# Proposition d'interface pour les wrappers de modèle dans CroptimizR

S. Buis

13/05/2020

## Problématique

L'objectif de cette note technique est de proposer une solution concernant le design de l'interface des wrappers de modèles de culture dans CroptimizR en prenant en compte les éléments de contraintes et de contexte des différents cas d'étude envisagés pour l'instant.

Les principaux objectifs sont les suivants :

- Faciliter au maximum le développement des wrappers pour une utilisation « standard » de CroptimizR (i.e. hors modes avancés), afin de permettre le développement de nombreux wrappers, y compris de façon occasionnelle pour des modèles jouets très simples.
- Donner un maximum de souplesse dans l'interface des wrappers pour permettre des utilisations avancées et des cas particuliers.
- Faciliter l'utilisation des wrappers et de CroptimizR en ayant des arguments faciles à manipuler
- Optimiser consommation mémoire et temps de calcul sachant que des cas d'étude peuvent être gourmands.

Les cas d'études considérés sont les suivants :

- Estimation de paramètre :
  - Cas standard : on estime des paramètres du modèle en minimisant un critère calculant un écart entre des observations d'une sélection de ses variables de sortie à certaines dates sur plusieurs situations et les valeurs simulées correspondantes.
  - Cas multi-variétal : idem mais les valeurs des paramètres dépendent des situations à simuler.
  - Cas simulations ou observations transformées dynamiquement : Il est possible que les observations ne correspondent pas aux variables simulées par le modèle, ou que la fonction objectif à minimiser soit composée de termes calculés à partir des simulations et observations. Dans ce cas, des template de fonctions de transformations prenant en entrée les résultats des simulations et les observations peuvent être utilisées pour redéfinir les valeurs simulées et/ou observées. Elles sont appelées systématiquement après l'exécution du wrapper et avant le calcul du critère.
  - Méthodes d'estimation : soit purement itératives (une valeur par paramètre est fournie au wrapper pour réaliser les simulations, puis le critère est évalué, le tout répété x fois), soit non itératives (un plan d'expérience numérique composé d'un jeu de valeurs pour les paramètres est fourni au wrapper en une seule fois puis le critère est calculé pour chaque individu du plan d'expérience), soit mixte.

=> Dans ce cas, le wrapper doit retourner les données simulées pour le jeu de valeurs de paramètres fourni, **a minima** pour les situations, dates et variables observées (ou celles nécessaires pour pouvoir effectuer les transformations de variables). En effet, ces données sont ensuite intersectées avec les observations avant le calcul du critère.

- Analyse d'incertitude et de sensibilité (AI / AS):
  - Un plan d'expérience numérique composé d'un jeu de valeurs pour les paramètres est fourni au wrapper en une seule fois. Ces valeurs peuvent éventuellement varier selon les situations à simuler. Le wrapper retourne les résultats simulés pour une sélection de situations, variables et dates ou stades.
- Multi-simulations :
  - Cas assez similaire au précédent mais en général composé d'un grand nombre de situations à simuler et d'un seul jeu de valeurs de paramètres par situation.

Trois questions sont traitées ici : comment designer les structures de données pour 1) le passage des valeurs des paramètres au wrapper, 2) la spécification des situations, variables et dates à simuler et pour lesquelles le wrapper doit renvoyer les résultats et 3) le stockage des sorties simulées.

# Valeurs des paramètres

Sous quelle forme transmettre au wrapper les valeurs des paramètres à utiliser pour effectuer les simulations ?

## Eléments de contraintes et de contexte :

- Les valeurs des paramètres peuvent varier selon les situations (e.g. estimation multi-variétale, multi-simulations) ... ou pas (estimation de paramètres standard) !
- Il peut y avoir un très grand nombre de valeurs de paramètres => ne pas les répliquer inutilement s'ils sont identiques pour plusieurs situations

## Solution proposée

Un `tibble` appelé `param_values` contenant une colonne par paramètre + 1 colonne "situation" optionnelle.

Si la colonne situation n'est pas fournie les simulations des différentes situations doivent être réalisées pour l'ensemble des situations. Ce sera typiquement le cas en analyse de sensibilité / incertitude et estimation de paramètres standard.

```
## # A tibble: 4 x 3
##      p1    p2    p3
##   <dbl> <dbl> <dbl>
## 1     1     1     1
## 2     2     1     1
## 3     3     2     1
## 4     4     2     1
```

La colonne situation sera fournie si les valeurs des paramètres dépendent des situations. Par exemple:

```
## # A tibble: 4 x 4
##   situation    p1    p2    p3
##   <chr>     <dbl> <dbl> <dbl>
## 1 sit1          1     1     1
## 2 sit2          1     1     1
## 3 sit3          2     2     1
## 4 sit4          2     2     1
```

dans le cas d'une estimation de paramètre multi-variétale, avec `p1` et `p2` paramètres variétaux, `p3` paramètre spécifique, et ("sit1", "sit2") et ("sit3",sit4") des situations observées sur 2 variétés différentes.

Un autre exemple typique sera le cas multi-simulation pour lequel des paramètres varient en fonction de la situation (par exemple condition initiale, stade forcé, ...):

```
## # A tibble: 4 x 4
##   situation    p1    p2    p3
##   <chr>     <dbl> <dbl> <dbl>
## 1 sit1          1     1     1
## 2 sit2          2     1     1
## 3 sit3          3     2     1
## 4 sit4          4     2     1
```

Dans le cas d'un plan d'expérience numérique qui dépend des situations, il y aura des réplications de valeur dans la colonne situation :

```
## # A tibble: 4 x 4
##   situation    p1    p2    p3
##   <chr>     <dbl> <dbl> <dbl>
## 1 sit1          1     1     1
## 2 sit1          2     1     1
## 3 sit2          3     2     1
## 4 sit2          4     2     1
```

## Eléments de Discussion

Cette solution a l'avantage :

- de gérer la plupart des cas sans duplication d'information,
- d'être simple à manipuler pour l'utilisateur qui souhaite exécuter des simulations directement avec le wrapper en forçant des valeurs de paramètres.

Mais a l'inconvénient de devoir traiter 2 formes dans les entrées du wrapper, donc complique un peu (mais pas beaucoup) son développement.

Le plan d'expérience numérique à appliquer par situation sera soit le tibble si pas de colonne "situation", soit une extraction du tibble en fonction de la colonne "situation":

```
# boucle sur les situations à simuler
for (sit in situation_list) {

  # extraction du plan d'expérience numérique en fonction de la situation à simuler
  if (is.null(param_values$situation)) {
```

```
    params <- param_values
  } else {
    params <- filter(param_values, situation==sit)
  }

  # boucle sur le plan d'experience numérique par situation
  for (iparams in 1:nrow(params)) {

    # run the model with parameters values defined by params[iparams,]

  }

}
```

## Gestion des situations, variables et dates

Comment spécifier au wrapper les situations, variables et dates pour lesquelles le wrapper doit renvoyer les résultats ?

### Eléments de contraintes et de contexte :

- Par défaut il est attendu que le wrapper fournisse les résultats pour l'ensemble des situations, variables et dates / stades simulés. Les situations à simuler seront cependant en général définies plus ou moins indirectement via l'argument `model_options` (e.g. en définissant un rép. de travail qui contient des fichiers correspondant à une liste de situations).
- La réduction des résultats renvoyés par le wrapper permet de faciliter leur manipulation par l'utilisateur (cas multi-simulation, AI/AS) qui n'a pas besoin de les filtrer
- Le coût mémoire peut dans certains cas être largement diminué si la liste des situations, variables et dates / stades renvoyées par le wrapper est réduite au minimum requis pour l'application.
- Le temps de calcul des wrappers peut également être diminué si un nombre réduit de variables est simulé. Cela dépend du modèle (e.g. requête dans une base de donnée pour lire les résultats pour APSIM, écriture et lecture +/- lourde dans les fichiers de sortie des modèles si on peut leur spécifier les variables à écrire, ...).
- Dans la plupart des cas d'estimation de paramètres la liste des situations et variables à simuler peut être fournie automatiquement puisqu'elle est déduite des observations. Ce n'est cependant pas le cas s'il est nécessaire d'effectuer des transformations (observations qui ne correspondent pas à des variables simulées par exemple).

## Solution proposée

Quatre arguments :

- sit_names : vecteur de nom des situations ...
- var_names : vecteur de nom des variables ...
- dates : vecteur des dates ...
- stages : vecteur des stades ... ... pour lesquelles le wrapper doit renvoyer les résultats

Tous ces arguments seraient optionnels : cela permet de construire rapidement un wrapper qui fonctionne.

Par exemple:

```
simple_wrapper <- function(param_values, model_options, ...) {

  workspace_path <- model_options$workspace_path

  # lecture de la liste des situations à simuler dans le workspace

  # simulation de toutes les variables du modèle pour toutes dates sur toutes les
  # situations pour les valeurs prescrites des paramètres

}
```

permettra facilement de faire de l'estimation de paramètre, de la multi-simulation ou même AI/AS. Dans ces deux derniers cas, l'utilisateur devra par contre lui-même extraire les résultats qui l'intéressent en termes de variables et de dates / stades. Ce wrapper sera cependant probablement assez peu efficace en consommation mémoire, voire en temps de calcul, dans certains cas (typiquement AI / AS).

Les noms des arguments sont fixés pour faciliter le passage d'un modèle à l'autre et pour fournir automatiquement les informations dans le cas estimation de paramètres (on aurait pu sinon laisser les développeurs de wrapper gérer cela via les options du wrapper).

Les arguments `sit_names` et `var_names` seront automatiquement fournis d'après les observations dans le cas estimation de paramètre. Si l'utilisateur souhaite faire une transformation des sorties simulées ou des observations et a besoin pour cela de simuler des variables non observées, alors il pourra définir ces arguments en entrée d'estim_param, ils seront passés tels quels au wrapper (peut-être via une solution du type `...` pour estim_param : tous les arguments non définis dans l'interface de estim_param pourraient être passés à l'ensemble des fonctions utilisateurs (wrapper, fonction de transformation, ...)).

Si `sit_names` n'est pas fourni, les résultats pour toutes les situations simulées doivent être retournés. Si `var_names` n'est pas fourni, les résultats pour toutes les variables simulées doivent être retournés. Si ni `dates`, ni `stades` ne sont fournis, les résultats pour toutes les dates doivent être retournés. Si `dates` et `stades` sont fournis, les résultats pour l'ensemble des deux doivent être retournés.

**On pourrait simplifier en ne mentionnant pas les arguments dates et stades, ou alors seulement dans un deuxième temps pour une version avancée du wrapper. En effet, ils ne sont pas utiles pour l'estimation de paramètres et à terme il est probable que les fonctions de multi-simulation et AI/AS prennent en charges ces arguments et filtrent les résultats des wrappers ...**

## Eléments de Discussion

Cette solution a l'avantage :

- de fonctionner dans les cas standards même si ces arguments ne sont pas gérés dans les wrappers à partir du moment où il simule au moins les situations, variables et dates nécessaires,
- d'être assez souple pour le développement des wrappers (seuls certains de ces arguments peuvent être pris en compte dans le wrapper en fonction des cas d'études traités, méthodes utilisées, optimisation du temps de calcul du wrapper ...),
- de ne pas demander à l'utilisateur de spécifier les situations et variables à simuler dans le cas estimation de paramètre sans transformation.

Mais a l'inconvénient :

- de complexifier le développement des wrappers si on veut qu'ils fonctionnent de façon optimale dans tous les cas (il faut traiter les différents arguments, leur caractère optionnel, compatibilité avec ce qui peut être réellement fourni par le modèle, ...),

- de multiplier la liste des arguments des wrappers
- de ne pas permettre de définir finement les variables et dates / stades en fonction des situations (comme le permet l'argument sit_var_date_mask actuel et que je propose d'abandonner). La liste des variables et dates / stades sera donc commune à l'ensemble des situations. Dans certains cas cela nécessitera donc que l'utilisateur fasse un filtrage des résultats a posteriori (mais pas plus complexe a priori que de construire sit_var_date_mask ...).

# Stockage des sorties simulées

Comment stocker les valeurs de différentes variables simulées pour différentes situations, dates / stades et valeurs de paramètres ?

## Eléments de contraintes et de contexte :

- la liste des variables peut différer selon les situations, et la liste des dates / stades selon les situations et variables.
- Comme il peut y avoir beaucoup de situations et/ou variables et/ou dates et/ou valeurs de paramètres, l'optimisation de la quantité de mémoire utilisée et du coût traitement de ces données est important
- Les tests réalisés pas Thomas Robine (cf. https://github.com/SticsRPacks/documentation/tree/master/tests_data_structure/Output_tests/output_tests.html ) montrent qu'un stockage sous forme d'un seul tibble est efficace en temps de traitement et ergonomique mais peut poser des problèmes mémoires pour des tailles importantes, mais réalistes, des différents cas d'étude. Par ailleurs, l'implémentation actuelle (liste de liste de data.frame) n'est pas satisfaisante en raison de son manque d'ergonomie.

## Solution proposée

Une solution, intermédiaire entre la solution actuelle et la solution « un seul tibble », est une liste de tibble par situation. Le tibble contiendrait les résultats des simulations des différentes variables (en colonne), dates et stades pour la situation concernée et pour l'ensemble des valeurs du plan d'expérience. Une colonne `DoE` contiendrait le numéro de ligne du plan d'expérience numérique, une colonne `Date` la date et une colone 'Stage' le (ou les) noms des stades pour lesquels la variable est demandée.

Dans un cas multi-simulation simple de 2 situations, pour lesquelles on demande 2 dates et 1 stade "rec" on aurait par exemple :

```
## $sit1
## # A tibble: 3 x 4
##     DoE Date                 var1 Stage
##   <dbl> <dttm>              <dbl> <chr>
## 1     1 2020-01-01 00:00:00   1.1 <NA>
## 2     1 2020-03-01 00:00:00   1.2 <NA>
## 3     1 2020-07-12 00:00:00   1.3 rec
##
## $sit2
## # A tibble: 3 x 4
##     DoE Date                 var1 Stage
##   <dbl> <dttm>              <dbl> <chr>
## 1     1 2020-01-01 00:00:00   1.1 <NA>
## 2     1 2020-03-01 00:00:00   1.2 <NA>
## 3     1 2020-05-01 00:00:00   1.3 rec
```

Dans un cas AI /AS avec une situation et deux stades, on aurait :

```
## $sit1
## # A tibble: 6 x 4
##     DoE Date                 var1 Stage
##   <dbl> <dttm>              <dbl> <chr>
## 1     1 2020-03-11 00:00:00   1.1 lev
## 2     1 2020-07-12 00:00:00   1.3 rec
## 3     2 2020-03-10 00:00:00   1.4 lev
## 4     2 2020-07-10 00:00:00   1.6 rec
## 5     3 2020-03-09 00:00:00   1.6 lev
## 6     3 2020-07-09 00:00:00   1.8 rec
```

### Eléments de Discussion

Cette solution a l'avantage :

- de résoudre les problèmes mémoires rencontrés avec la solution « un seul tibble » en répartissant les données simulées par variable en différents vecteurs (1 par situation)
- d'être plus ergonomique que la solution actuellement implémentée (liste de liste) :
  - extraction de simulations d'une variable pour une date ou un stade et différents individus du DoE (cas typique AI/AS) : `filter(data,Date=...)$var` ou `filter(data,Stage=...)$var`
  - extraction des simulations d'une variable à toutes (ou une liste de) dates (cas typique estimation de paramètres ou multi-simulation, 1 seul DoE) : `data$var` ou `filter(data,Date=...)$var`
  - L'extraction de simulations d'une variable pour différentes situations pourra se faire soit via la transformation de la liste en un seul tibble si la mémoire le permet (bind_rows) soit via une instruction type apply.

## Exemple d'un wrapper simple qui implémente ces solutions

```r
lai_toymodel <- function(year, jul_beg, jul_end, max_lai=8, inc_slope=5, dec_slope=2) {
  # Compute lai for a given year with a simple double-logistic function
  #
  # Arguments
  #   - year: simulation year
  #   - jul_beg, jul_end: beginning and end of simulation in julian days
  #   - max_lai: max value for lai
  #   - inc_slope and dec_slope: increasing and decreasing slope
  #
  # Value
  #   - lai: vector of simulated lai
  #   - date_laimax: date of maximum lai (POSIXct)
  #   - dates: vector of dates (POSIXct) for which lai is computed

  # julian day of maximum slopes are hard-coded in this simple example depending
  # on jul_beg and jul_end
  julday_incslope <- jul_beg+(jul_end-jul_beg)/3
```

```
  julday_decslope <- jul_end-(jul_end-jul_beg)/3
  jul_days <- jul_beg:jul_end

  lai <- max_lai * ( 1/(1+exp((julday_incslope-jul_days)/inc_slope)) -
                     1/(1+exp((julday_decslope-jul_days)/dec_slope)) )

  dates <- as.POSIXct(as.character(as.Date(jul_beg:jul_end,
                                           origin=paste0(year,"-01-01"))),
                      format = "%Y-%m-%d",tz="UTC")

  date_laimax <- dates[which.max(lai)]

  return(list(lai=lai,date_laimax=date_laimax, dates=dates))

}


laitm_simple_wrapper <- function(param_values=tibble(),
                                 model_options=list(year=c(2001),
                                                    jul_beg=c(70),
                                                    jul_end=c(150)), ...) {
  # A simple wrapper for lai_toymodel that does not handle sit_names, var_names,
  # dates and stages optionnal arguments
  #
  # Arguments
  #   - param_values: tibble containing the values of the lay_toymodel parameters
  #     to force among max_lai, inc_slope and dec_slope (depending on the situation
  #     to simulate if column situation is provided)
  #   - model_options: list containing vector of years and beginning/end julian days,
  #     that defines the situations to simulate
  #   - ...: mandatory since CroptimizR will give additional arguments not used here
  #
  # Value:
  #   A named list of tibble per situation (names of situations are the years).
  #   Each tibble contains columns:
  #      - Date (POSIXct dates of simulated results),
  #      - Stage (name of phenological stage(s) reached at the given date),
  #      - DoE (line number of the Design of Experiment for the corresponding situation),
  #      - One column per simulated variable (lai in this case)
  #
  # Details:
  #  - Runs the lai_toymodel for a set of situations defined in model_options
  #    (year, jul_beg, jul_end)
  #  - Forces the lai_toymodel parameters max_lai, inc_slope and dec_slope with
  #    the values given in param_values argument (that may depend on the situation to
  #    simulate)
  #  - Returns all the simulated values (i.e. all situations, variables and dates)
  #
  res <- list()
  for (isit in 1:length(model_options$year)) {

    jul_beg <- model_options$jul_beg[isit]
    jul_end <- model_options$jul_end[isit]
```

```
    year <- model_options$year[isit]
    sit=as.character(year) # in this example, the situation name is the year

    # Extract the design of experiment depending on the situation to simulate
    if ("situation" %in% names(param_values)) {
      params <- filter(param_values, situation==sit)
    } else {
      params <- param_values
    }

    # Loop on design of experiment per situation
    for (iparams in 1:max(nrow(params),1)) {

      res_laitm <- do.call('lai_toymodel', append(lapply(params,'[',iparams),
                                  list(year=year, jul_beg=jul_beg, jul_end=jul_end)))


      # Store "maximum lai" stage in the stage vector
      stage <- rep(NA,length(res_laitm$lai))
      stage[res_laitm$dates==res_laitm$date_laimax] <- "laimax"

      # fill the result variable
      res[[sit]] <- bind_rows(res[[sit]],
                              tibble(DoE=rep(iparams, length(res_laitm$lai)),
                                     Date=res_laitm$dates,
                                     Stage=stage,
                                     lai=res_laitm$lai))
    }
  }

  return(res)
}

# AI/AS, for default situation
# Two of the three lai_toymodel parameters are randomly sampled
n=10
res<-laitm_simple_wrapper(param_values=tibble(inc_slope=runif(n,2,7),
                                              max_lai=runif(n,3,10)))
# Results are extracted for one situation, variable and date but all DoE with the
# following command
dplyr::filter(res[["2001"]],Stage == "laimax")
```

```
## # A tibble: 10 x 4
##      DoE Date                Stage    lai
##    <int> <dttm>              <chr>  <dbl>
##  1     1 2001-04-24 00:00:00 laimax  4.21
##  2     2 2001-04-23 00:00:00 laimax  3.96
##  3     3 2001-04-25 00:00:00 laimax  8.78
##  4     4 2001-04-23 00:00:00 laimax  3.58
##  5     5 2001-04-26 00:00:00 laimax  5.94
##  6     6 2001-04-22 00:00:00 laimax  8.22
##  7     7 2001-04-24 00:00:00 laimax  5.58
##  8     8 2001-04-23 00:00:00 laimax  6.52
```

```
##  9     9 2001-04-21 00:00:00 laimax  4.88
## 10    10 2001-04-23 00:00:00 laimax  4.25
```

```
# Multi-simulation: the model is run for a set of different situations, and max_lai
# parameter is forced to a constant value different from the defautl one
res<-laitm_simple_wrapper(model_options=list(year=2000:2010,
                                             jul_beg=sample(60:80,11),
                                             jul_end=sample(130:170,11)),
                          param_values=tibble(max_lai=7))
# Results are extracted for lai at one stage but for all situations with the
# following command
bind_rows(res,.id="Situation") %>% dplyr::filter(Stage == "laimax")
```

```
## # A tibble: 11 x 5
##    Situation   DoE Date                Stage    lai
##    <chr>     <int> <dttm>              <chr>  <dbl>
##  1 2000          1 2000-05-04 00:00:00 laimax  6.74
##  2 2001          1 2001-04-13 00:00:00 laimax  6.37
##  3 2002          1 2002-04-16 00:00:00 laimax  6.31
##  4 2003          1 2003-04-28 00:00:00 laimax  6.87
##  5 2004          1 2004-04-24 00:00:00 laimax  6.82
##  6 2005          1 2005-04-29 00:00:00 laimax  6.74
##  7 2006          1 2006-04-21 00:00:00 laimax  6.80
##  8 2007          1 2007-04-18 00:00:00 laimax  6.31
##  9 2008          1 2008-05-05 00:00:00 laimax  6.74
## 10 2009          1 2009-04-15 00:00:00 laimax  6.56
## 11 2010          1 2010-04-22 00:00:00 laimax  6.71
```

```
# Parameter estimation: in this example we just simulate how the estim_param function
# will call the wrapper to see if it works
obs_list <- list(`2000`=tibble(Date=as.POSIXct(as.character(as.Date(c(90,100,110),
                                                                    origin="2000-01-01")),
                                               format = "%Y-%m-%d",tz="UTC"),
                               lai=c(1.8, 2.4, 2.6)))
sit_names <- names(obs_list)
var_names <- setdiff(unique(unlist(lapply(obs_list,names))),c("Date","Stage"))
laitm_simple_wrapper(param_values=tibble(max_lai=4.1),
                     model_options=list(year=2000:2002,
                                        jul_beg=c(66,62,71),
                                        jul_end=c(154,151,163)),
                     var_names=var_names,
                     sit_names=sit_names)
```

```
## $`2000`
## # A tibble: 89 x 4
##      DoE Date                Stage    lai
##    <int> <dttm>              <chr>  <dbl>
##  1     1 2000-03-07 00:00:00 <NA>  0.0116
##  2     1 2000-03-08 00:00:00 <NA>  0.0141
##  3     1 2000-03-09 00:00:00 <NA>  0.0173
##  4     1 2000-03-10 00:00:00 <NA>  0.0211
##  5     1 2000-03-11 00:00:00 <NA>  0.0257
##  6     1 2000-03-12 00:00:00 <NA>  0.0313
```

```
##  7     1 2000-03-13 00:00:00 <NA>  0.0382
##  8     1 2000-03-14 00:00:00 <NA>  0.0466
##  9     1 2000-03-15 00:00:00 <NA>  0.0567
## 10     1 2000-03-16 00:00:00 <NA>  0.0691
## # ... with 79 more rows
##
## $`2001`
## # A tibble: 90 x 4
##      DoE Date                Stage    lai
##    <int> <dttm>              <chr>  <dbl>
##  1     1 2001-03-04 00:00:00 <NA>  0.0108
##  2     1 2001-03-05 00:00:00 <NA>  0.0132
##  3     1 2001-03-06 00:00:00 <NA>  0.0161
##  4     1 2001-03-07 00:00:00 <NA>  0.0197
##  5     1 2001-03-08 00:00:00 <NA>  0.0240
##  6     1 2001-03-09 00:00:00 <NA>  0.0293
##  7     1 2001-03-10 00:00:00 <NA>  0.0358
##  8     1 2001-03-11 00:00:00 <NA>  0.0436
##  9     1 2001-03-12 00:00:00 <NA>  0.0531
## 10     1 2001-03-13 00:00:00 <NA>  0.0647
## # ... with 80 more rows
##
## $`2002`
## # A tibble: 93 x 4
##      DoE Date                Stage     lai
##    <int> <dttm>              <chr>   <dbl>
##  1     1 2002-03-13 00:00:00 <NA>  0.00888
##  2     1 2002-03-14 00:00:00 <NA>  0.0108
##  3     1 2002-03-15 00:00:00 <NA>  0.0132
##  4     1 2002-03-16 00:00:00 <NA>  0.0161
##  5     1 2002-03-17 00:00:00 <NA>  0.0197
##  6     1 2002-03-18 00:00:00 <NA>  0.0240
##  7     1 2002-03-19 00:00:00 <NA>  0.0293
##  8     1 2002-03-20 00:00:00 <NA>  0.0358
##  9     1 2002-03-21 00:00:00 <NA>  0.0436
## 10     1 2002-03-22 00:00:00 <NA>  0.0531
## # ... with 83 more rows
```

```
# All simulated situations, variables and dates are returned and will be intersected
# with obs_list in estim_param
```

En général, au contraire de ce cas simple, model_option contiendra plutot un path vers un répertoire de travail contenant les fichiers d'entrées du modèle, ce qui définira la liste (maximum) gdes situations à simuler.

## Idem mais en prenant en compte les arguments var_names, dates et stages

```
laitm_wrapper_V2 <- function(param_values=tibble(),
                             model_options=list(year=c(2001),
                                                jul_beg=c(70),
```

```
                                                    jul_end=c(150)),
                            sit_names=NULL, var_names=NULL, dates=NA, stages="") {
# A wrapper for lai_toymodel that handles sit_names, var_names, dates and stages
# optionnal arguments
#
# Arguments
#   - param_values: tibble containing the values of the lay_toymodel parameters
#     to force among max_lai, inc_slope and dec_slope (depending on the situation
#     to simulate if column situation is provided)
#   - model_options: list containing vector of years and beginning/end julian days,
#     that defines the situations to simulate
#   - sit_names: list of situations for which results must be returned (optional,
#     returns results for all simulated situations if not given)
#   - var_names: list of variables for which results must be returned (optional,
#     returns results for all simulated variables if not given)
#   - dates: list of dates for which results must be returned (optional, returns
#     results for all dates if neither dates nor stages are given, returns only
#     results at given dates if dates is given but not stages, returns results for
#     both given dates and stages otherwise)
#   - stages: list of stages  for which results must be returned (optional, returns
#     results for all dates if neither dates nor stages are given, returns only
#     results at given stages if stages is given but not dates, returns results for
#     both given dates and stages otherwise)
#
# Value:
#   A named list of tibble per situation (names of situations are the years).
#   Each tibble contains columns:
#      - Date (POSIXct dates of simulated results),
#      - Stage (name of phenological stage(s) reached at the given date),
#      - DoE (line number of the Design of Experiment for the corresponding situation),
#      - One column per simulated variable (lai in this case)
#
# Details:
#  - Runs the lai_toymodel for a set of situations defined in sit_names (if given)
#    or model_options (year, jul_beg, jul_end)
#  - Forces the lai_toymodel parameters max_lai, inc_slope and dec_slope with
#    the values given in param_values argument (that may depend on the situation to
#    simulate)
#  - Returns all OR A SELECTION OF the simulated values (depending on var_names,
#    dates and stages arguments)
#
res <- list()

if (is.null(sit_names)) sit_names <- model_options$year

for (isit in which(as.character(model_options$year) %in% sit_names)) {

  jul_beg <- model_options$jul_beg[isit]
  jul_end <- model_options$jul_end[isit]
  year <- model_options$year[isit]
  sit=as.character(year) # in this example, the situation name is the year

  # Extract the design of experiment depending on the situation to simulate
```

```
    if ("situation" %in% names(param_values)) {
      params <- filter(param_values, situation==sit)
    } else {
      params <- param_values
    }

    # Loop on design of experiment per situation
    for (iparams in 1:max(nrow(params),1)) {

      res_laitm <- do.call('lai_toymodel', append(lapply(params,'[',iparams),
                                  list(year=year, jul_beg=jul_beg, jul_end=jul_end)))


      # Store "maximum lai" stage in the stage vector
      stage <- rep(NA,length(res_laitm$lai))
      stage[res_laitm$dates==res_laitm$date_laimax] <- "laimax"

      # fill the result variable
      res_tmp <- tibble(DoE=rep(iparams, length(res_laitm$lai)),
                        Date=res_laitm$dates,
                        Stage=stage,
                        lai=res_laitm$lai)

      if (!is.na(dates[1]) | stages[1]!="")
        res_tmp <- dplyr::filter(res_tmp, Date == dates | Stage == stages)
      if (!is.null(var_names)) res_tmp <- select(res_tmp, DoE, Date, Stage,!!var_names)

      res[[sit]] <- bind_rows(res[[sit]],res_tmp)


    }
  }

  return(res)

}


# AI/AS, for default situation
n=10
laitm_wrapper_V2(param_values=tibble(inc_slope=runif(n,2,7),
                                     max_lai=runif(n,3,10)),
                 var_names="lai", stages="laimax")
```

```
## $`2001`
## # A tibble: 10 x 4
##      DoE Date                Stage    lai
##    <int> <dttm>              <chr>  <dbl>
##  1     1 2001-04-26 00:00:00 laimax  4.81
##  2     2 2001-04-25 00:00:00 laimax  4.79
##  3     3 2001-04-23 00:00:00 laimax  8.98
##  4     4 2001-04-25 00:00:00 laimax  3.62
##  5     5 2001-04-25 00:00:00 laimax  8.36
```

```
##  6     6 2001-04-25 00:00:00 laimax  3.18
##  7     7 2001-04-26 00:00:00 laimax  7.58
##  8     8 2001-04-26 00:00:00 laimax  5.09
##  9     9 2001-04-21 00:00:00 laimax  4.04
## 10    10 2001-04-24 00:00:00 laimax  9.52
```

```
# In this case, results are directly returned in the expected format

# multi-simulation : change year, jul_begin and jul_end but keep constant parameters
res<- laitm_wrapper_V2(model_options=list(year=2000:2010,
                                          jul_beg=sample(60:80,11),
                                          jul_end=sample(130:170,11)),
                       param_values=tibble(max_lai=7),
                       var_names="lai", stages="laimax")
# In this case, a simple bind_rows returns the expected format
bind_rows(res,.id="Situation")
```

```
## # A tibble: 11 x 5
##    Situation   DoE Date                Stage    lai
##    <chr>     <int> <dttm>              <chr>  <dbl>
##  1 2000          1 2000-04-30 00:00:00 laimax  6.92
##  2 2001          1 2001-04-22 00:00:00 laimax  6.21
##  3 2002          1 2002-04-25 00:00:00 laimax  6.83
##  4 2003          1 2003-05-02 00:00:00 laimax  6.87
##  5 2004          1 2004-04-21 00:00:00 laimax  6.39
##  6 2005          1 2005-04-26 00:00:00 laimax  6.54
##  7 2006          1 2006-04-17 00:00:00 laimax  6.68
##  8 2007          1 2007-04-25 00:00:00 laimax  6.75
##  9 2008          1 2008-04-28 00:00:00 laimax  6.82
## 10 2009          1 2009-05-06 00:00:00 laimax  6.76
## 11 2010          1 2010-04-17 00:00:00 laimax  6.72
```

```
# Parameter estimation
obs_list <- list(`2000`=tibble(Date=as.POSIXct(as.character(as.Date(c(90,100,110),
                                                                    origin="2000-01-01")),
                                               format = "%Y-%m-%d",tz="UTC"),
                               lai=c(1.8, 2.4, 2.6)))
sit_names <- names(obs_list)
var_names <- setdiff(unique(unlist(lapply(obs_list,names))),c("Date","Stage"))
laitm_wrapper_V2(param_values=tibble(max_lai=4.1),
                 model_options=list(year=2000:2002,
                                    jul_beg=c(66,62,71),
                                    jul_end=c(154,151,163)),
                 var_names=var_names,
                 sit_names=sit_names)
```

```
## $`2000`
## # A tibble: 89 x 4
##      DoE Date                Stage    lai
##    <int> <dttm>              <chr>  <dbl>
##  1     1 2000-03-07 00:00:00 <NA>  0.0116
##  2     1 2000-03-08 00:00:00 <NA>  0.0141
##  3     1 2000-03-09 00:00:00 <NA>  0.0173
```

```
##  4     1 2000-03-10 00:00:00 <NA>  0.0211
##  5     1 2000-03-11 00:00:00 <NA>  0.0257
##  6     1 2000-03-12 00:00:00 <NA>  0.0313
##  7     1 2000-03-13 00:00:00 <NA>  0.0382
##  8     1 2000-03-14 00:00:00 <NA>  0.0466
##  9     1 2000-03-15 00:00:00 <NA>  0.0567
## 10     1 2000-03-16 00:00:00 <NA>  0.0691
## # ... with 79 more rows
```

```
# In this case, only the observed situation and variable is returned (but for all
# dates) and will be intersected with obs_list in estim_param
# If the user provide var_names to estim_param, it will be given to the wrapper
# instead of this computed from obs_list
```

## Remarques

Les cas intercropping et successions culturales ne sont pas (encore) pris en compte dans cette réflexion.

Je n'ai pas non plus encore traité le cas des warnings / erreurs détectés dans le wrapper.