Module 1 majeure science des données/ Cours Optimisation Classique

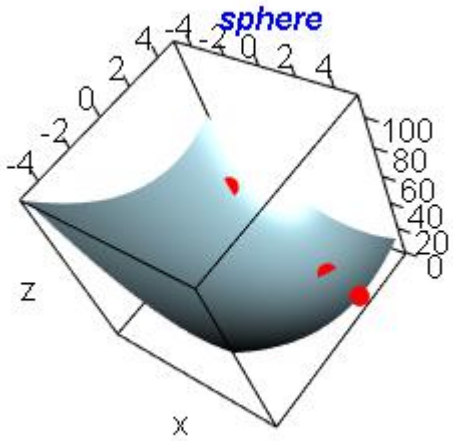
TP rapport

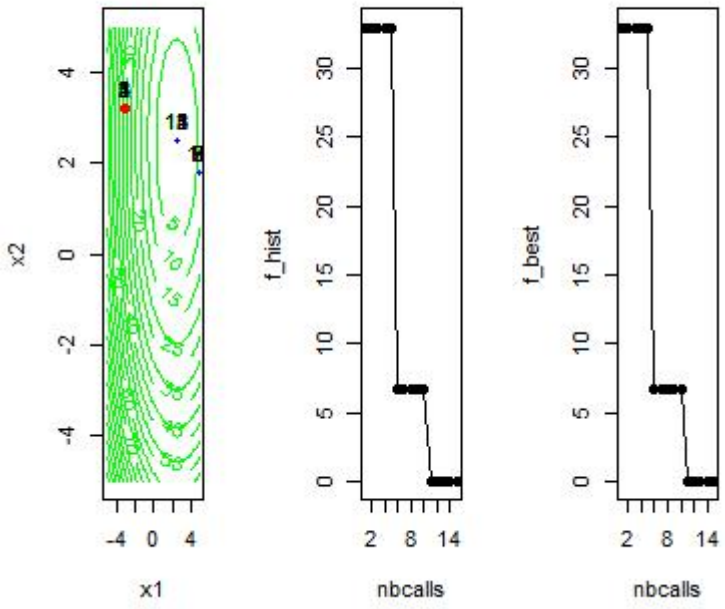
Andi WANG

Phase 1 : prise en main des programmes en R

2. Ouvrir 3Dplots.R . Ce fichier permet de visualiser des fonctions 2D et des trajectoires d'optimiseurs. Utiliser l'optimiseur local « lbfgs » sur une fonction convexe (la fonction "sphere" ou "quadratic") puis sur une fonction multimodale au choix. Faire varier le point initial. Que se passe-t-il ?

J'ai utilisé l'optimiseur<<lbfgs>>sur la fonction convexe *sphere*, je peux voir 2 images, un et 3D et on peux le tourner pour le observer de direction différente.

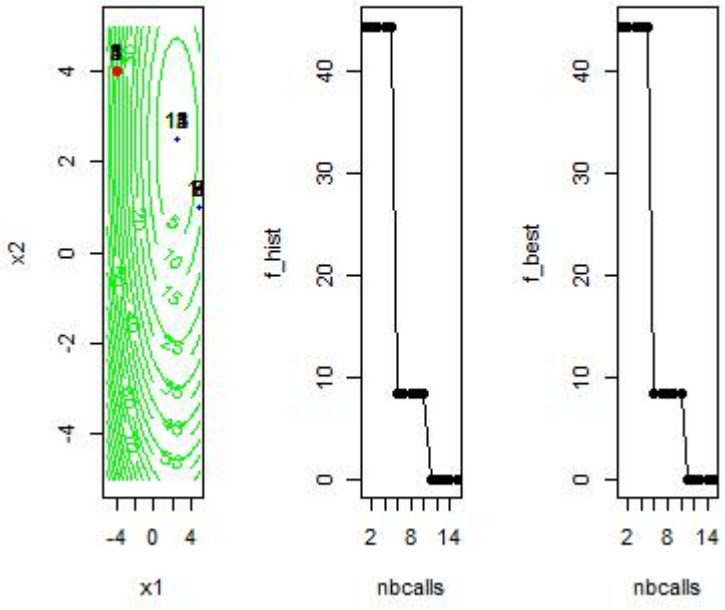




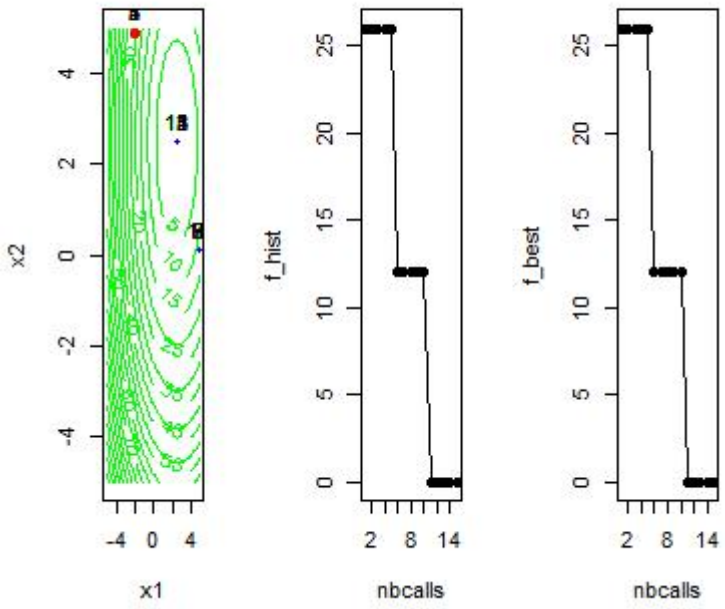
Et quand on varie le point initial :

On peut obtenir :

Quand xinit=c(-4,4) :



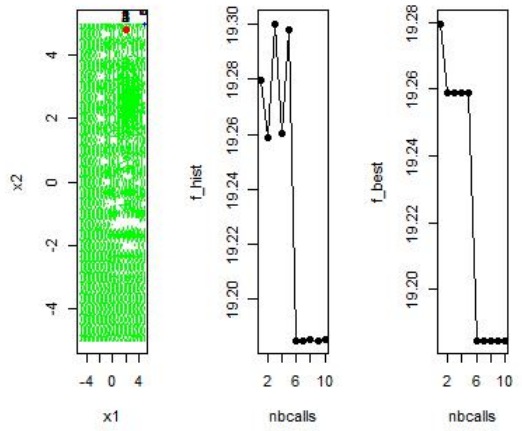
Quand xinit=c(2,4.9) :



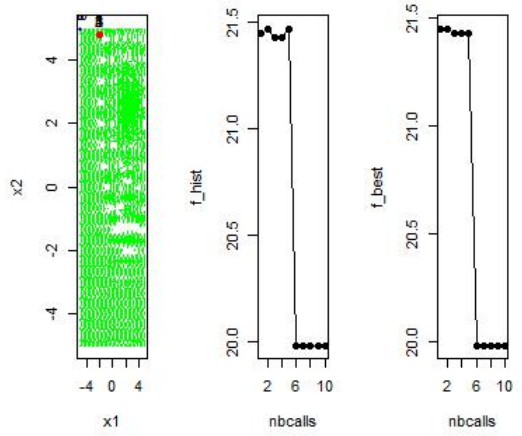
Après avoir testé avec quelques valeurs de point initial, on peut voir que les valeur de f_hist et f_best vont toujours changer.

On fait le même chose sur une fonction multimodale, on utilise la fonction *ackley* comme un test.

Quand xinit=c(2,4.8) :



Quand xinit=c(-2,4.8) :



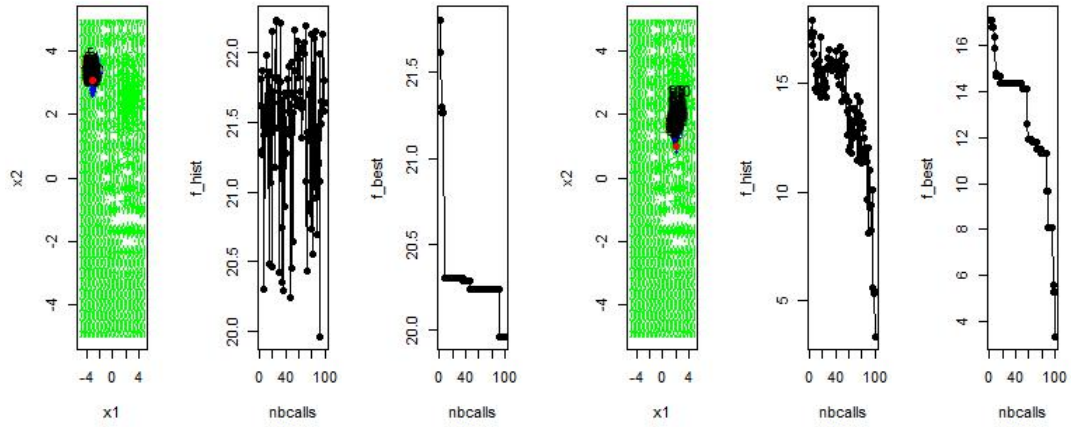
On peut conclure quand on changer le point initial, les valeurs de f_hist et f_best vont tous changer. C'est évident que cet optimiseur est optimiseur local.

3. Répéter l'expérience précédente avec un des deux optimiseurs globaux fournis (« random_search » ou « normal_search »). Noter que « normal_search » peut se comporter comme un optimiseur local ou global en fonction du choix de sigma.

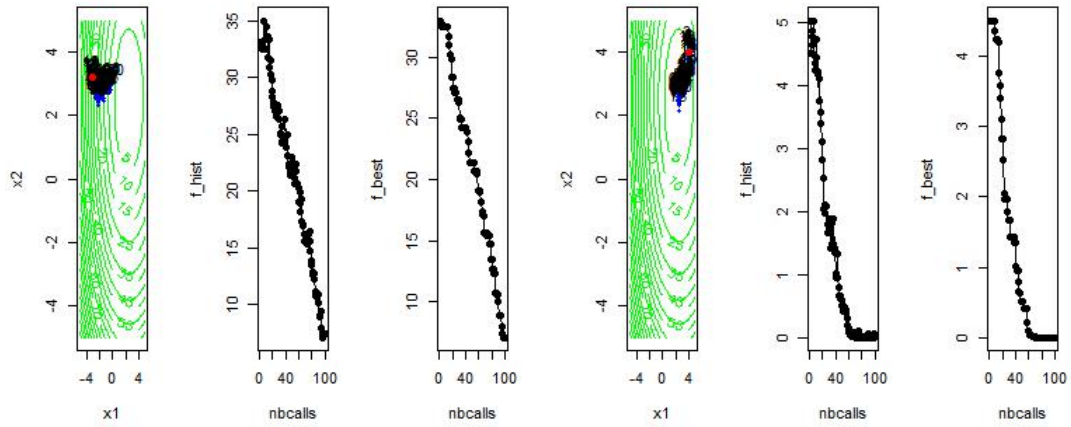
On utiliser l'optimiseur normal_search comme test :

Quand sigma=0.1 :

Avec la fonction ackley, on a testé avec quelques xinit dans $[L,U]^2$:



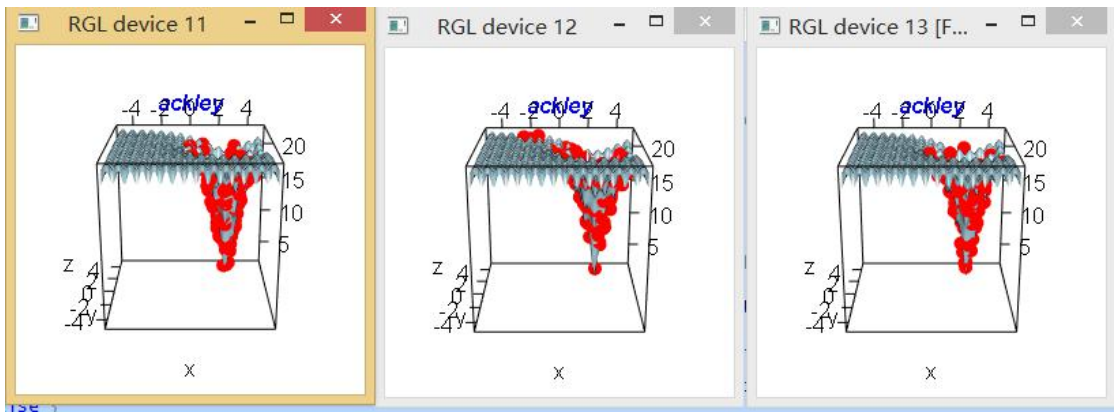
Avec la fonction sphere,on a :



Donc dans cette condition , c'est un optimiseur local.

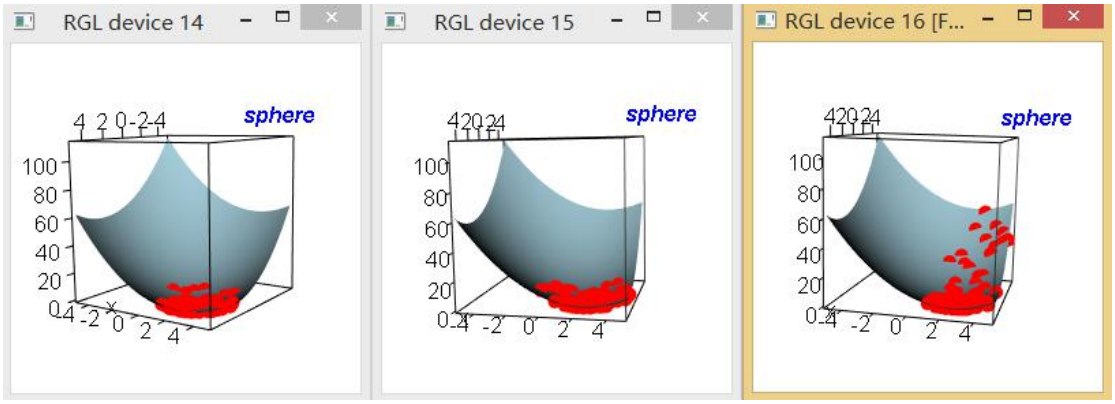
Quand sigma=1 :

Avec la fonction ackley, on a des résultats :



C'est presque un optimiseur global pour la fonction ackley

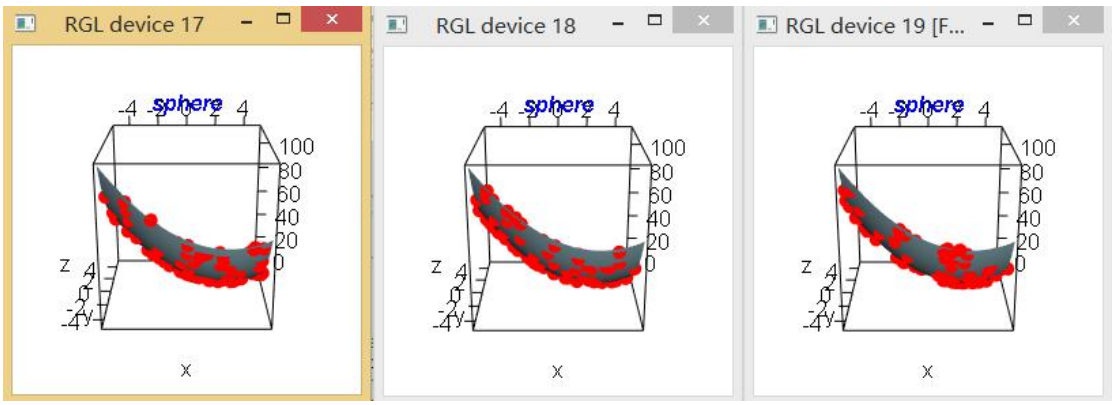
Avec la fonction sphere, on a des résultat :



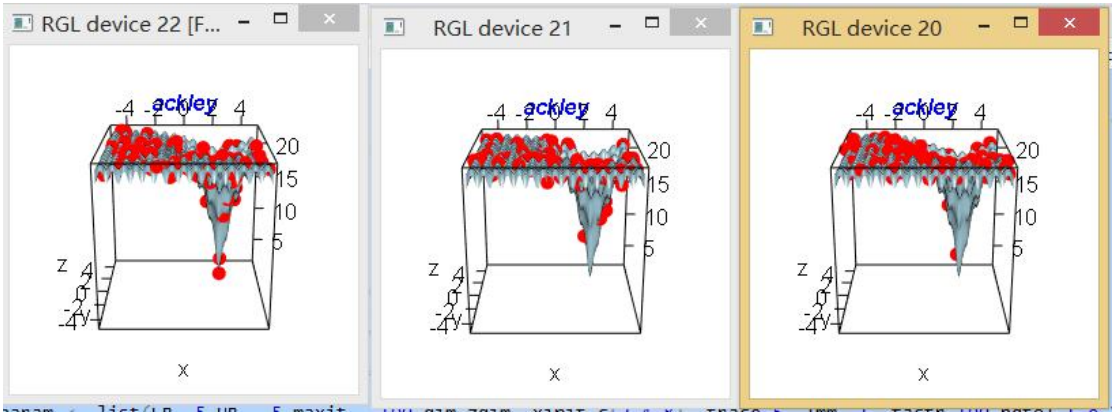
Dans cette condition, c'est un optimiseur local .

Quand sigma=10 :

Avec la fonction sphere, on a des résultat :

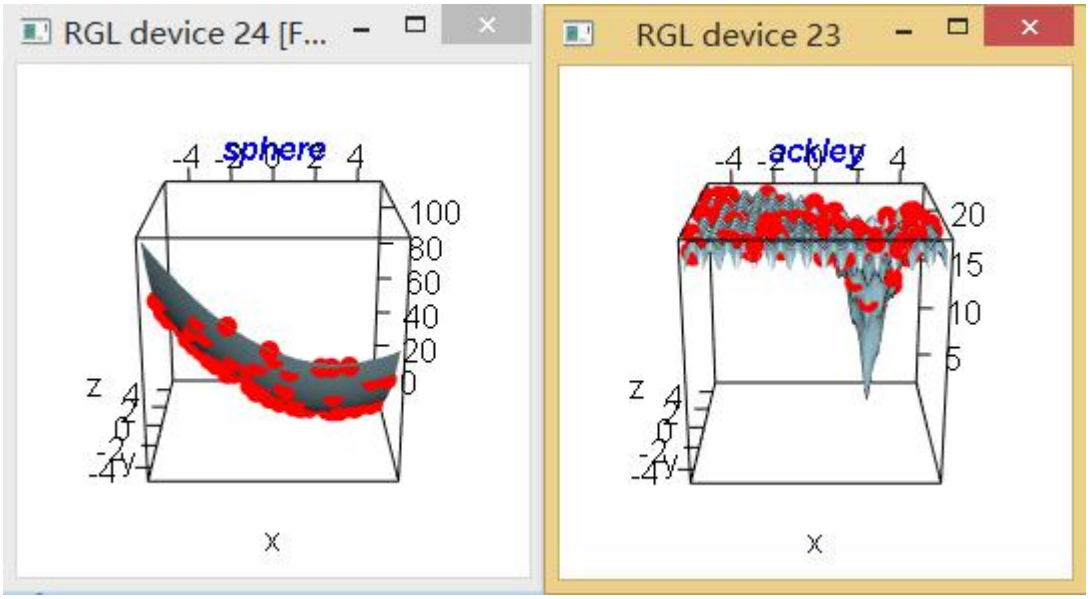


Avec la fonction ackley, on a des résultat :



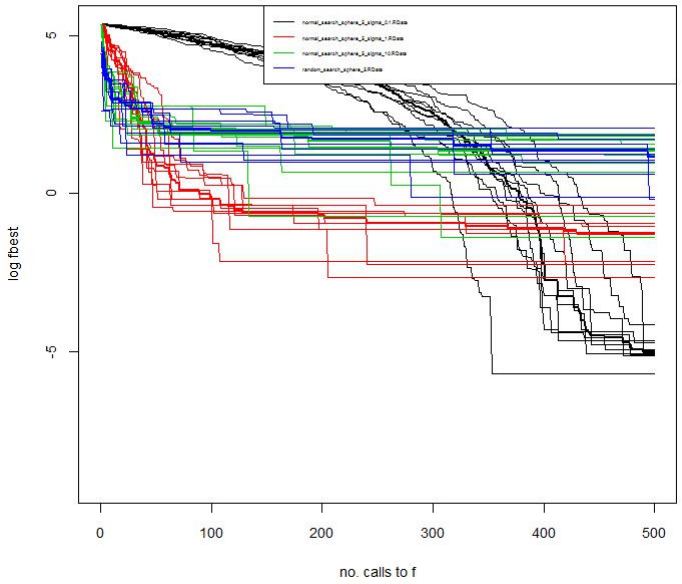
On peut conclure quand on augmente la valeur de sigma, on peut obtenir un optimiseur de plus en plus global.

Pour l'optimiseur random_search, le point initial n'est pas un paramètre pour cet optimiseur, c'est évident optimiseur global. Avec les deux fonction, on a des résultats :



4. Ouvrir « main4tests.R ». Nous allons maintenant faire des expériences numériques en répétant des optimisations. Choisir comme optimiseur random_search puis normal_search, et faire des tests sur la fonction « sphere » en faisant varier la taille du pas sigma (cas de normal_search). Utiliser le fichier « postproc_tests_optimizers.R » pour faire des courbes synthétiques. Discuter les résultats.

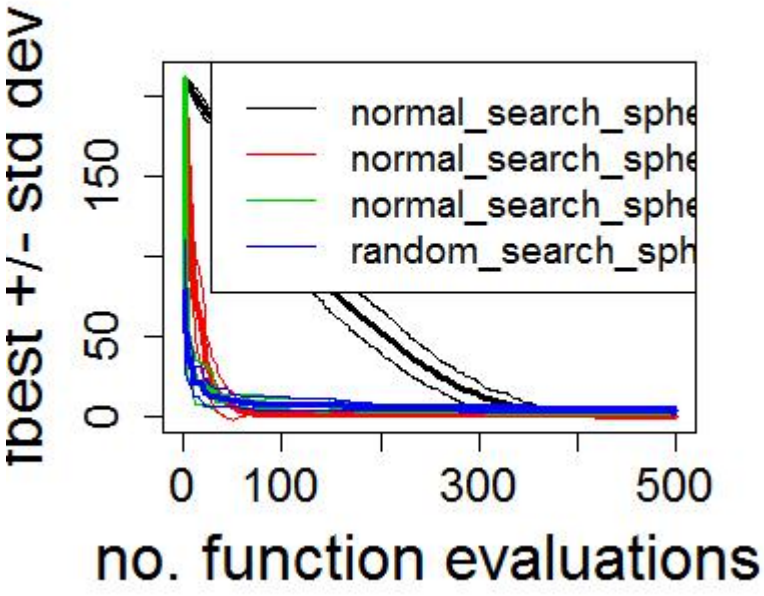
On teste avec sigma=0.1, 1 et 10 et l'optimiseur random, et on obtient 2 figures :



Dans la figure 1, on peut voir que le résu ltat de random_search sont le plus concentré, et normal_search en sigma=0.1 est le plus diffusif. A mon avis , comme ils ont des performance différente pour faire une optimisation globale, random_search est la meilleur et normal_search en sigma=0.1 est pire. Et quand on augmente sigma, c'est de plus en plus

facile à faire une optimisation globale. On peut conclure d'après cette figure : pour l'optimisation globale, l'efficacité est :

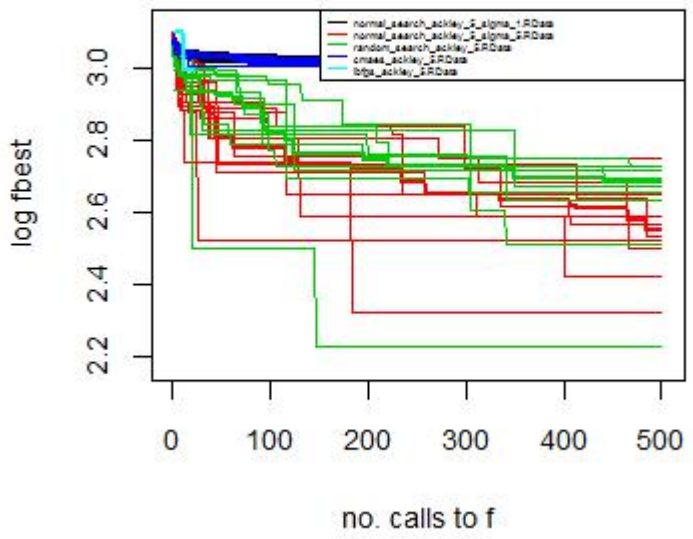
random_search> normal_search en sigma=10, normal_search en sigma=1 > normal_search en sigma=0.1



On peut voir que les 3 lignes normal_search_sphere_1, normal_search_sphere_10 et random_search_sphere_5 sont les plus similaire dans la figure 2. Et tous les lignes vont converger à 0 quand no.function evaluations est très grand.

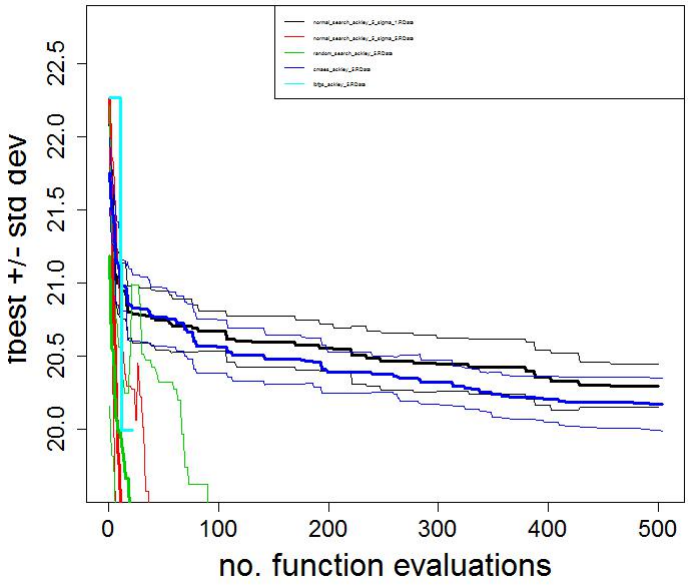
5. Comparaison d'optimiseurs locaux et globaux sur une fonction multimodale. Choisir une fonction multimodale en dimensions 5 et comparer les performances des optimiseurs fournis (lbfgs, normal_search, random_search, cmaes). Pour lbfgs, cmaes et normal_search, prendre le même point initial. lbfgs étant un optimiseur déterministe, un seul test est suffisant.

on va tous utiliser xinit=rep(-4,zdim) et la fonction ackley pour les tests :



On peut voir d'après cette figure l'optimiseur cmaes est le plus concentré dans ces 5 optimiseur. lbgfs est le plus court comme c'est un un optimiseur déterministe. Donc on peut ranger les autres 4 optimiseur d'après cette figure de leur efficace pour optimisation globale

cmaes> normal_search en sigma=1> random>normal_search en sigma=5



Seulement les lignes de cmaes et celles de normal en sigma=1 sont convergentes sur cette figure, ils sont aussi les meilleurs optimiseur pour faire une optimisation globale en utilisant la function ackley.

Phase 2: étude d'un problème, le plan d'expériences optimal

1. Programmer une nouvelle fonction objectif (comme celles dans "test_functions.R")

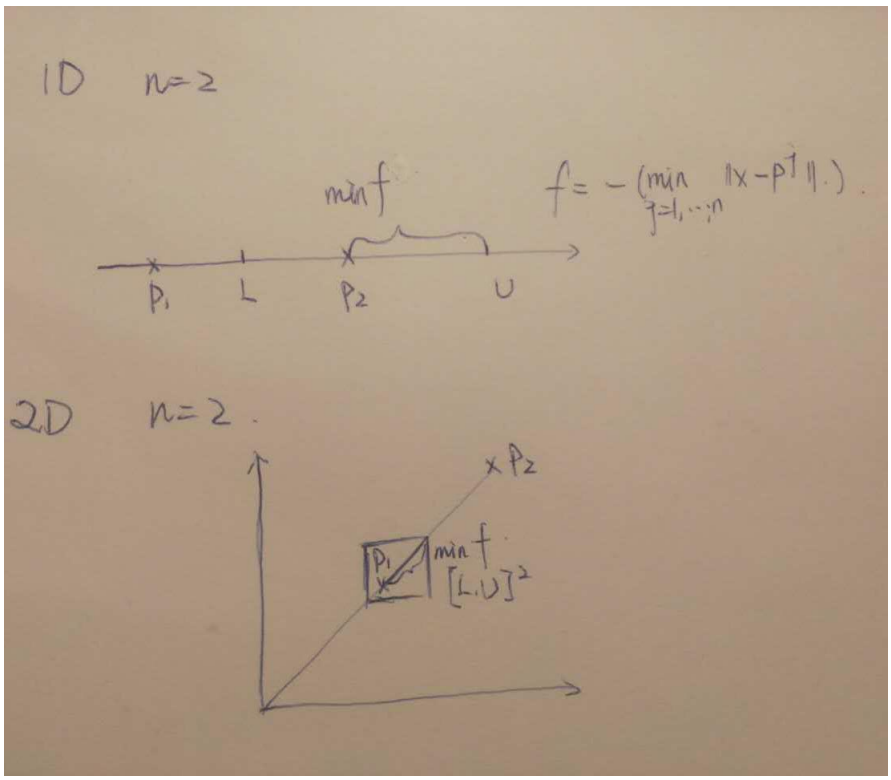
correspondant au problème suivant: trouver un points x qui soient le plus éloigné possible de n points pj déjà fixés. C'est un problème classique, difficile et important de planification d'une nouvelle expérience quand n expériences ont déjà été réalisées et chaque expériences est décrite par d paramètres.

Mathématiquement, le problème est formulé comme

$$\boldsymbol{min}_{x\in [\mathbf{L},\mathbf{U}]^{\mathbf{d}} \sqsubset \mathbf{R}^{\mathbf{d}}} - (\boldsymbol{min}_{j=1,\ldots,n}\|\boldsymbol{x}-\boldsymbol{p}^{j}\|)$$

$\mathbf{p}^{\mathbf{j}} \in \mathbf{R}^{\mathbf{d}}$, j=1,..,n sont des points du plan d'expériences donnés et L, U sont les bornes sur les variables.

Pour se faire une intuition, on considérera d'abord les cas d=1 et 2 et on fera des dessins.



2. Etudier quels optimiseurs semblent bien ou mal fonctionner sur ce problème en fonction de d (la dimension) et n (le nombre de points déjà connus).

Pour calculer $-(\boldsymbol{min}_{j=1,\ldots,n}\|\boldsymbol{x}-\boldsymbol{p}^{j}\|)$ sur chaque x, on utilise des codes :

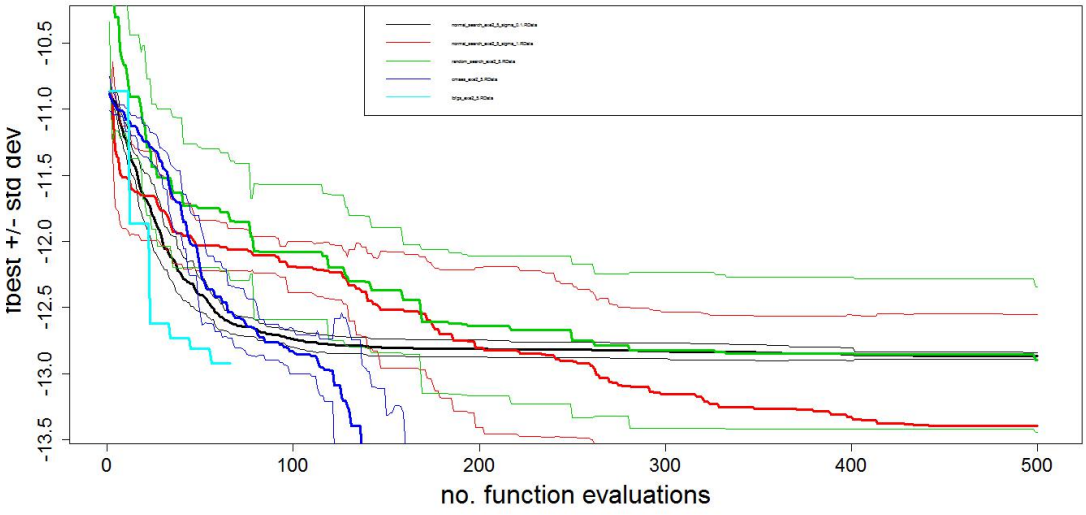
```
zz<-matrix(0,n,1)
```

```
for(i in 1:n){
  zz<-sqrt(sum(t(xx-pp[i,])*(xx-pp[i,])))
}
y<- -min(zz)
return(y+noise)
```

et puis on utilise des optimiseur pour trouver le min y et xx à ce moment.

On suppose n=5 et d=5.

On utilise normal avec sigma=0.1 et 1, random, cmaes et lbfgs, et on obtient des figures :



On peux voir que les optimiseur <<lbfgs>> est mal fonctionner sur ce problème. <<cmaes>> est moyen pour ce probème. Et les autres trois optimiseur peuvent bien fonctionner sur ce problème.

3. (Bonus) modifier soit l'optimiseur "normal_search" (dans l'esprit CMA-ES ou en ajoutant des recherches locales) soit "lbfgs" (en ajoutant des redémarrages) pour essayer d'améliorer l'efficacité des optimiseurs. L'évaluation sera plus sur les idées proposées que la performance constatée (car vous avez peu de temps).

```
Initialisations : x, f(x), m, C, tmax

Tant que t < tmax

 faire,

        Instancier N(m,C) --> x'

        Calculer f(x') , t = t+1

        Si f(x')<f(x), x = x' ,

                f(x) = f(x')

        Fin si

        Mettre à jour m (e.g., m=x) et C

Fin tant que
```

C'est les algorithmes de ES-(1+ 1), si on veut améliorer l'efficace de l'optimiseur normal_search, on peut utiliser cette méthode.

```
Initialisations : x, f(x), m, C, tmax

Tant que t < tmax

 faire,

        Instancier N(m,C) --> x'

        xinit<-x'

        Calculer min f avec l'optimiseur normal_search en xinit,et obtenir l'optimiseur local f(x'),

         t = t+1

        Si f(x')<f(x),

                x = x' ,

                f(x) = f(x')

        Fin si

        Mettre à jour m (e.g., m=x) et C

Fin tant que
```

on remplace f(x') dans ES-(1+1) avec le résultat obtenu par l'optimiseur normal_search on fait une circulation, et chaque fois quand on obtient un optimisation local plus petit, on remplace f(x) et x avec des nouvelles valeurs qu'on a juste calculé.