

TP optimisation globale – Part 1

AIT BAALI Hamza, AMINI Nada, ESSAYEGH Nour, LEFDALI Rida

- **Connaissances apportées par ce TP :
- A) Comment tester un optimiseur global
- B) Optimiseur local versus optimiseur global
- C) Observer l'effet de la variance (le « pas ») dans un optimiseur stochastique simple

A)

MAIN PROGRAMS The 3 files 3Dplots.R, main4tests.R, postproc_tests_optimizers.R :

- 3Dplots.R : make 3D plots of 2D functions and superpose 1 trajectory of an optimizer.
- main4tests.R : repeat optimizations to study the behavior of a particular optimizer with a particular objective function.
- postproc_tests_optimizers.R : do summary plots of one or many repeated optimizations that were previously performed with main4test.R .
- In main4tests.R describe the function you want to optimize (with the string nameoffun) and the optimizer you want to use (with nameofoptim). Also choose the number of repetitions of the optimization (no_tests) In postproc_tests_optimizers.R compare the results of different optimizations by filling the vector of strings filenames.

BASIC USE

- In 3Dplots.R and main4tests.R :
- fill in the name of the objective function (string nameoffun) and the name of the optimizer (string nameofoptim). Possible names for nameoffun are the functions given in test_functions.R . Possible names for nameofoptim are the optimization algorithms R functions present in the directory. Examples of both are given in comments.
- fill in other characteristics of the objective function: number of dimensions (variable zdim, which must be = 2 in 3Dplots.R), position of the optimum (variable glob_xstar), noise characteristics (glob_noisy, glob_tau)
- fill in parameters of the optimizer: they are all fields of the param list, cf. param specific to each optimizer. Typically, one finds lower and upper bounds on the optimization variables (LB and UB), and maximum number of calls to the objective function (budget), and the rest depends on nameofoptim.
- then execute the file.
- The results of 3Dplots.R are R plots which are also saved as image files fileofplot.png and contour.png . You may want to copy these files elsewhere or change their names otherwise they will be overwritten by the next call to 3Dplots.R .
- The results of main4tests.R are stored in the files "nameofoptim""nameoffun"{some characteristic params}.RData with plots in "nameofoptim""nameoffun"{some characteristic params}.pdf . The data in the .RData file is meant to be reloaded later (cf below). The names of the result files are weakly unique, it is probably safer to change them (or move them to another directory) before using main4tests.R again.
- In postproc_tests_optimizer.R :
- fill in the names of the .RData files you want to process in order to compare the performance of the optimizers (string vector filenames). Source the file and enjoy.

-
- **LIST OF FILES**

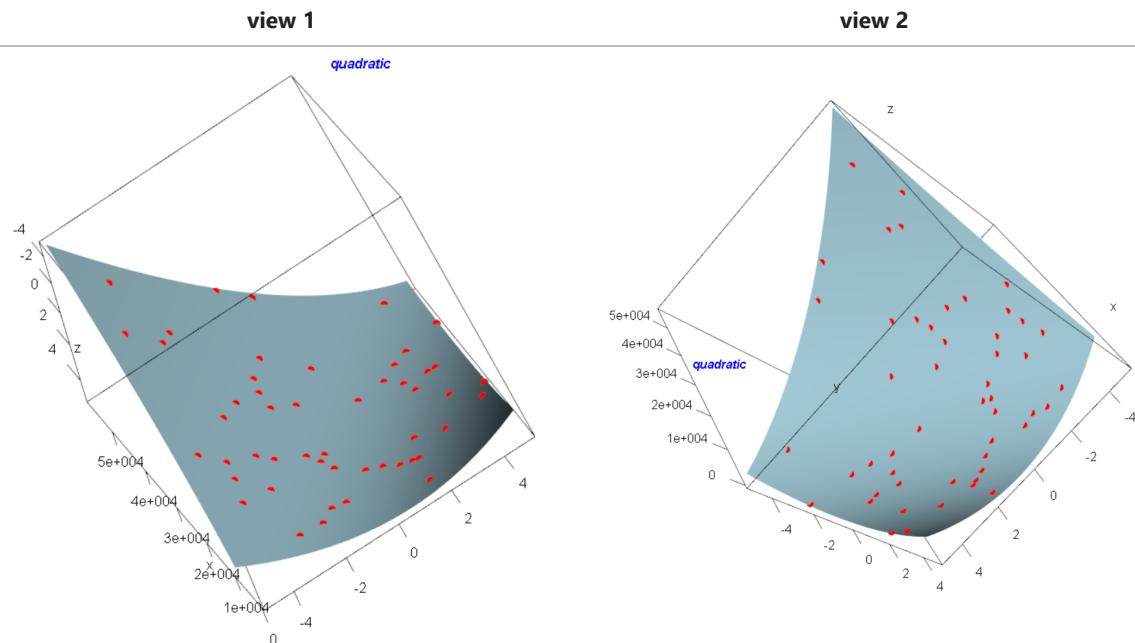
- **3Dplots.R** : plots 2D functions in 3D and superimposes optimizer trajectories
- **cmaes.R** : CMA-ES optimization algorithm
- **lbfgs.R** : memory light BFGS optimization algorithm
- **main4tests.R** : main program to call optimizer. Can perform many optimizations.
- **normal_search.R** : a fixed step ES-(1+1) search algorithm
- **of_wrapper.R** : wraps calls to the objective function in order to store all calls thru global variables when necessary
- **postproc_tests_optimizers.R** : creates plots to compare different optimizers
- **readme.txt : some explanations
- **random_search.R** : a random search algorithm
- **utility_optim_functions.R** : set of utility functions for plotting and casting our data structures
- **test_functions.R** : a set of test functions (potentially noisy) with control on the location of the optimum

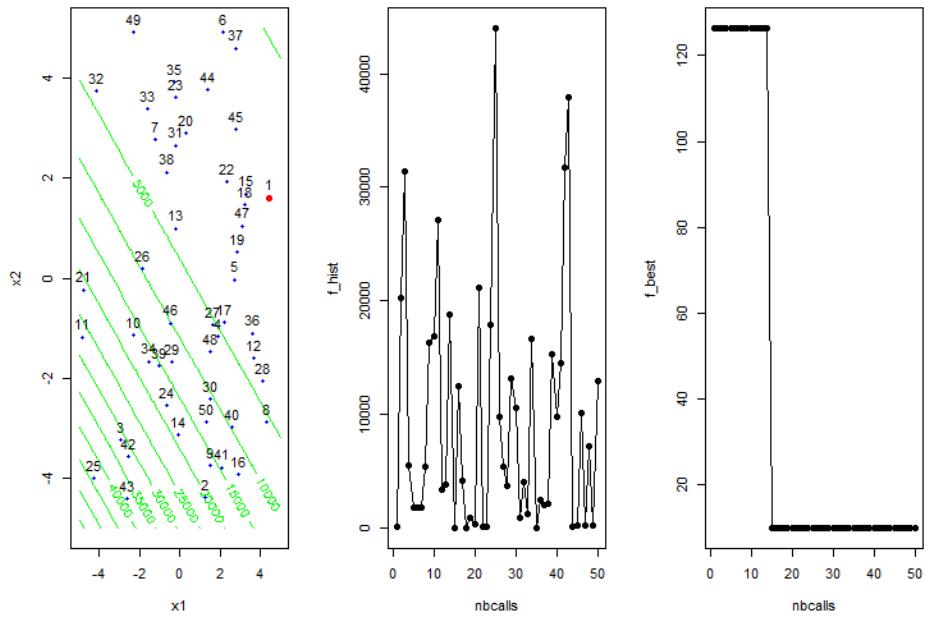
B) Optimiseur local versus optimiseur global

B.1. En 2D.

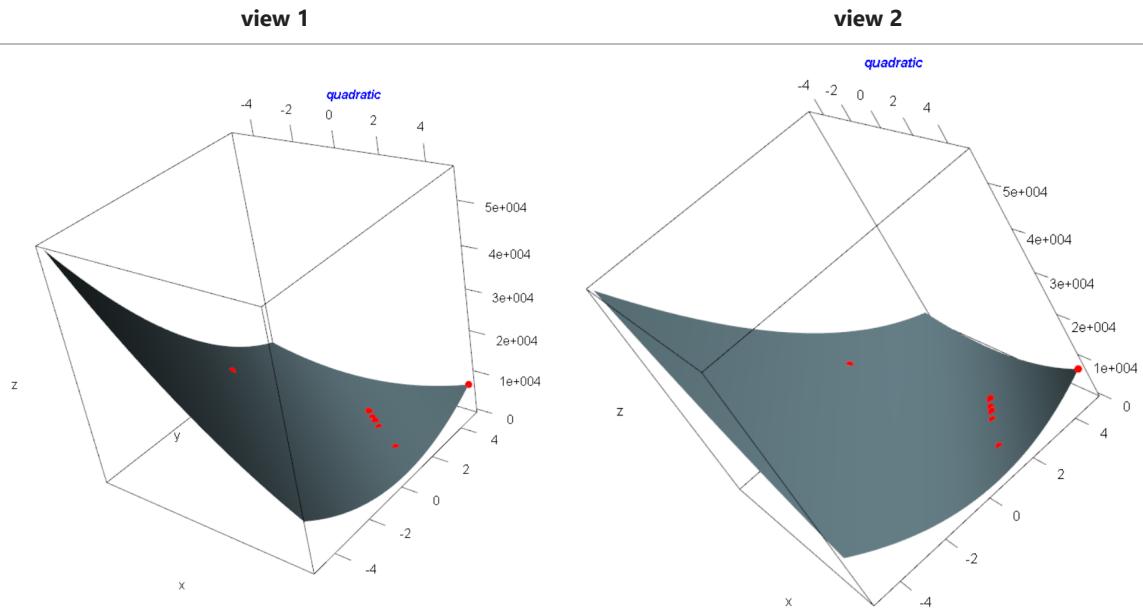
Comparaison des optimiseurs `random_search` et `lgbs` sur fonction unimodal (quadratique)

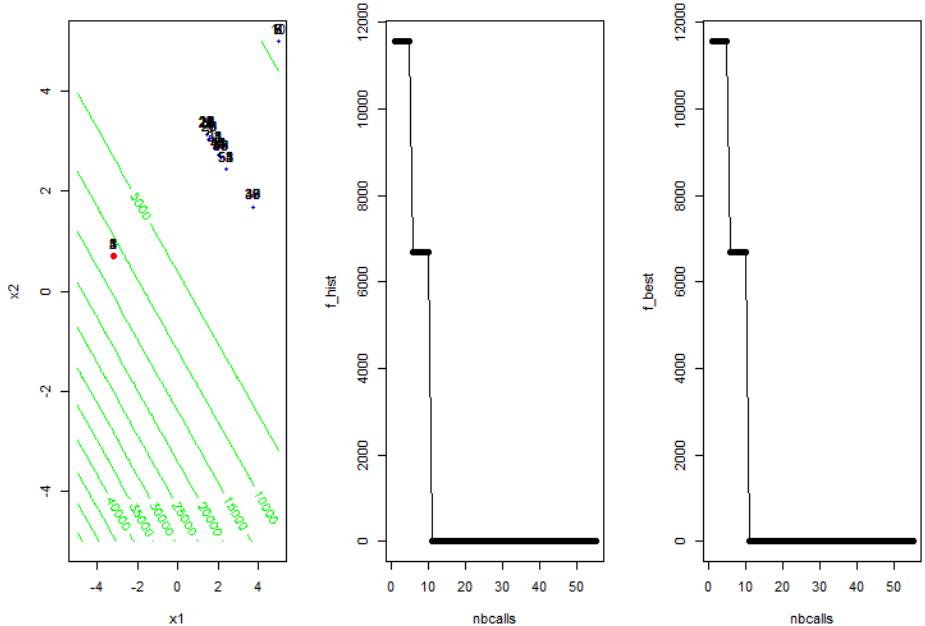
```
In [1]: #L'optimiseur global « random_search » sur la fonction unimodal « quadratic »
####parameters of the optimizer as a list.
zdim <- 2
nameoffun <- "quadratic"
nameofoptim <- "random_search"
```





```
In [3]: #L'optimiseur Local « Lbfgs » sur la fonction unimodal « quadratic »
###parameters of the optimizer as a list.
zdim <- 2
nameoffun <- "quadratic"
nameofoptim <- "lbfgs"
```



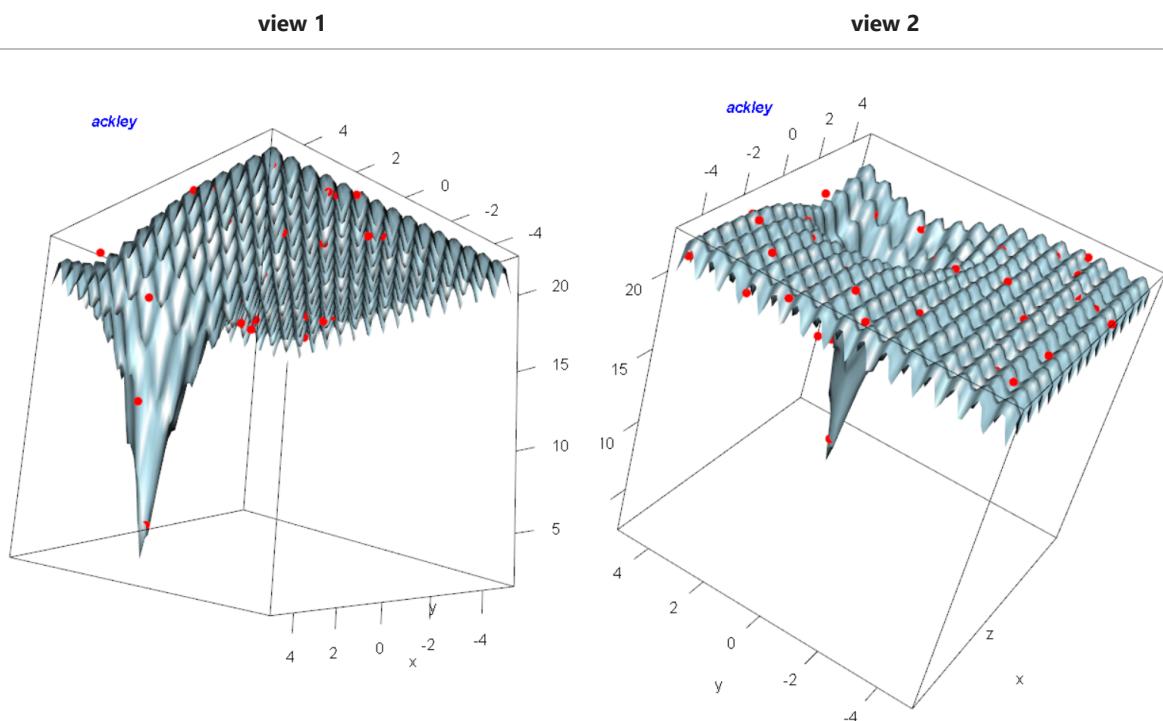


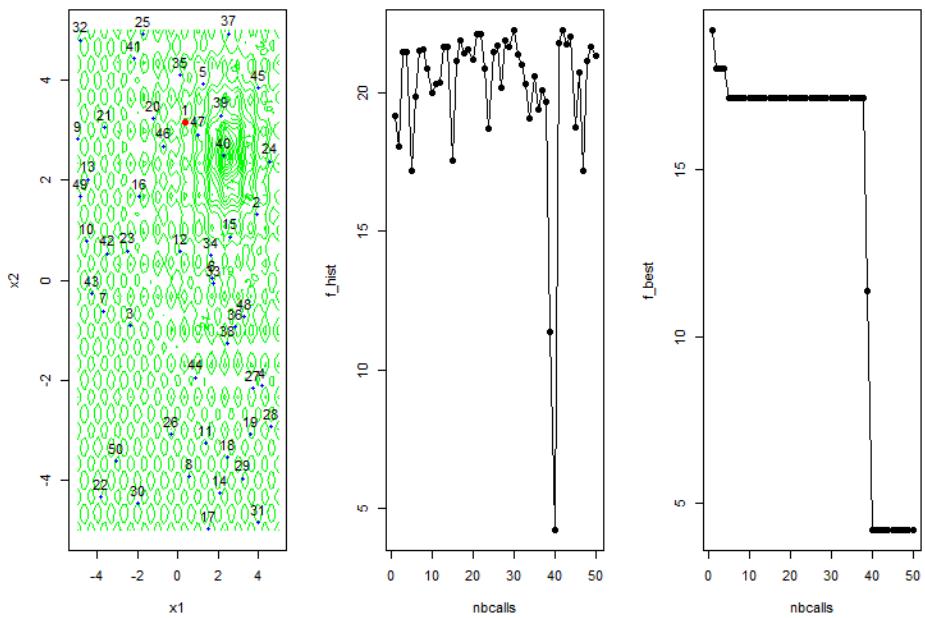
Sur une fonction unimodale comme la fonction quadratique, l'optimiseur local est plus efficace que le global. En effet, l'optimiseur lbfgs est un optimiseur local adapté aux fonctions quadratiques on a donc une convergence rapide vers le minimum global. Tandis que le caractère aléatoire (explorateur) de « random_search » fait qu'on parcourt -quelque part- « bêtement » les points sans pour autant intensifier les recherches autour d'un point où la fonction est minimale ce qui rend la convergence assez lente.

Comparaison des optimiseurs random_search et lgbs sur fonction multimodale

Ackley

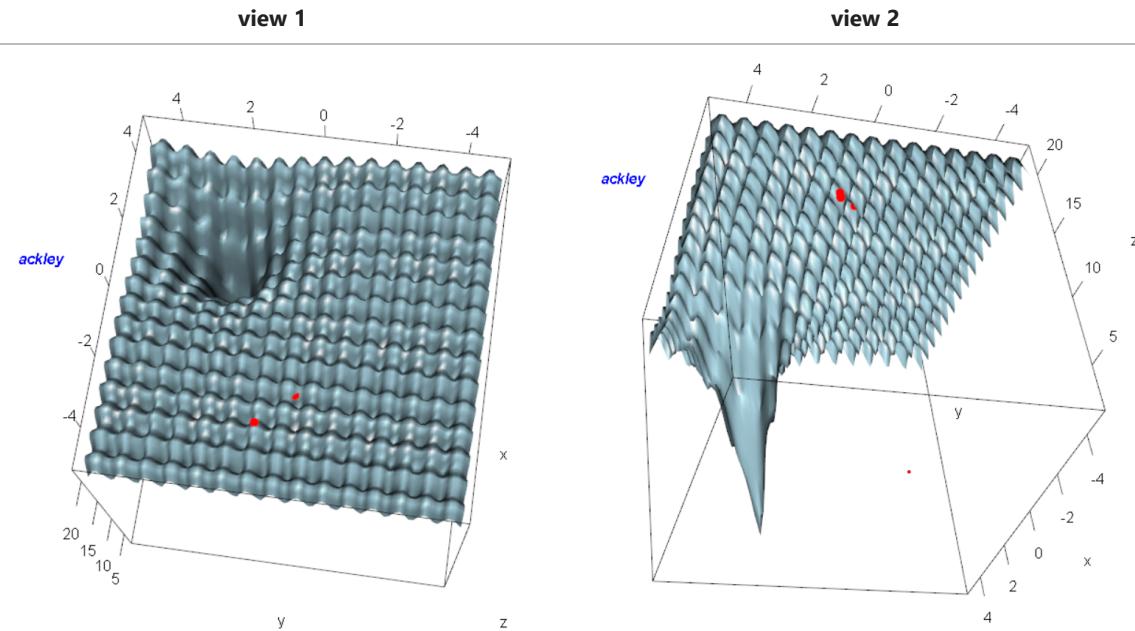
```
In [4]: #L'optimiseur global « random_search » sur la fonction multimodale « ackley »
###parameters of the optimizer as a list.
zdim <- 2
nameoffun <- "ackley"
nameofoptim <- "random_search"
```

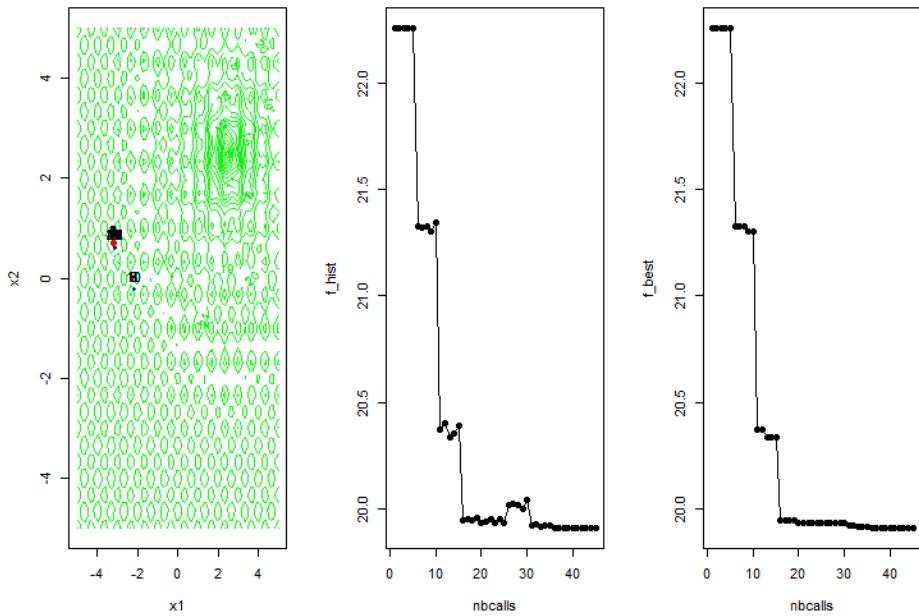




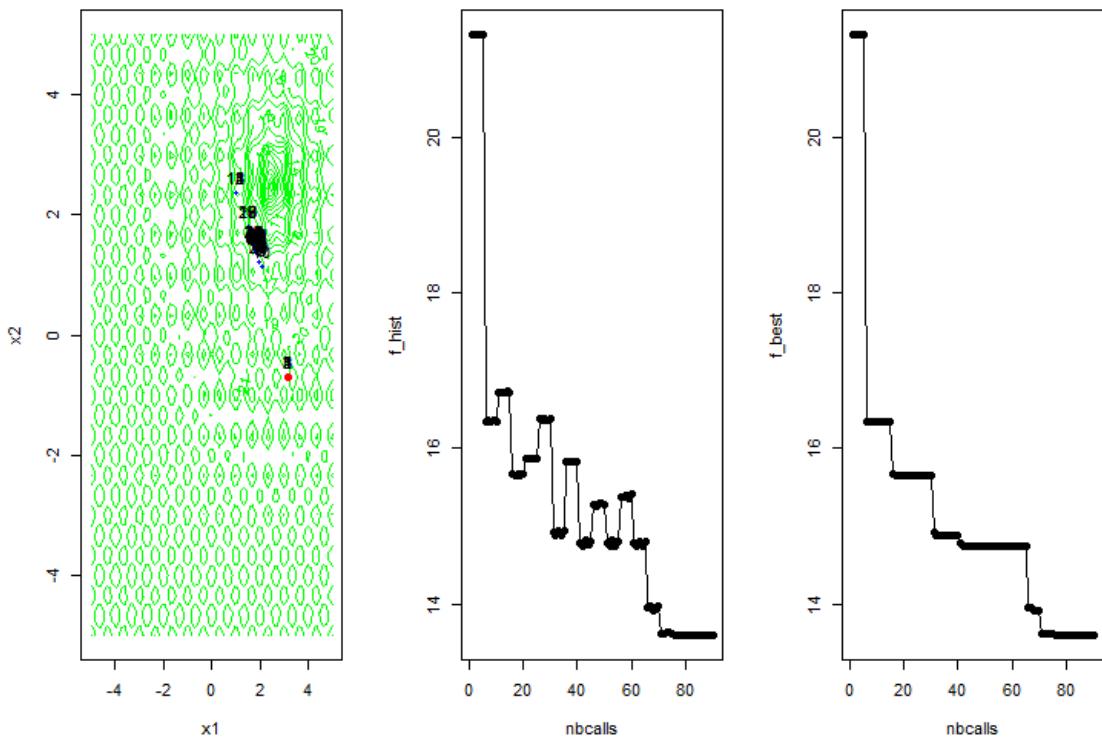
```
In [5]: #L'optimiseur Local « Lbfgs » sur la fonction multimodale « ackley »
###parameters of the optimizer as a list.
```

```
zdim <- 2
nameoffun <- "ackley"
nameofoptim <- "lbfgs"
xinit <- c(-3.2,0.7)
```





```
In [3]: xinit <- c(3.2,-0.7)
```

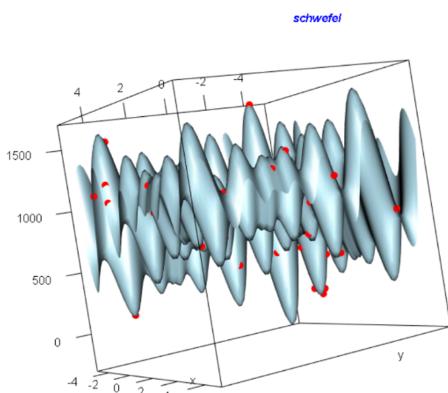


La fonction « ackley » présente plusieurs minima locaux et un minimum global prépondérant. Ici, l'optimiseur local converge vers le minimum local le plus proche du point de départ ce qui est assez naturel pour cette catégorie d'optimiseur. Le seul cas de figure où un optimiseur local, notamment « lbfsgs » converge vers le minimum global -dans le cas de fonction multimodale- est si on choisit un point de départ judicieux. Cependant il est très difficile de trouver un point qui poussera notre optimiseur vers l'optimum global de cette fonction. En effet, nous n'avons pas assez de d'information sur la fonction pour choisir un point initial proche de l'optimum global surtout en dimension supérieure à 2.

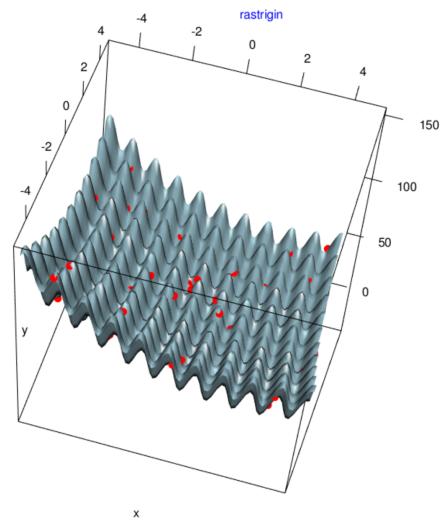
Grâce à son caractère d'explorateur aléatoire, la fonction « random_search » donne une approximation meilleure du minimum global de « ackley ».

Schwefel et Rastrigin

view 1

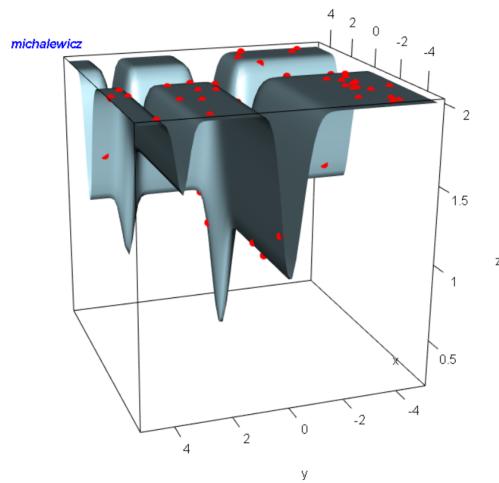


view 2



De même que pour « ackley », le caractère aléatoire de « random_search » fait que on explore plusieurs points et on approche le minimum global lentement. Mais pour des fonctions ayant plusieurs minimums locaux il est presque impossible pour un optimiseur local tel que lgbs de converger vers le minimum global.

Michalewicz



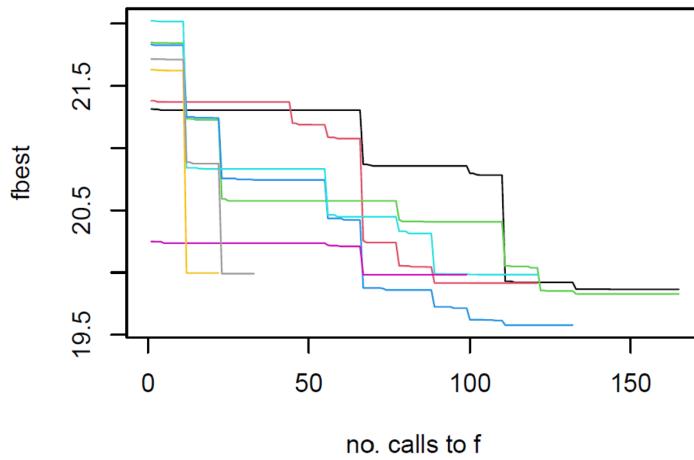
On peut dire la même chose sur cette fonction mais cette fonction a la particularité d'avoir des vallées continues qui forme un minimum global continue et un pic étroit pour le minimum global.

Les optimiseurs locaux comme lgbs convergent vers un point de la vallée la plus proche. Avec Random search comme on peut le voir sur la figure plusieurs points sont explorée dans cette recherche aléatoire de minima global. Elle est lente et n'aboutit pas toujours à cause du nombre d'itération limité.

Puisque la région de minimum global est un pic étroit et que les surfaces des régions planes sont plus grandes, la probabilité d'explorer le minimum est plus faible (cf formule volumes cours), ce qui explique le grand nombre de points rouges dans ces régions planes.

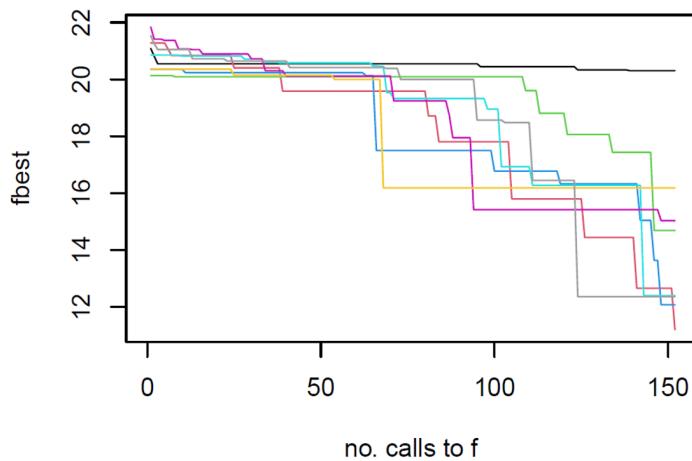
B.2 En 5D, avec répétition des tests

1- lgfs



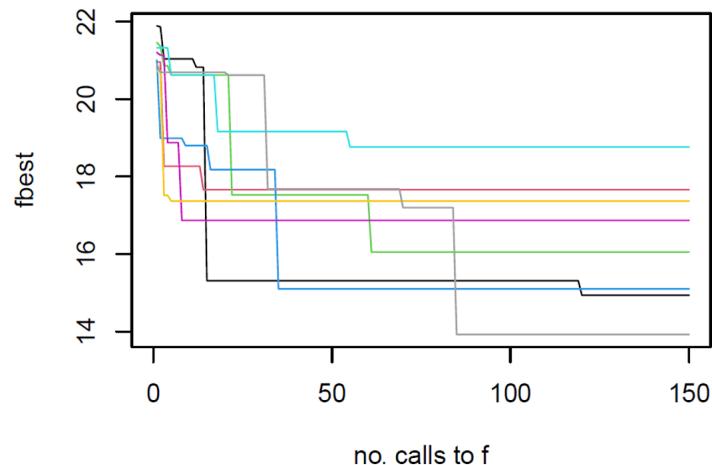
Pour « lbfgs », la vitesse de convergence vers une valeur minimale globale de f dépend du point initial et cela est normal vu qu'il s'agit d'un optimiseur local. De plus, On remarque que plusieurs test s'arrête avant les 150 itération et on peut interpréter cela par le fait que la fonction « tombe » dans un minimum local et n'arrive pas à en sortir. (Les courbes jaune et grise). La valeur minimal atteinte par cet optimiseur est légèrement supérieur à 19,5. Cet optimiseur n'est pas adapté aux fonctions multimodales.

2- cmaes



On remarque que « cmaes » prends plus de temps à converger. Pour remédier à ce problème, il faudrait faire plus de tests et avoir un budget (en termes d'appel à la fonction) plus grand, ce qui est généralement coûteux en pratique. Cette recherche d'optimum n'aboutit pas pour la fonction ackley même après 150 appels à la fonction mais on remarque qu'on approche pour quelques tests une valeur minimal aux alentours de 12 contre 19,5 pour l'optimiseur local.

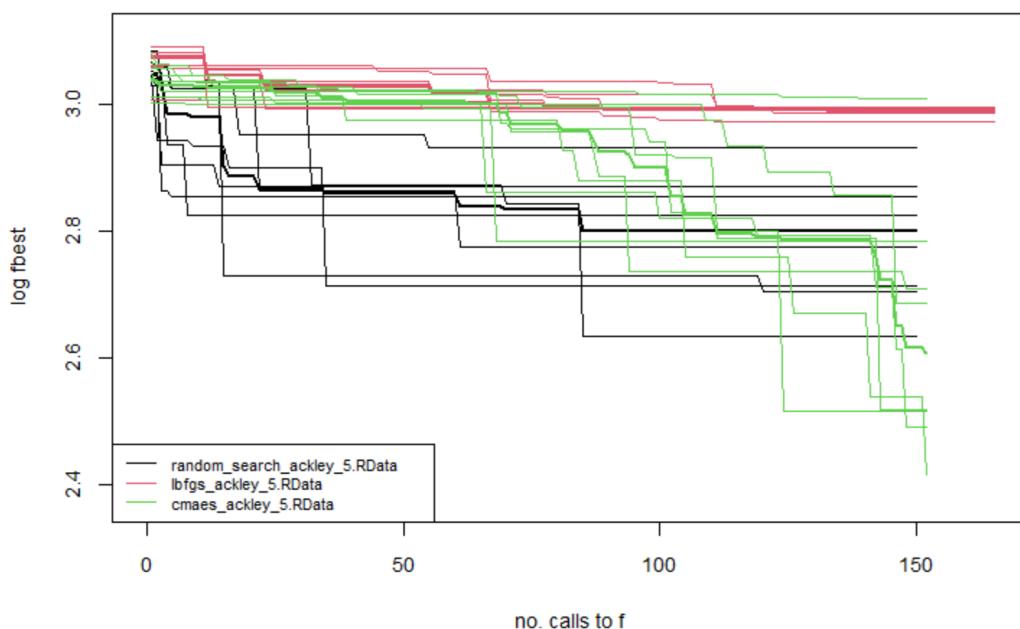
3- Random_ search



Pour random_search, on voit que l'algorithme se stabilise sur des valeurs que ne sont pas le minimum global pour plusieurs de ses tests. On retrouve la valeur 14 de l'optimiseur cmaes pour un des 8 tests.

Conclusion

Pour pouvoir comparer les 3 algorithmes nous avons utiliser le fichier postproc_tests_optimizers qui permet de representer les evolutions des 3 algorithmes avec une même echelle.



On peut dire que pour les fonctions multimodales les optimisateurs globaux font une meilleure approximation du minimum global que l'optimiseur local. On peut aller jusqu'à dire que cmaes se donne des résultats meilleurs que le random_search mais aurai besoin de plus d'appel à la fonction.

C) Effet du pas de l'algorithme stochastique ES-(1+1)

C.1. Code du Programme "normal_search.R"

```
In [ ]: # code ancien :
repeat { # sampling in-bounds
### TP1: remplacer ligne ci-dessous par La bonne commande, qui utilise par, current_par, sigma et stop("remplacer cette ligne par \"par <- ...\"")
nb_samples <- nb_samples + 1
```

```

if (all((par < UB) & (par > LB))) break()
if (nb_samples > maxnbsamp) {
  # too many attempts at satisfying bounds, issue a warning and project on bounds
  warning("too many attempts at satisfying bounds, project on bounds", immediate. = TRUE,
  iviol <- par > UB
  par[iviol] <- UB[iviol]
  iviol <- par < LB
  par[iviol] <- LB[iviol]
  break()

#completed code:
repeat { # sampling in-bounds
  par <- current_par + rnorm(n=dim, mean=0, sd=sigma)
  nb_samples <- nb_samples + 1
  if (all((par < UB) & (par > LB))) break()
  if (nb_samples > maxnbsamp) {
    # too many attempts at satisfying bounds, issue a warning and project on bounds
    warning("too many attempts at satisfying bounds, project on bounds", immediate. = TRUE,
    iviol <- par > UB
    par[iviol] <- UB[iviol]
    iviol <- par < LB
    par[iviol] <- LB[iviol]
    break()
  }
}

```

C.2. Sigma petit

In []:

```

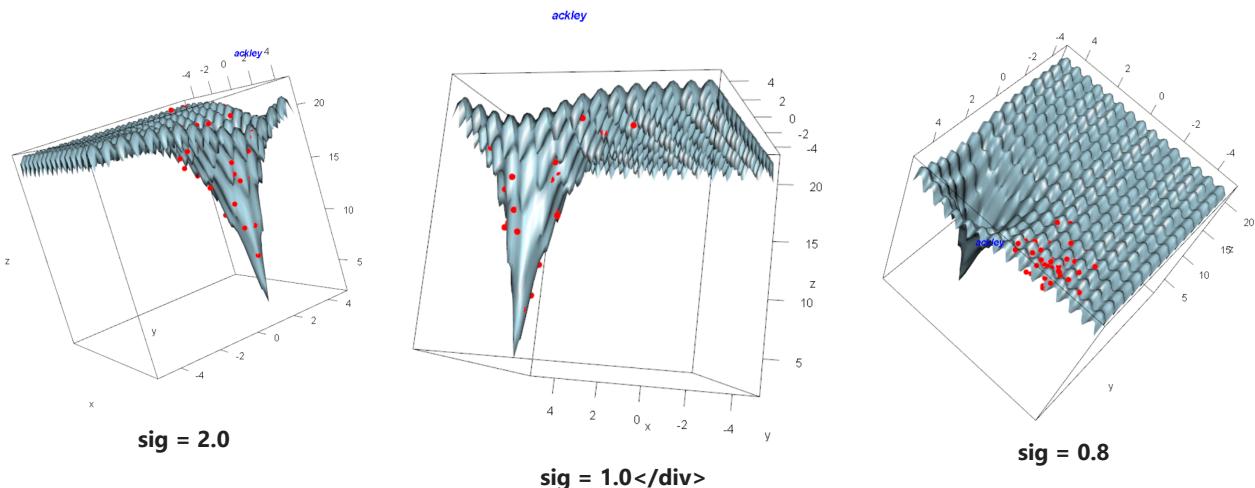
##### user data #####
nameoffun <- "ackley"
nameofoptim <- "normal_search"

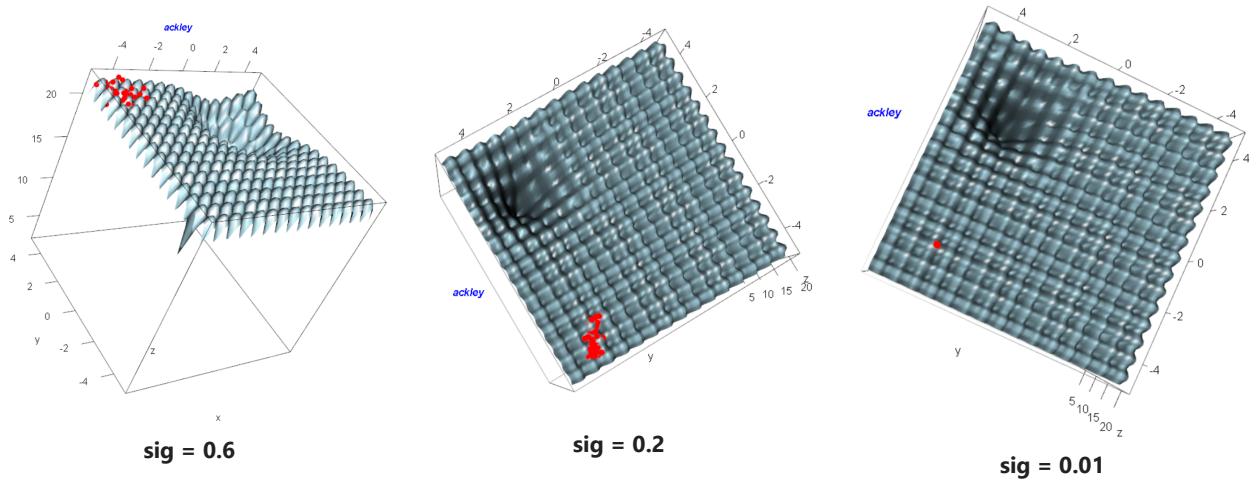
###parameters of the optimizer as a list.
zdim <- 2
budget <- 50
sig <- 2.0

...
if (nameofoptim == "normal_search") {
  param <- list(LB=-5,UB = 5,budget = budget,dim=zdim, xinit=c(-3.2,3.2),sigma=sig) # param for
}

# On va essayer sig = {2.0}, {1.0}, {0.8}, {0.6}, {0.2}, {0.01}

```



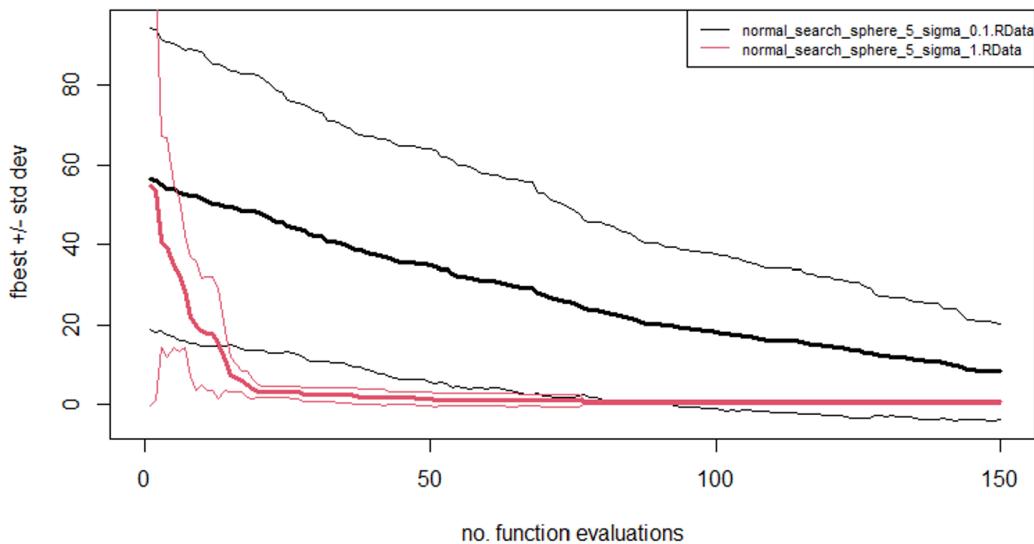


Effectivement quand sigma est faible l'optimiseur « normal_search » se comporte comme un optimiseur local et quand sigma prend de grandes valeurs il se comporte comme un optimiseur global.

C.3 l'effet de la taille du pas (sigma) de normal_search en 5D

1-Sur une fonction unimodale : sphère

De même qu'en 2D, quand la valeur de sigma diminue, l'optimiseur se comporte comme un optimiseur local et non global.

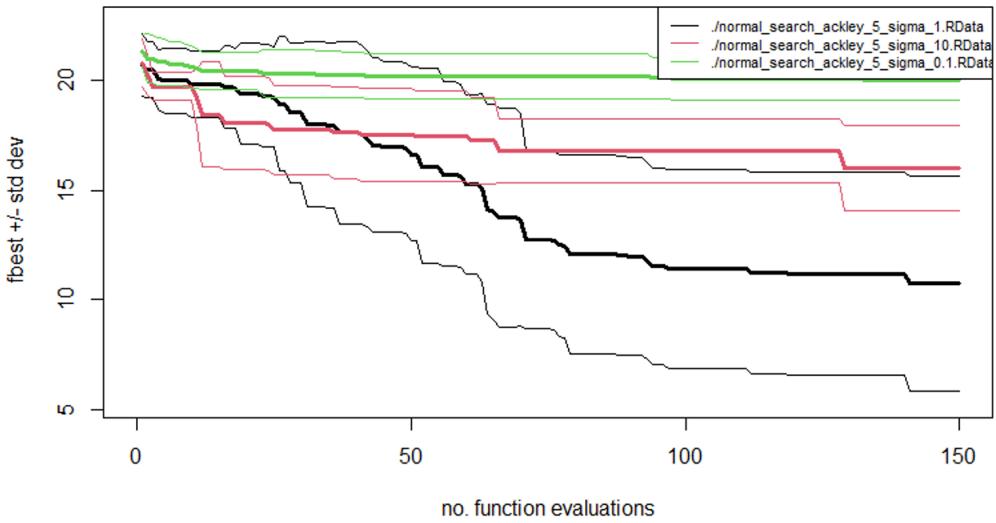


On voit bien que pour une faible valeur de sigma, l'optimiseur a du mal à converger vers la valeur minimale globale même au bout de 150 itérations alors que pour sigma = 1 l'optimiseur converge au bout d'une trentaine d'itérations.

On remarque aussi que pour une grande valeur de sigma l'incertitude autour du minimum global devient plus grande vu que l'intervalle de confiance devient plus grand.

2- Sur Une fonction multimodale : ackley

Comme nous avons pu le voir dans la partie B, pour une fonction multimodale, l'optimisateur local n'est pas adapté. On peut donc prévoir que pour des valeurs de alpha importante l'optimisateur sera meilleur car il se comporte comme un optimisateur global.



En effet, on peut voir dans la figure si dessous que pour un alpha = 1 la fonction converge plus vers une valeur plus intéressante au bout de 150 itérations par rapport aux 2 autres valeurs que l'on a prises. Si on prend alpha trop grande, l'optimiseur a du mal à converger et si on la prend trop faible l'optimiseur se comporte comme un optimiseur local.