TP de l'UP4 : Exploitation mathématique de simulateurs numériques

Majeure Data Science

Réalisé par : ABDELMOULA Mohamed Taha -AKRIM Anass

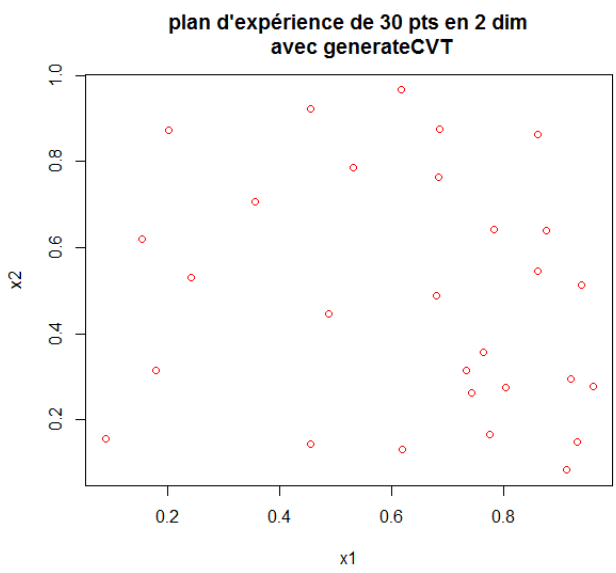
Partie I : plans remplissant l'espace et krigeage

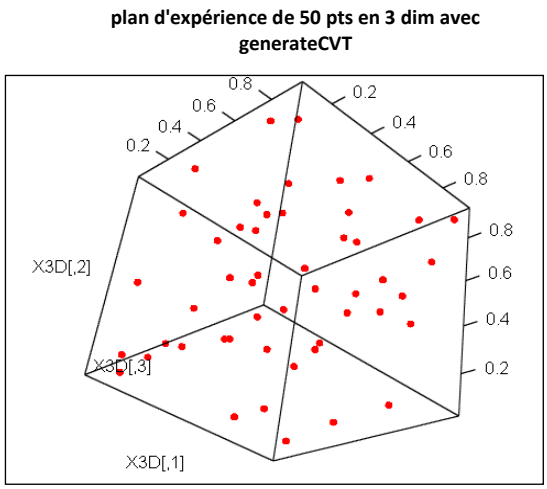
I) Construction de plans

a) Tesselations centroïdales de Voronoï

On construit notre plan d'expérience, en utilisant l'algorithme de McQueen, et voici un aperçu de l'algorithme implémenté sur R :

On obtient ainsi, les résultats ci-dessous :





On voit bien que les points générés sont bien répartis et remplissent le plan (2D et 3D)

b) Hypercubes latins

Script R générant un hypercube latin aléatoire normalisé, avec 'npts' points et de dimension 'dim' :

```
generateLHS <- function(npts, dim){
  temp<-replicate(dim, runif(npts,0,1))
  return(temp)}
```

c) Critères

Script R pour calculer le critère maximin d'un plan, avec « allDist » la matrice des interdistances entre points, et « minDist » vecteur correspondant à la plus petite distance pour chaque point Xi :

```
evalMinDist <- function(X){
  D<-as.matrix(dist(X, method='euclidean'))
  temp<-matrix(NA, nrow = nrow(X), ncol = 1)
  for(i in 1:nrow(X)) {
    s<-as.vector(D[i,-i]) #retire le pt
    ind<-which.min(s)
    temp[i,]<-s[ind] }
  d<-temp
  return( list(minDist = d, allDist = D) )
}
```

d) Hypercubes latins optimisés

Script R pour construire un Hypercube latin optimisé par échange de recuit simulé (ceci pour un seul critère : critère maximin):

```
optimiseLHS<-function(lhs,nite,T0){
  for(i in 1:nite){
    Temp<-T0/(log(i))
    rnd<-runif(1, min=0, max=1)
    lhs.temp<-lhs
    tmp<-evalMinDist(lhs)$minDist
    ind.ptCritique<-which.min(tmp)
    distActuelle<-tmp[ind.ptCritique]
    rnd.col<-round(runif(1, min=1, max=ncol(lhs))) #numero de colonne du pt critique aleatoire
    repeat{#on repete jusqua ce que l'indice du pt aleatoire soit different à l'indice du pt critique
      rnd.pt<-round(runif(1, min=1, max=nrow(lhs)))
      if(rnd.pt!=ind.ptCritique){break}
    }

    #on cree des variables temporaires (ici les points apres echange)
    temp1<-lhs[ind.ptCritique,]
    temp1[rnd.col]<-lhs[rnd.pt,rnd.col]

    temp2<-lhs[rnd.pt,]
    temp2[rnd.col]<-lhs[ind.ptCritique,rnd.col]

    #on cree une matrice temporaire (ici apres substitution des coordonnees de deux points)
    lhs.temp[ind.ptCritique,]<-temp1
    lhs.temp[rnd.pt,]<-temp2

    #on recupere la nouvelle distance minimum de l'ancien point critique
    distNouvelle<-evalMinDist(lhs.temp)$minDist[ind.ptCritique]
    if(distNouvelle>distActuelle) {lhs<-lhs.temp
    distActuelle<-distNouvelle}

    else if(rnd<=exp(-(distActuelle-distNouvelle)/Temp)){
    lhs<-lhs.temp
    distActuelle<-distNouvelle}
  }
  return(lhs)}
```

Ici avec une température initiale de 10° et par défaut 5000 itérations. On aurait pu réaliser un algorithme de recuit simulé pour Hypercubes latins optimisés plus performant en rajoutant des critères au problème (discrepancy, phiP…), et nous nous retrouverons face à un problème multicritère.

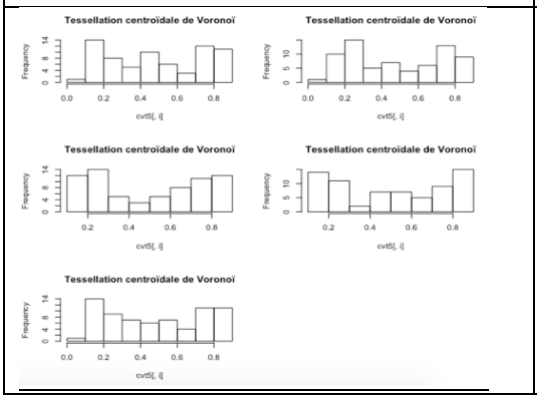
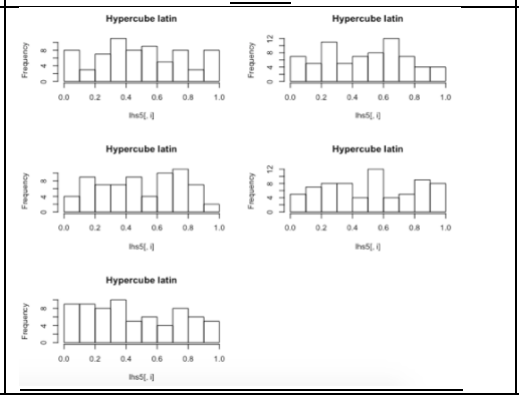
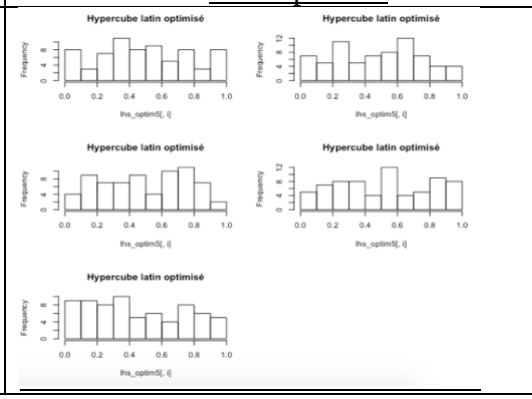
De ce fait, nous utiliserons la fonction « maximinSA_LHS » récupérée dans le package « DiceDesign », celle-ci prenant en compte plusieurs critères, et nous permettant d'obtenir un meilleur plan d'expérience.

II) Analyse

Nous nous intéresserons dans cette partie à générer trois plans d'expérience et à les comparer en fonction de plusieurs critères : critères calculatoires et visualisation graphique.

Nous nous intéressons plus particulièrement ici aux plans de dimension 5 (soit en rapport avec le cas test volcan traité par la suite) et de 70 points.

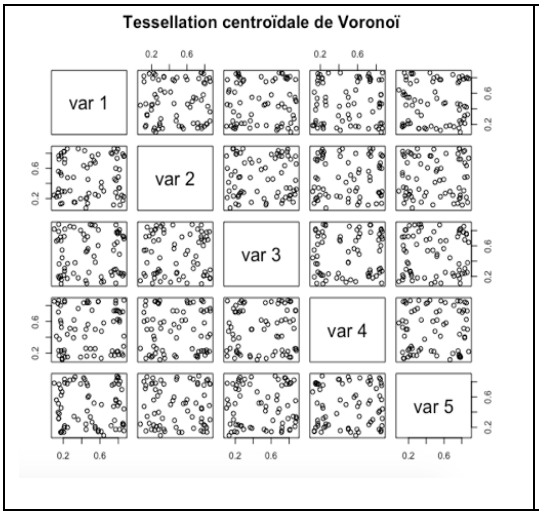
a) Visualisation graphique

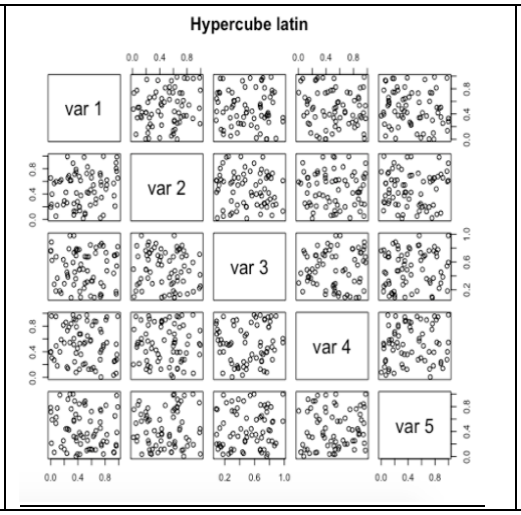
TC de Voronoï	LHS	LHS optimisé
		

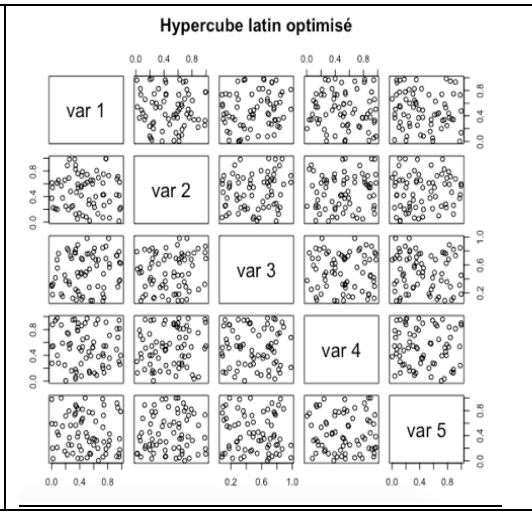
Nous cherchons un plan réalisant un bon compromis entre le nombre d'expérience (soit le nombre de points) et l'exploration du domaine, soit une bonne répartition spatiale des points.

Les histogrammes correspondent à la distribution des points sur chaque dimension. On Nous cherchons un plan d'expérience dans lequel nos points sont répartis le plus uniformément possible sur l'ensemble de l'intervalle [0,1] de chaque dimension, et donc on cherche à avoir un plan d'expérience pour lequel chaque histogramme comporte des barres de longueur équivalente (et non des inégalités de longueur trop importantes).
On remarque que les points Xi des hybercubes latins (aléatoire et optimisé) sont mieux répartis sur l'ensemble de l'intervalle [0,1] que le centroïdale de Voroinoï (répartition 'quasi-uniforme' pour les hypercubes latins, car par exemple pour le centroïdale, nous avons en dimension 1, 2 et 5, environ 15 points dans l'intervalle [0.1,0.2] contre 1 à 2 points dans [0,0.1]) sont très peu représentés dans l'intervalle [0,0.1], alors que toutes les dimensions des hypercubes latins sont quasiment toutes remplies).

Notre hypercube latin aléatoire et l'hypercube optimisé ont forcément les mêmes histogrammes car par recuit simulé, nous avons gardé les valeurs de l'hypercube initial dans chaque dimension : nous avons seulement déplacé nos points en permutant des coordonnées entre points.



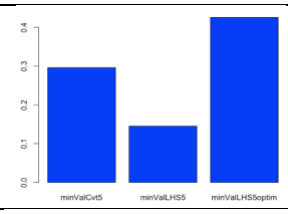
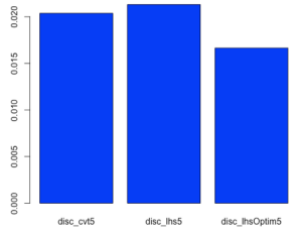
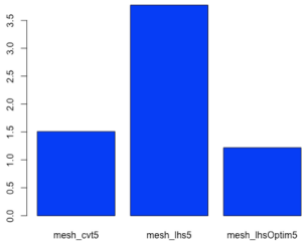




Ensuite en ce qui concerne la répartition sur les marginales de dimension 2, nous cherchons un plans d'expérience dont les points arrivent à couvrir le maximum de l'espace disponible. On remarque que le TC de Voroinoï présente des concentrations de points sur certains endroits, mais aussi des espaces vides, tandis que l'hypercube latin optimisé présente des plans contenant des points plus espacés et couvrant la quasi-totalité de l'espace.

Ainsi la visualisation graphique nous pousse à choisir l'hypercube latin optimisé.
Nous allons vérifier ceci à l'aide de tests plus rigoureux.

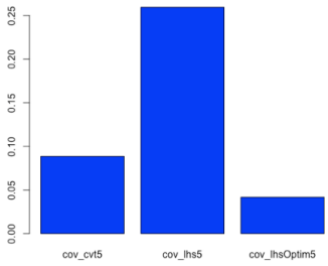
b) Critères

Maximin		
Discrepancy		
Mesh ratio		

```
          minValCvt5 minValLHS5 minValLHS5optim
maximin  0.2958539  0.1451904       0.4254704
```

```
             disc_cvt5  disc_lhs5 disc_lhsOptim5
discrepancy 0.02036963 0.02130808     0.01664821
```

```
           mesh_cvt5 mesh_lhs5 mesh_lhsOptim5
mesh ratio  1.509938  3.775261       1.220356
```

Coverage	cov_cvt5 cov_lhs5 cov_lhsOptim5 coverage 0.08859475 0.2594628 0.04178012	

Nous avons ensuite eu recours à d'autres critères plus formels (dont des critères importés du package DiceDesign) :

1-Le critère maximin :

Nous remarquons que pour l'hypercube latin optimisé, le plus petit écart entre les points du plan est de 0.37, qui est plus grand que celui des autres plans d'expérience. Ce critère nous indique donc que les points de l'hypercube latin optimisé sont plus espacés que ceux des autres plans d'expérience.

2-Le critère discrepancy :

Le critère « discrepancy » est une mesure de non uniformité du plan, et nous cherchons donc à avoir un plan dont la valeur de ce critère est la plus proche de 0. On voit bien que celle de l'hypercube latin optimisé est la plus petite (0.017) et donc la proche de 0.

4-La mesure meshRatio :

Le meshRatio est la valeur du rapport entre la distance maximale et la distance minimale entre deux points du plan exploratoire. Ainsi on a un rapport proche de 1 lorsque nos points sont étalés et répartis uniformément dans le plan.
On remarque que le meshRatio de l'hypercube latin optimisé est le plus proche de 1 (égal à 1.22).

5-Le critère « coverage »:

Le critère « coverage » est une mesure de l'étalement des points dans le plan. Plus la valeur de la mesure est proche de 0, mieux sont répartis les points dans le plan.
La formule mathématique de la mesure est :

$$coverage = \frac{1}{\bar{\gamma}} \left[\frac{1}{n} \sum_{i=1}^{n} (\gamma_i - \bar{\gamma})^2 \right]^{1/2}$$

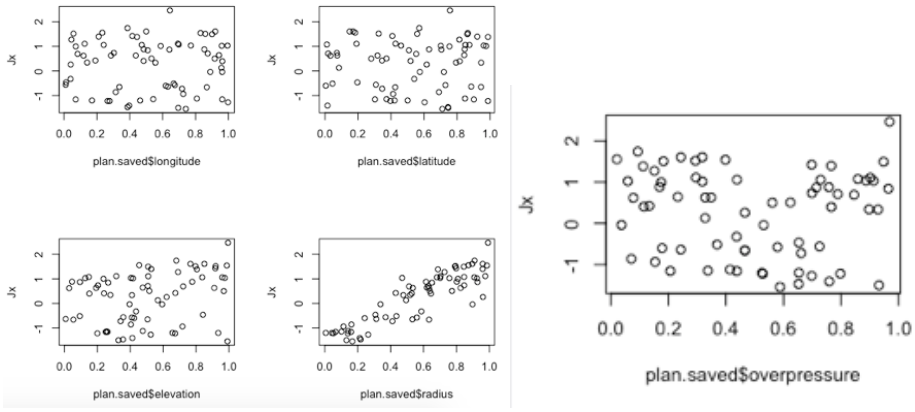
avec γi la distance minimale entre le point Xi et les autres points,γ_hat la moyenne des γi.
On remarque que la valeur de la mesure pour l'hypercube optimisé est la plus proche de 0.

Ainsi suite à ces critères, nous avons choisi de **conserver l'hypercube latin optimisé** comme plan d'expérience principal qui, suite aux visualisations graphiques et calcul de critères, semble le plan le plus adapté à notre étude (compromis entre "étalement" des points et répartition uniforme).

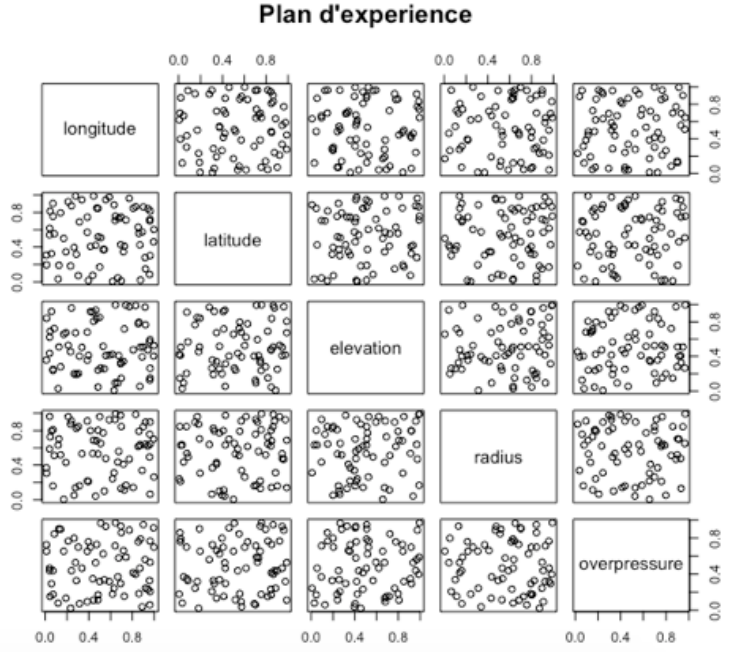
III) Modèle de Krigeage

3.1)Réflexions préliminaires

On commence tout d'abord par visualiser graphiquement la corrélation entre nos variables et la variable à expliquer. Ceci avec le plan d'expérience précédent (hypercube latin optimisé) comme base d'apprentissage.



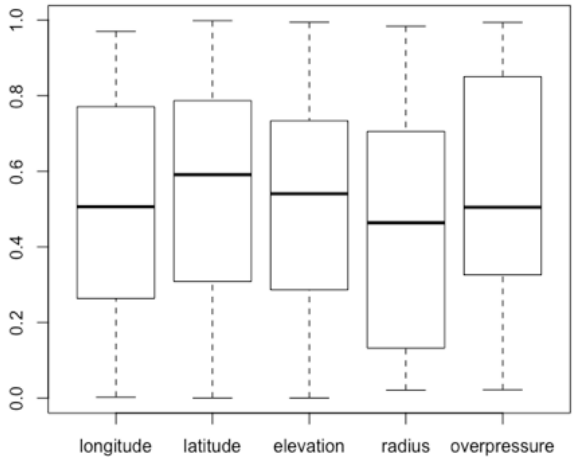
On remarque que les variables 1,2, 3 et 5 (soit la longitude, latitude, elevation et overpressure) semblent indépendantes de la variable à expliquer, tandis que la 4^{e} (la source de rayons) semble très corrélée avec la variable à expliquer.
Nous allons ensuite visualiser graphiquement la corrélation entre nos 5 variables.



On remarque ainsi que les variables explicatives ne sont pas corrélées entre elles.

Ensuite un prétraitement reste toujours nécéssaire :
-données aberrantes : ici pour une base d'apprentissage, nous n'avons pas besoin de traiter des données aberrantes car nous nous sommes chargés de construire un plan d'experience comportant des points le plus uniformément répartis possible.

En effet on peut le voir graphiquement sur la boîte à moustaches du plan :



De plus, sachant que les points du plan sont compris dans l'intervalle de variation $[0,1]^5$, nous n'avons pas besoin de les normaliser.

Toutefois, après avoir établi notre modèle de krigeage avec la base d'apprentissage, nous devrions penser à pré-traîter n'importe quel jeu de données (valeurs aberrantes si valeurs trop excessives, normaliser les variables…) avant de vouloir l'utiliser pour prédire.

Ensuite, nous pouvons nous appuyer sur la physique du problème afin d'anticiper quelles sont les variables les plus influentes, l'importance de l'interaction entre les variables…
Dans notre cas de test, nous pouvons penser que la source de rayons et l'élévation de la source sont influentes ici, car la fonction objective mesure les déplacements à la surface. Nous pouvons aussi penser à leur interaction, soit la distance entre la source de rayons et la surface.

Nous pourrons vérifier nos hypothèses par analyse de sensibilité à l'aide de la fonction **fast99.**

Après avoir construit notre modèle et afin de pouvoir le valider, nous allons tester la stationnarité et normalité des résidus.
En ce qui concerne le choix du modèle, nous avons décider de se baser sur 3 critères nous permettant de comparer les modèles :

a) Critère Q2

Le critère Q2 correspond à la part de la variance de la variable reponse expliquée par le métamodèle.
Plus il est proche de 1, meilleur est l'ajustement du modèle aux observations.

b) Critère mse (mean squared error) :

Ce critère, que l'on cherche à minimiser, est l'erreur issue de la cross validation « leave-one-out », et de formule mathématique :

$$MSE_{LOO} = \frac{1}{n}\sum_{i=1}^{n}(m_i(x_i) - Y_i)^2.$$

(différence entre les observations et la moyenne des prédictions).

c) Critère de l'erreur standardisée (mean squared error) :

Ici on calcul la variance des résidus standardizes. Ce critère est équivalent au critère précédent (mean squared error), et nous cherchons à avoir la valeur la plus proche de 1.

Nous calculons les critères à l'aide du script R suivant :

```
test<-function(model){
  resLOO <- leaveOneOut.km(model, type="UK", trend.reestim=FALSE)
  Q <- 1 - sum((resLOO$mean - model@y)^2) / sum( (model@y - mean(model@y))^2)
  M <- mean((model@y-resLOO$mean)^2)
  standardised_residuals <- (model@y-resLOO$mean)/resLOO$sd
  return(list(Q2=Q, mse= M, sd_error=sd(standardised_residuals)))
}
```

3.2) Construction des modèles

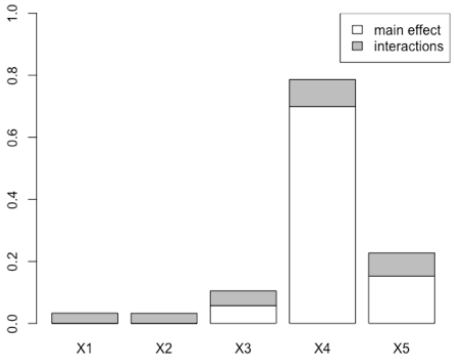
a) Choix du modele en fonction du kernel :

Nous avons commencé par construire cinq modèles, chacun associé avec une fonction kernel différente.
Nous avons ensuite comparé nos cinq modèles, et avons décidé de garder le kernel gaussien (car Q2 le plus proche de 1, mse la plus petite et standardised error la plus proche de 1).

	gauss	powexp	exp	matern5_2	matern3_2
Q2	0.94295191	0.93500508	0.8964195	0.93019156	0.92302295
mse	0.05918114	0.06742512	0.1074534	0.07241862	0.07985526
sd_error	1.02239015	0.96099146	0.7712989	1.03918182	1.03521100

Ensuite, nous allons améliorer notre modèle en changeant les tendances et manipulant les prédicteurs.

Nous commençons par estimer les indices de sobol, afin de pouvoir identifier les variables les plus importants et les interactions à utiliser dans notre modèle.
Il nous est impossible de calculer exactement les indices de sobol, nous allons donc utiliser la fonction fast99 du package « sensitivity » afin de pouvoir estimer les indices.
Les valeurs ne sont que des estimations, nous allons donc seulement évaluer le poids des variables. Nous obtenons donc les résultats suivants :




```
Estimations of the indices:
   first order total order
X1 0.0002243126  0.03328405
X2 0.0001567220  0.03256918
X3 0.0574163931  0.10500256
X4 0.6990256934  0.78641353
X5 0.1525354621  0.22720576
```

Nous remarquons que la 4e variable (ie la source de rayons) est la plus importante avec un indice de sobol S4 estime à 70%. On remarque de plus que les trois dernieres variables sont les plus importantes (elevation, source de rayons et overpressure), ceci individuellement. Nous remarquons aussi la présence d'interactions entre ces trois variables dans notre modèle.

Nous allons donc tester un nouveau modèle sous la forme :

```
m13<-km(~1+(radius+elevation+overpressure)^3,design=plan.saved, Jx, estim.method = "MLE", covtype="gauss")
```

```
Trend  coeff.:
                                 Estimate
                  (Intercept)    -1.5436
                       radius     2.7318
                    elevation     0.4886
                 overpressure     0.5404
              radius:elevation    0.2036
           radius:overpressure   -1.3025
        elevation:overpressure   -0.7352
radius:elevation:overpressure     1.8172
```

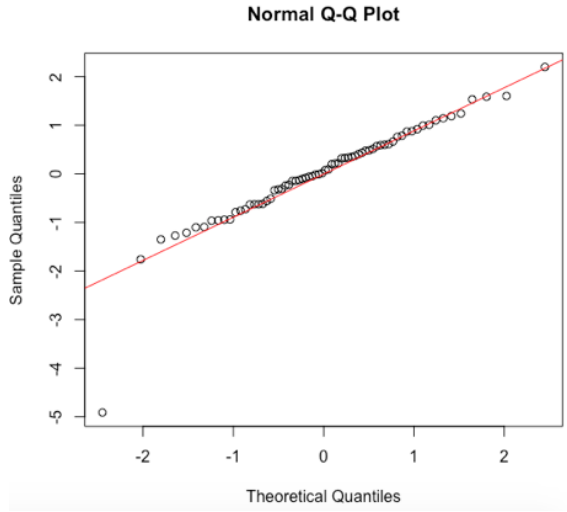
Et en testant notre modèle :

	test2
Q2	0.9550634
mse	0.0466168
sd_error	1.0033759

Nous pouvons donc voir que modifier la tendance permet d'ajuster notre modèle, en remarquant que le Q2 et standard error s'approchent de 1, le mse s'approche de 0. Nous avons donc un modèle un peu plus performant, et qui nous semble le mieux adapté à notre problème.

Validation du modele :

b)Vérification graphique :
Nous vérifions alors la normalité des résidus en traçant la droite de Henri:



On remarque que les points sont très bien alignés sur la droite, à part un seul point en retrait (en bas à gauche). On pourrait supposer que c'est un outlier ? Ceci serait difficile à dire car nous avons choisi un plan d'expérience dans lequel les points sont uniformément répartis… Nous allons donc vérifier nos hypothèses à l'aide de tests plus formels.

a)Tests statistiques:

Test de Ljung-Box (des fois appelé test « portmanteau » qui nous permet d'étudier la stationnarité des résidus) :

```
        Box-Ljung test

data:  std_res
X-squared = 2.4407, df = 1, p-value = 0.1182
```

La p-value est supérieure à 0.05, alors on ne peut pas rejeter la **non corrélation** entre les résidus.

Test de Kolmogorov Smirnov:

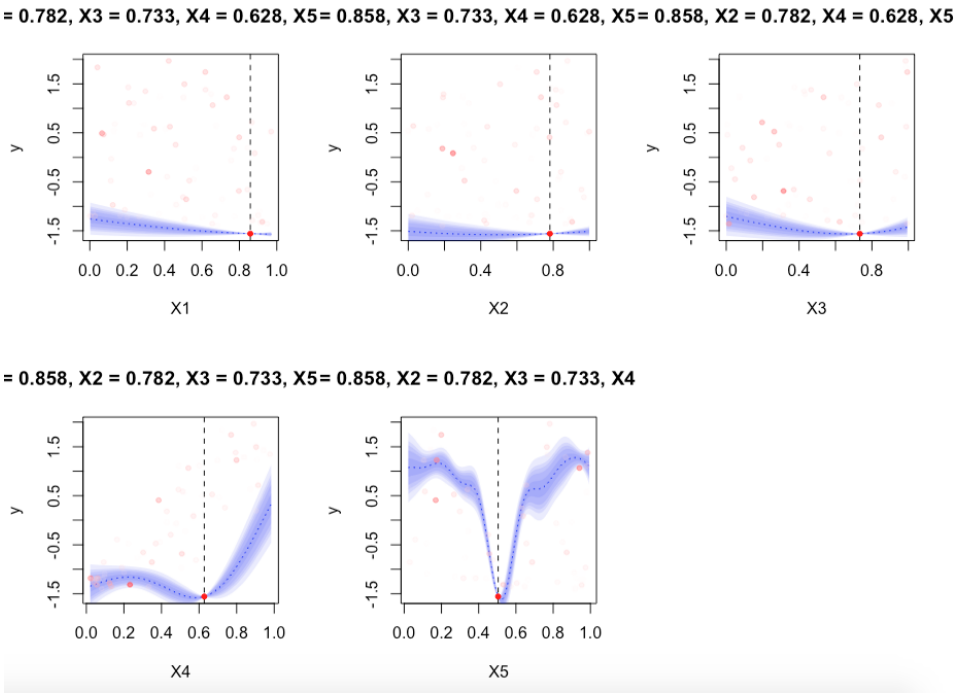
```
        One-sample Kolmogorov-Smirnov test

data:  std_res
D = 0.078561, p-value = 0.7508
alternative hypothesis: two-sided
```

```
> 0.895-D*(sqrt(70)+0.85)/(sqrt(70)-0.01)#si positive ca passe
        D
0.8083541
```

Là aussi, la p-value est supérieure à 0.05, on ne peut donc pas rejeter l'hypothèse de normalité. De plus, au niveau du seuil de 5%, on rejetterait l'hypothèse de normalité si $(\sqrt{n} + 0.85/ \sqrt{n} - 0.01)$ D était supérieur à 0.895, ce qui n'est pas le cas (avec D la statistique du test).
Alors les intervalles de confiance à 95% sur les valeurs prédites supposent que les résidus sont gaussiens. (et donc seraient indépendants vu la non corrélation)
Nos tests nous permettent donc de valider la stationnarité et la normalité des résidus, et donc notre modèle de krigeage.
Visualisation de notre modèle à l'aide de la fonction sectionview (centré en l'optimum) :

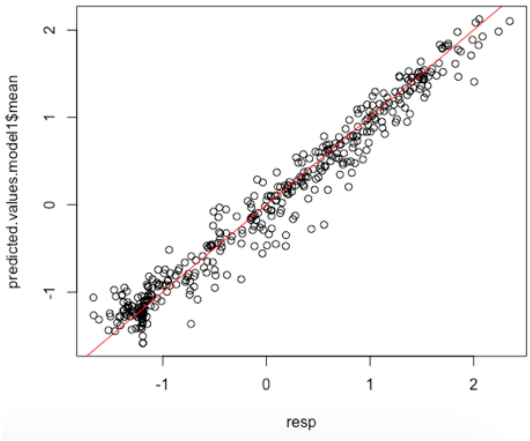
```
sectionview.km(m13, type='UK',center = as.matrix(plan.saved[which.min(Jx),]))
```



Nous allons ensuite introduire une vision critique à notre modèle, en vérifiant si l'on ne s'est pas précipité dans de l'over-fitting, et voir si l'on peut appliquer ce modèle à notre problème.
Nous effectuons alors une prédiction à partir d'un nouveau data test généré tel que :

```
n<-400
X1 <- runif(n,min = 0, max = 1 )
X2 <- runif(n,min = 0, max = 1 )
X3 <- runif(n,min = 0, max = 1 )
X4<-seq(0,1,length = 20)
X5<-seq(0,1,length = 20)
X<-cbind(X1,X2,X3,expand.grid(X4,X5))
```

Et en effectuant alors une prédiction à partir de notre modèle, nous obtenons le graphique suivant (vraies valeurs vs valeurs prédites) :



Nous remarquons que les points sont assez bien alignés, et nous pouvons donc dire que nos prédictions sont satisfaisantes et que la qualité du modèle est suffisante pour apprendre des informations sur la fonction objectif.

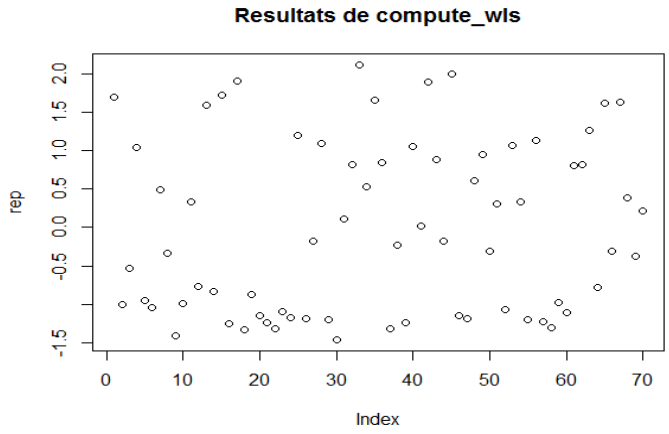
Bonus :

Afin de pouvoir améliorer notre modèle, nous pouvons ajouter séquentiellement le point qui maximise la variance de prédiction du modèle. Nous avons pensé à effectuer ceci de la manière suivante :

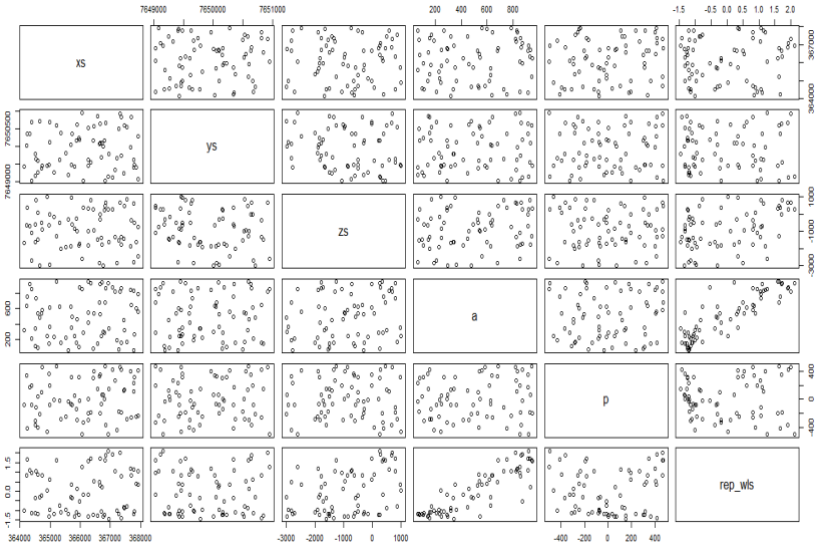
- On effectue une prédiction à partir d'un nouveau plan d'expérience (par exemple comme le plan généré précédemment pour la prédiction).
- On récupère le vecteur des résidus obtenus suite à la prédiction.
- On identifie l'indice i du résidu le plus grand (soit l'erreur la plus grande), puis l'on récupère le point i du plan d'expérience et on l'ajoute à notre base d'apprentissage, et on actualise la variable réponse.
- On répète la procédure précédente afin d'ajuster notre modèle de krigeage.

Partie I : Cas test « réservoir magmatique »

Ci-dessous les observations générées par la fonction compute_wls (par le plan d'experience conservé):

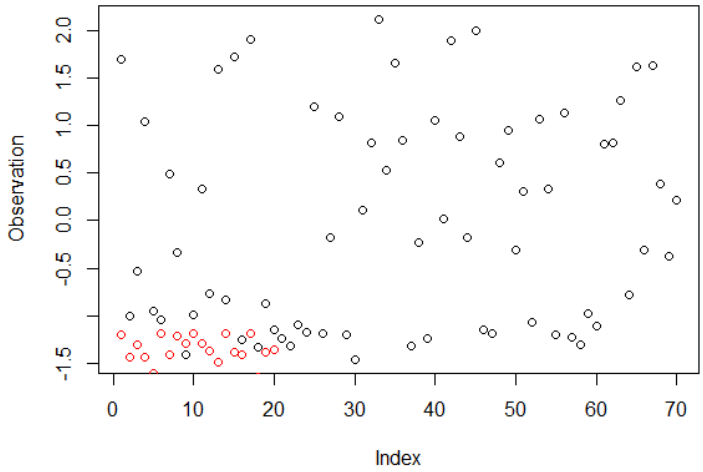


On peut aussi visualiser le comportement de la réponse en fonction des variables

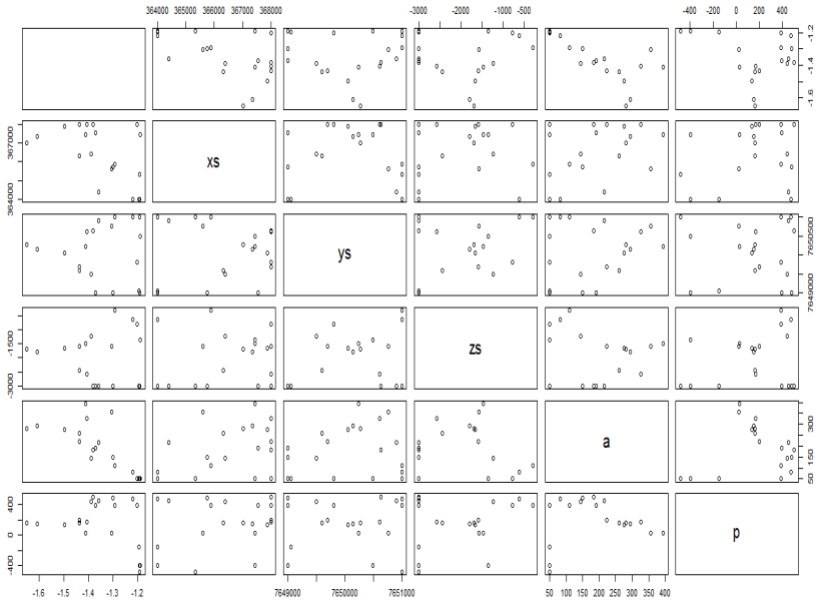


Nous avons remarqué dans la partie 3.1 (réflexions préliminaires) que les variables sont non corrélées entre elles, mais on note en revanche, une corrélation linéaire positive entre la réponse et la source de rayons a.

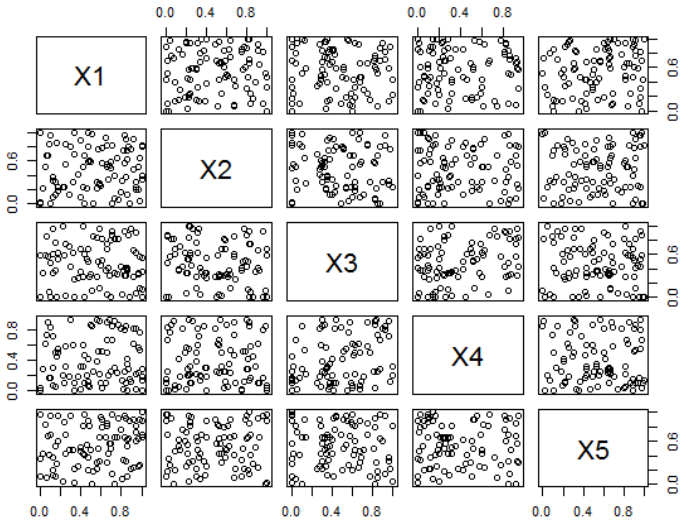
En suite, à l'aide de la fonction EGO.nsteps (20 steps) on calcule 20 nouvelles observations, à partir d'un modèle de krigeage simple (tendance constante) puis on « plotte » ces nouveaux points en rouge (observations) ; on obtient donc :



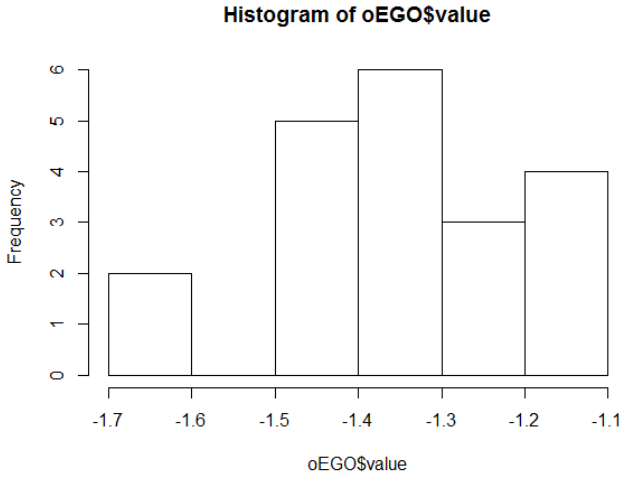
On voit clairement que les nouveaux points (en rouge) optimisent bien notre fonction objective (car minimisent la fonction). Ensuite, on étudiera le comportement des nouveaux points en fonction des variables :



Puis pour visualiser la distribution de nos observations dans le domaine d'étude, on peut voir une bonne distribution des points :



En fin, pour trouver les paramètres définissant notre source, on cherchera les valeurs des variables au minimum de notre fonction objective obtenu à l'aide de EGO on obtient donc le résultat suivant :




```
> min(oEGO$value)
[1] -1.650625
> ind <- which.min(oEGO$value)
> source_V<- unn_ego[ind,]
> source_V
          xs           ys           zs           a           p
 367011.2186 7650274.2095   -1689.1738    281.8459    161.2144
```