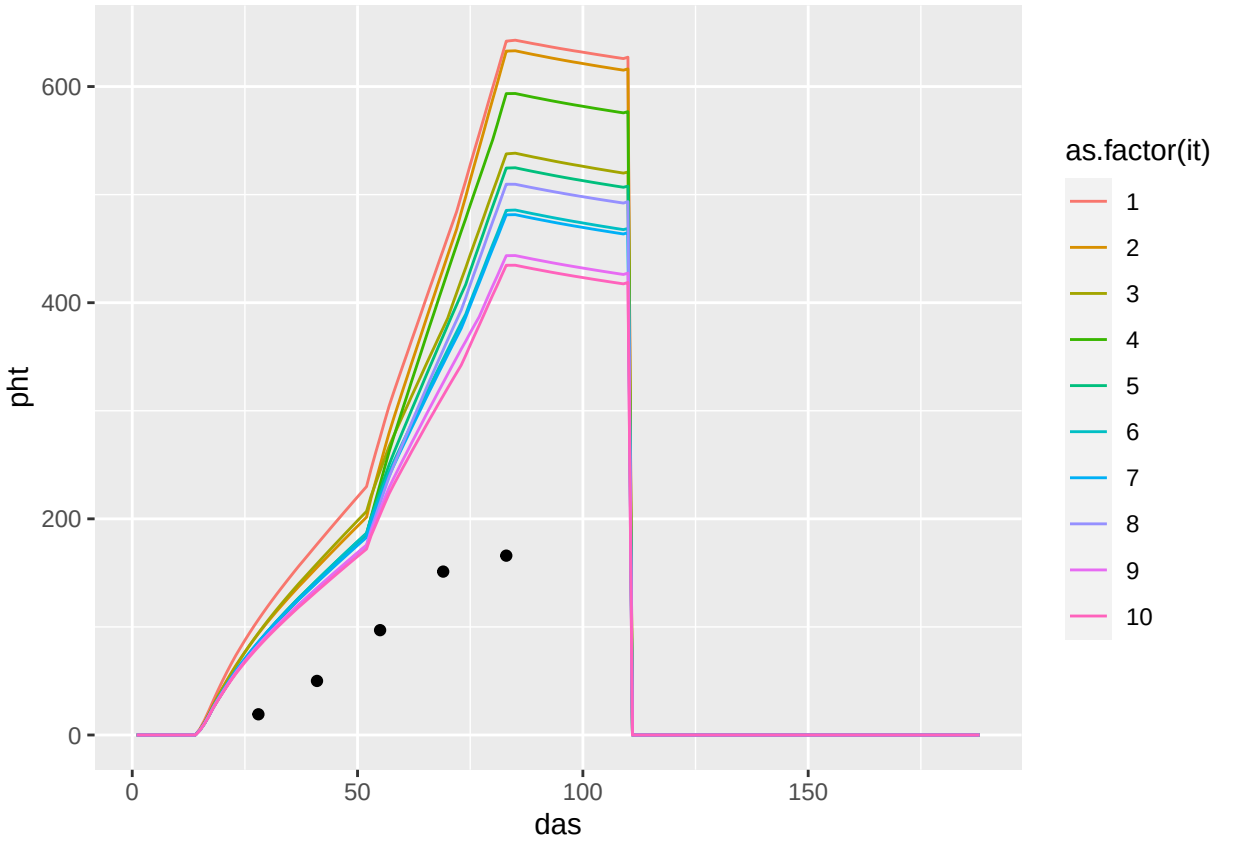

De façon similaire, on peut également regarder l'évolution de l'estimation en traçant le graphe du meilleur candidat à chaque itération, pour l'exemple on regarde ici un autre trait simulé/mesuré (pht = la hauteur de la plante) :

```r
# récuperer 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[1,match(paramNames, colnames(param))] <- bestmems[i,]

  samara::init_sim_idx_simple(2+i, param, weather)
  bestSim <- samara::run_sim_idx(2+i)
  bestRes <- rbind(bestRes, data.frame(das = as.numeric(row.names(bestSim)),
                                       pht = bestSim$PlantHeight, it = i))
}
```

```r
# tracer le graphe
plt2 <- ggplot(bestRes, aes(x=das, y=pht, col=as.factor(it))) +
  geom_line() + geom_point(data = obs, aes(x = nbjas, y=plantheight), 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 calculée 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 intéressées, 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 du poids sur l'un ou l'autre trait prioritaire par exemple.

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