TP1 Krigeage

mercredi 10 décembre 2025

Note: il est recommandé de travailler avec RStudio. A l'ouverture de ce document R markdown (.Rmd), si les accents apparaissent mal, dans RStudio sélectionner File, Reopen with encoding puis utf8. Une fois le document réalisé, vous pouvez utiliser le bouton knit pour générer une sortie au format html.

Exercice 1. Un exemple minimal de Krigeage Simple

Nous allons ici considérer un exemple très minimal, que nous allons conduire pas à pas, sans utiliser de packages dédiés, pour bien comprendre la méthode.

On considère ici une fonction de réponse, disons

$$f(x) = a + sin(x)$$

On prendra dans un premier temps $a = 0.75$.

Cette fonction f est supposée inconnue, mais on observe $n = 5$ valeurs de cette fonction aux abscisses contenues dans le vecteur $X = (-2, -1, 4, 5, 7)$.

Question 1a. Noyau de covariance

Nous allons chercer à interpoler les valeurs observées de la fonction au moyen de krigeage simple.

Pour cela, nous allons modéliser les réponses par un processus Gaussien $Y(.)$, supposé centré. On suppose dans un premier temps que la covariance entre les valeurs du processus est donnée par la fonction suivante:

$$Cov(Y(x), Y(x')) = k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\theta^2}\right)$$

- Cette fonction ne dépend que de $x - x'$, quelle est la conséquence sur la nature du processus Y?
- Quel est le nom de cette fonction de covariance (noyau de covariance)?
- Rapidement, à quelle page cette fonction de covariance est-elle décrite dans le livre http://www.gaussianprocess.org/gpml/chapters/RW.pdf? Vous pourrez lire le détail plus tard, vous disposez désormais d'une référence très complète!

```
#insérer vos réponses sous forme de commentaires
```

Réponse:

```
# le processus Y est un processus stationnaire car la covariance ne dépend que de $x-x'$
#  ce noyau porte le nom de  noyau gaussien ou encore squared exponential kernel
# on le trouve par exemple décrit dans l'équation (4.9) page 101 du livre GPML, dans le chapitre 4 inti
```

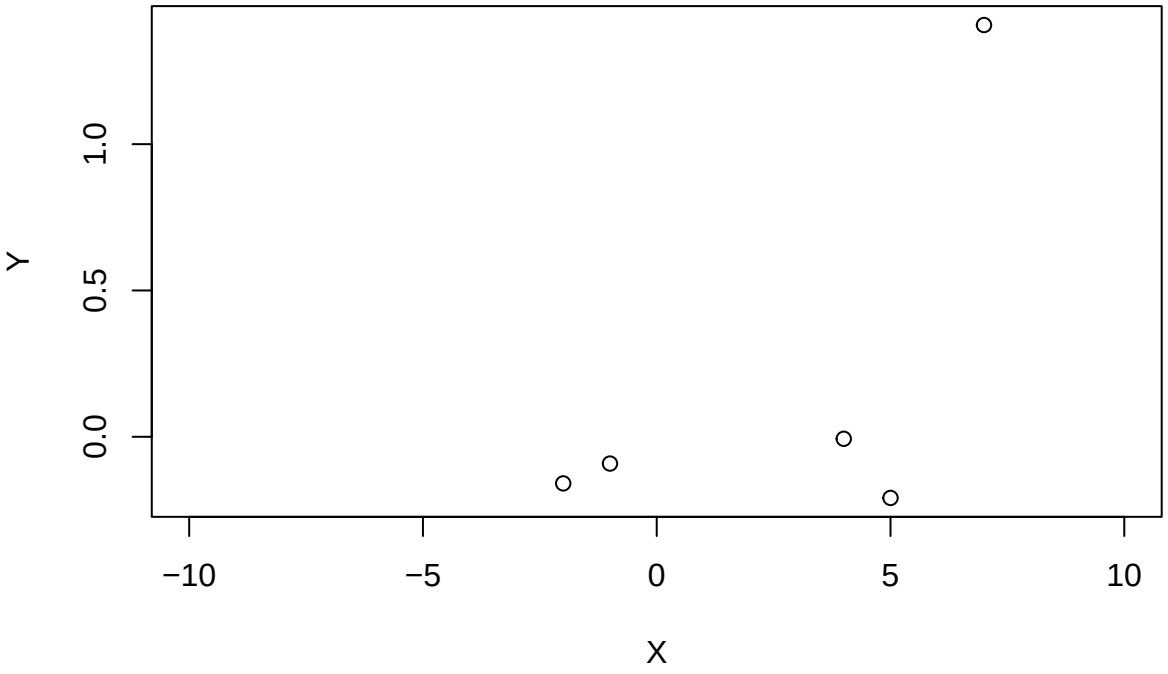
Question 1b. Observations

Créer les inputs X et le vecteur de réponses $Y = f(X)$ en ces abscisses.

```
# votre code ici
```

Réponse:

```
a <- 0.75

f <- function(x, a) { return(a+sin(x)) }
X=c(-2, -1, 4, 5, 7)
Y= f(X, a)

n = length(X)
plot(X,Y,xlim=c(-10,10))
```



Question 1c. Matrices de covariance

Nous allons chercher à prédire la fonction en des abscisses réparties régulièrement sur l'intervalle $[-10, 10]$, par exemple en $q = 100$ abscisses. Ces abscisses sont regroupées dans un vecteur colonne x^{new}.

Créer les abcisses x^{new}. En supposant $\theta = 1$ et $\sigma^2 = 1$, créer deux matrices :

- la matrice $n \times n$ de covariances $K = (K_{ij})$ avec $K_{ij} = k(x_i, x_j)$, où x_i et x_j sont des inputs observés. On note parfois cette matrice $K = k(X, X)$.
- la matrice $n \times q$ de covariances $h = (h_{ij})$ avec $h_{ij} = k(x_i, x_j^{new})$. On note parfois cette matrice $h = k(X, x^{new})$.

Ne cherchez pas à faire un code optimal, mais simplement un code lisible.

```
# votre code ici
```

Réponse:

```
q <- 100 #number of prediction points
xnew <- seq(-10, 10, length.out = q)

k <- function(x1, x2, variance, theta) {
  return(variance * exp(-(x1-x2)^2/(2*theta^2)))
}

computeK <- function(X1, X2, variance, theta) {
  n = length(X1)
  q = length(X2)
  K <-matrix(nrow = n, ncol=q)
  for(i in seq(1,n)) {
    for(j in seq(1,q)) {
      K[i,j] = k(X1[i], X2[j], variance, theta)
    }
  }
  return(K)
}

sigma2=1
theta=1

K <- computeK(X1=X, X2=X, variance=sigma2, theta=theta)
h <- computeK(X1=X, X2=xnew, variance=sigma2, theta=theta)
```

Montrer pour un vecteur **z** de votre choix que $z^\top K z \geq 0$. Ce sera le cas pour tout z si et seulement si les valeurs propres de K sont ≥ 0, montrer que ce sera bien le cas ici.

```
# votre code ici
```

Réponse:

```
z = 2*runif(nrow(K))-1
shouldBePositiveOrZero = t(z) %*% K %*% z
message("z^t K z est positif ou nul: ", (shouldBePositiveOrZero>=0))
```

```
## z^t K z est positif ou nul: TRUE
```

```
smallestEigenValue = min(eigen(K)$values)
message("K est semi-définie positive: ", (smallestEigenValue>=0))
```

```
## K est semi-définie positive: TRUE
```

Question 1d. Prédictions

La moyenne de Krigeage Simple aux abscisses x^{new} est donné par la formule:

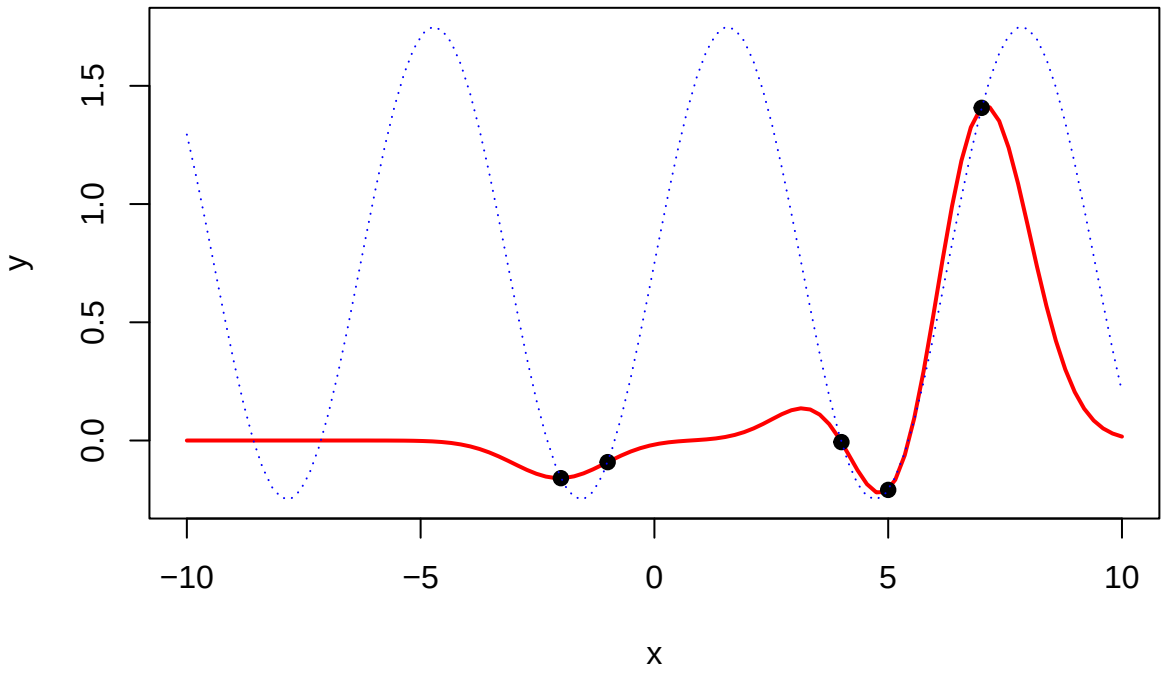
$$m(x^{new}) = h^\top K^{-1} \cdot Y$$

tracer cette moyenne de Krigeage. Ajouter le nuage de points observés (X, Y).

```
# votre code ici
```

Réponse:

```
alpha <- t(h) %*% solve(K)
m <- alpha %*% Y

trueY = f(xnew, a)

# dans les graphiques:
# type ="l" indique s'il faut tracer un trait ou des points
# lty = line type, 1 trait plein, 3 trait pointillé
# lwd = line width épaisseur du trait
# col = couleur
# pch = type de point (rond, croix, etc.)
# xlim, ylim délimitent la zone de traçage
# xlab, ylab sont les titres des axes

plot(xnew, m, type="l", xlim=c(-10,10), ylim=a+c(-1,1), xlab="x", ylab="y", lty=1, lwd=2, col="red")
points(X, Y,col="black", pch=19)
lines(xnew, trueY, col="blue", lty=3, lwd=1)
```



Au sujet du calcul de la moyenne de Krigeage:

- Une fois les matrices K et h remplies, quelle est la complexité de ce calcul, en présence de n observations?
- Chaque ligne de $\alpha = h^\top K^{-1}$ représente les poids affectés à chaque réponse dans la combinaison linéaire optimale. Ces poids se somment-ils à un ? y a-t-il des poids négatifs? Comment se comporte la prédiction loin des points observés? pourquoi?

```
# votre code ici
```

Réponse:

```
# complexité: on a une inversion de matrice n x n, complexité usuelle O(n^3), une multiplication à droi

# les poids ne se somment pas à un: pour des v.a. centrées, le prédicteur est sans biais même si les po
sumWeights <- alpha %*% seq(1,n)

# il peut y avoir des poids négatifs, rien ne l'interdit: on peut donc prédire des valeurs en dehors de
rangeWeights <- range(alpha)

message("range of weights alpha:", paste0(rangeWeights, sep=" "))
```

```
## range of weights alpha:-0.390723724203921 1.10165876556796
```

```
# on observe un retour à la moyenne du processus lorsque l'on est loin des observations: par exemple à
```

Question 1e. Intervalles de confiance

La covariance de Krigeage Simple aux abscisses x^{new} (dite encore a covariance posteriori) est donnée par la formule:

$$c(x^{new}, x^{new}) = k(x^{new}, x^{new}) - h^\top K^{-1} \cdot h$$

La variance de Krigeage est donnée par la diagonale de cette matrice

$$v(x^{new}) = diag(c(x^{new}, x^{new})) = diag(\sigma^2 I_d - h^\top K^{-1} \cdot h)$$

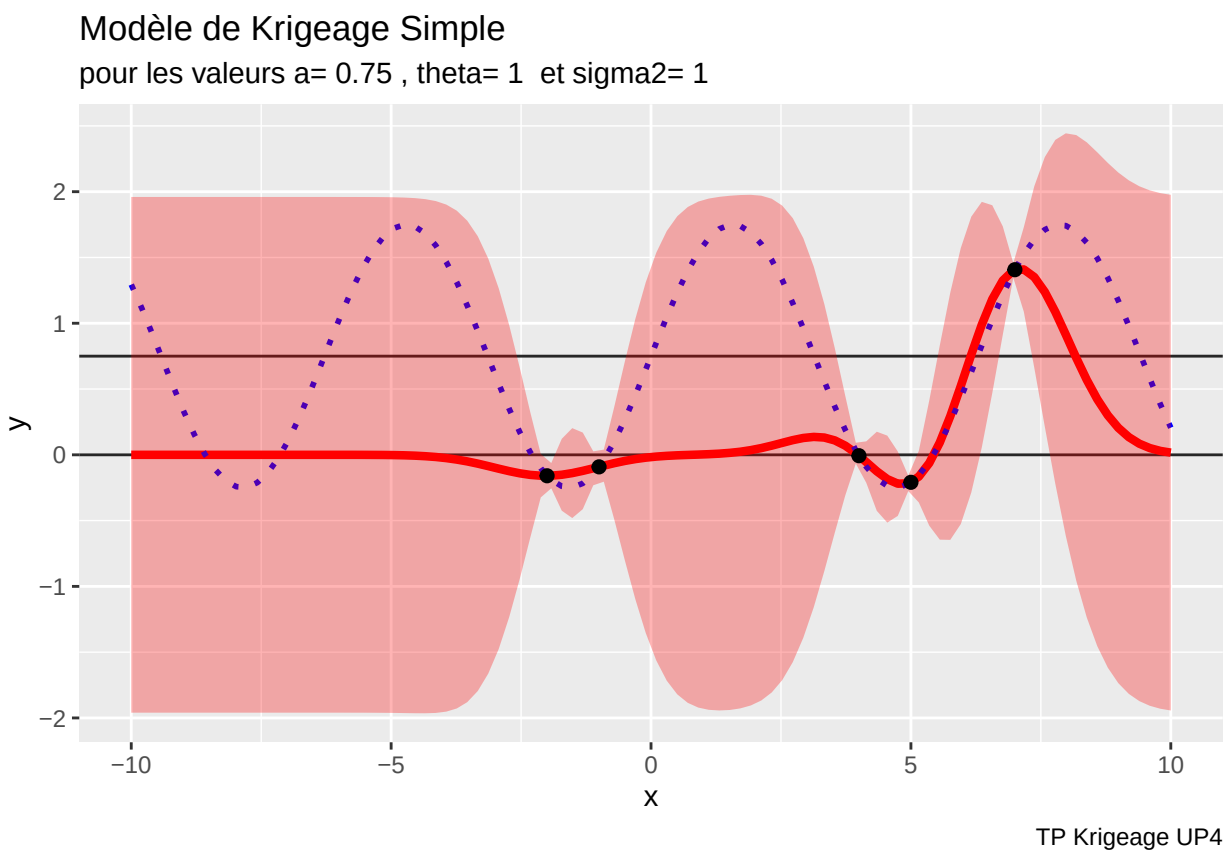
A l'aide de cette variance de Krigeage, ajouter des intervalles de confiances (sous une hypothèse Gaussienne) au graphique précédent.

```
# votre code ici
```

Réponse:

```
v <- diag(sigma2 - t(h) %*% solve(K) %*% h)
# on pourrait aussi réutiliser le vecteur alpha bien sûr

# pour le graphique, nous allons ici utiliser ggplot2

library(ggplot2)

#on englobe d'abord les résultats dans un data frame, pour nommer les colonnes

drawPrediction <- function(X, Y, xnew, m, v, title="Krigeage", colour="red") {
dfPoints <- data.frame(x= X, y=Y)
dfPredict <- data.frame(x=xnew, y=m, ymin = m - 1.96*sqrt(v), ymax = m + 1.96*sqrt(v) )
dfTrue <- data.frame(x=xnew, y=trueY)

 ggplot(dfPoints, aes(x,y)) +
   geom_hline(yintercept=c(0,a), linetype = "solid", linewidth=0.5, colour="grey15")+
   scale_x_continuous(limits = c(-10,10)) +
   geom_line(data= dfPredict, aes(x,y), colour=colour, linewidth=1.5, linetype = "solid") +
   geom_line(data= dfTrue, aes(x,y), colour="blue", linewidth=1, linetype ="dotted") +
   geom_ribbon(data= dfPredict, aes(x=x, ymin=ymin, ymax=ymax),    alpha=0.3, fill=colour)+
```

```
  labs(title = title,
    subtitle = paste("pour les valeurs a=", a,", theta=", theta, " et sigma2=", sigma2),
    caption = "TP Krigeage UP4") +
    geom_point(data=dfPoints, aes(x=x, y=y), size=2, colour="black")
}

drawPrediction(X, Y, xnew, m, v, "Modèle de Krigeage Simple")
```



Question 2. Utilisation de paquets logiciels

Nous allons reprendre ici l'exemple élémentaire précédent, et retrouver les résultats obtenus à l'aide d'un paquet logiciel tout prêt *DiceKriging*, disponible sur le dépôt logiciel CRAN. Sauf mention contraire, toutes les variables utilisées seront celles de la question 1.

Question 2a Utilisation de DiceKrigring

Installer la librairie DiceKriging (dans R Studio, utiliser Tools, Install Packages). Charger cette librairie

```
# votre code ici
```

Réponse:

```
library(DiceKriging)
```

Question 2b Modèle de Krigeage

Au moyen de la fonction *km* créer un modèle de Krigeage Simple, que vous stockerez dans une variable *monModele*. (Pour schématiser *km* va calculer la matrice K et son inverse, de façon à pouvoir faire ensuite plusieurs prédictions sans recalculer cette matrice.)

Vous aurez besoin d'utiliser les options suivantes:

- Pour le type de Krigeage,
  - `formula = ~1` et `coef.trend = 0` pour du Krigeage Simple (on a un trend constant égal à zero)
  - `formula = ~1` et `coef.trend = NULL` pour du Krigeage Ordinaire (on a un trend constant à estimer)
  - `formula = ~Y~1+X` (ou `formula = ~Y~1+X+I(X^2))`et `coef.trend = NULL` pour du Krigeage Universel.
- Pour les observations: `design = ...`, `response = ...` pour indiquer les inputs et output observés.
- Pour le noyau de covariance: `covtype=...` (type de noyau), `coef.cov = ...` pour le(s) paramètre(s) θ, `coef.var =` pour la variance σ^2

```
# votre code ici
```

Réponse:

```
monModele <- km(formula = ~1, design = X, response = Y, covtype = "gauss", coef.trend = 0, coef.cov = tl
```

Ensuite, établissez une prédiction aux points *x^{new}* au moyen de la fonction `predict` de DiceKriging. Vous aurez à utiliser les paramètres suivants:

- `object=...` pour spécifier le modèle de Krigeage, vous indiquerez `monModele`.
- `newdata=...` pour spécifier les nouveaux inputs où l'on souhaite prédire
- `type=...` pour spécifier le prédicteur désiré, "SK" pour Simple Kriging, ou "UK" pour Ordinary et Universal.
- `checkNames=FALSE` si vous ne souhaitez pas vérifier les noms des variables dans les objets en entrée de type dataframe.

trouver l'option à utiliser pour calculer les variances (ou écart types) de Krigeage. Extraire moyenne et variance de Krigeage dans des variables m_{Dice} et v_{Dice}.

```
# votre code ici
```

Réponse:

```
prediction <- predict(object = monModele, newdata = xnew , type="SK" , checkNames=FALSE, se.compute=TRU

m_Dice <- prediction$mean
v_Dice <- (prediction$sd)^2
```

Réponse:

Question 2c

Comparer les résultats obtenus avec ceux de l'exercice 1. Renvoyez un booléen qui renvoie TRUE ou FALSE selon que les résltats sont similaires ou non.

```
# votre code ici
```

```
maxErrorMean <- max(abs(m_Dice-m))
maxErrorVar <- max(abs(v_Dice-v))

isIdentical <- (maxErrorMean<1e-10)&(maxErrorVar<1e-10)

message("les résultats sont identiques : " , isIdentical)
```

```
## les résultats sont identiques : TRUE
```

Remarque: prenez l'habitude de tester ainsi vos résultats, à la façon de *tests unitaires*, cf. https://en.wikipedia.org/wiki/Unit_testing . Il est commun de vérifier manuellement le bon comportement de fonctions, puis malheureusement d'effacer ces vérifications. Le fait de préserver des tests automatisés, renvoyant des booléens faciles à agréger, permet de s'assurer du bon fonctionnement d'un programme au fur et à mesure qu'il évolue.

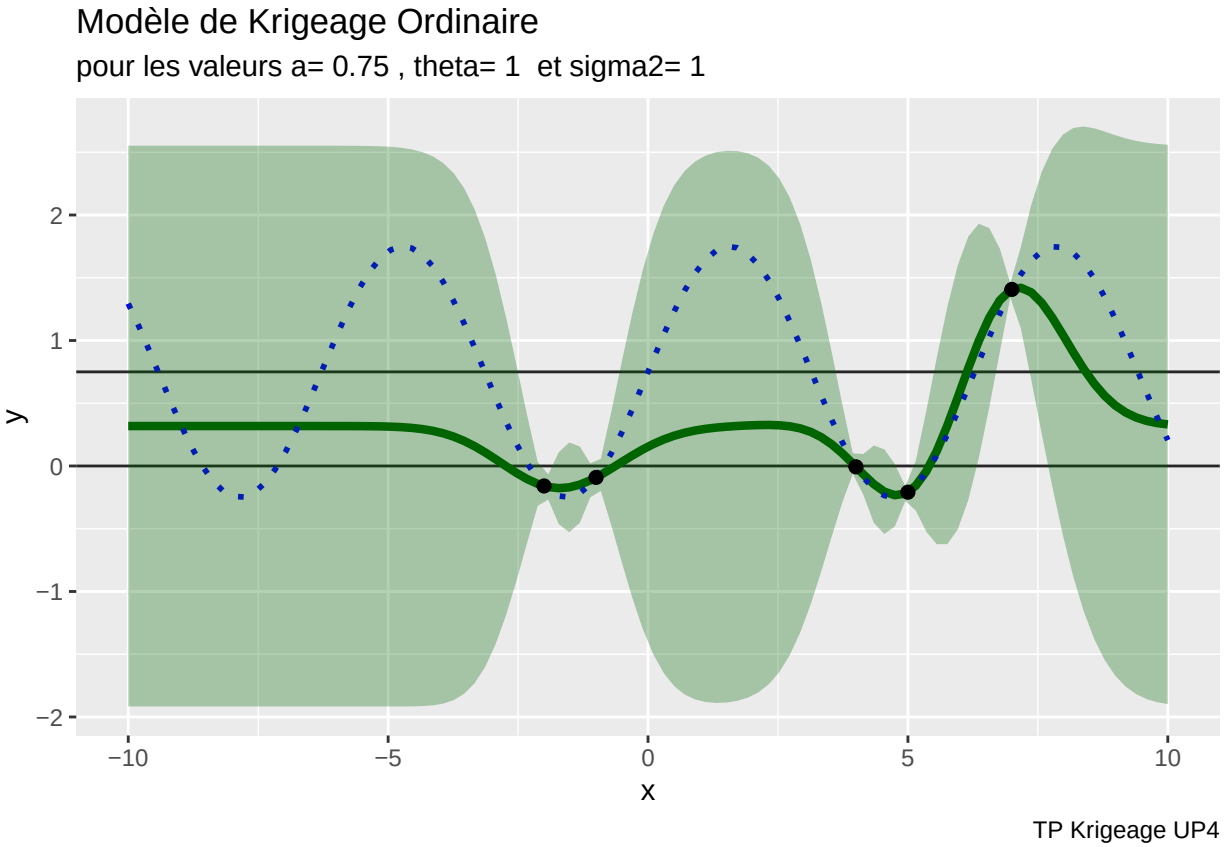
Question 2d

Effectuez une nouvelle prédiction avec DiceKriging, en utilisant un Krigeage Ordinaire avec les mêmes paramètres. Tracer a prédiciton obtenue, quelle est la différence avec le Krigeage Simple?

```
# votre code ici
```

Réponse:

```
monModele_Ordinary <- km(formula = ~1, design = X, response = Y, covtype = "gauss", coef.trend = NULL, c
prediction_Ordinary <- predict(object = monModele_Ordinary, newdata = xnew , type="UK" , checkNames=FALS

m_Ordinary <- prediction_Ordinary$mean
v_Ordinary <- (prediction_Ordinary$sd)^2
drawPrediction(X, Y, xnew, m_Ordinary, v_Ordinary, "Modèle de Krigeage Ordinaire", "darkgreen")
```




```
# On constate que, loin des observations, le processus se stabilise près de la moyenne estimée des obse
```

Remarque: des indications et exemples sur le package DiceKriging sont aussi disponibles ici: https://hal.archives-ouvertes.fr/hal-00495766/file/jss642.pdf

Question 3

Un industriel souhaite optimiser la conception d'aubes (de pales) de turbine. Il souhaite pour cela établir un plan d'expérience, puis un métamodèle qui lui permettra d'estimer la performance d'un profil d'aube donné.

A l'issue d'une première étude, deux paramètres de géométrie des aubes semblent cruciaux pour la performance globale de la turbine, ces deux paramètres ont été normalisés et prennent chacun des valeurs dans $[0, 1]$. Pour la construction future du métamodèle, l'industriel souhaite utiliser un Krigeage simple. Les performances qu'il mesure semblent évoluer de façon très régulière, il souhaite donc opter pour un noyau de covariance Gaussien, de longueurs de corrélation (lengthscales) $\theta = 0.3$ pour chacune des deux dimensions.

L'industriel envisage de procéder en deux étapes:

- Créer un plan d'expérience simple, construire des prototypes et effectuer des mesures précises de performance pour $n = 20$ profils différents.
- Puis créer et tracer un métamodèle, à partir d'un Krigeage Simple des observations effectuées.

La réalisation des n prototypes et essais devrait prendre un mois, il est donc important de ne pas faire d'essais trop similaires...

3a. Plan optimisé Kriging Based Design - grille d'évaluation

Créer une fonction pour évaluer numériquement le critère IMSE d'un plan donné.

Pour cela, on va tout d'abord créer une grille de 10000 points, couvrant le carré $[0,1]^2$. On stockera ces points dans une matrice x^{new} de taille 10000 x 2 par exemple. Vous pourrez utiliser pour cela la fonction `expand.grid`.

```
# votre code ici
```

Réponse:

```
set.seed(1234567) #in order to get reproducibe results
sequence1D = seq(0, 1, length.out = 100)
xnew = expand.grid(sequence1D, sequence1D, KEEP.OUT.ATTRS=FALSE)
```

3b. Plan optimisé Kriging Based Design - critère IMSE

Par Krigeage Simple et pour un plan X, on en déduit alors la valeur de la variance de Krigeage sur chaque point de la grille x^{new}, une moyenne de ces grandeurs donnera le critère IMSE.

Tout d'abord, pour un plan X donné, la variance de Krigeage dépend-elle des réponses Y observées? La minimisation d'un critère IMSE sera-t-elle affectée par la variance σ^2 de la réponse ?

```
# votre code ici
```

Réponse:

```
# La variance de Krigeage ne dépend pas des réponses, et elle est proportionnelle à $\sigma^2$. On peut
```

Créer un plan X de n essais dans $[0,1]^2$, tirés de façon uniforme qui permettra de tester votre fonction.

Calculer par Krigeage simple, à l'aide de DiceKriging, la variance de Krigeage en chaque point x^{new}. Attention, vous aurez besoin d'un paramètre de portée (lengthscale θ) par dimension. Quand tout fonctionne, encapsulez le tout dans une fonction IMSE, prenant en argument un plan X, et renvoyant le critère IMSE.

```
# votre code ici
```

Réponse:

```
library(DiceKriging)
nbEssais = 20
X <- matrix(runif(nbEssais*2), ncol=2, nrow = nbEssais)

theta=0.3


IMSE <- function(X) {
  Y = rep(0, times = nrow(X))
  sigma2=1

  modele <- km(formula = ~1, design = data.frame(X), response = Y, covtype = "gauss", coef.trend = 0, c
  prediction <- predict(object = modele, newdata = xnew , type="SK" , checkNames=FALSE, se.compute=TRUE
```

```
  v <- (prediction$sd)^2
  return(mean(v))
}

IMSE(X)
```

```
## [1] 0.01710645
```

3c. Plan optimisé 'Kriging Based Design' final

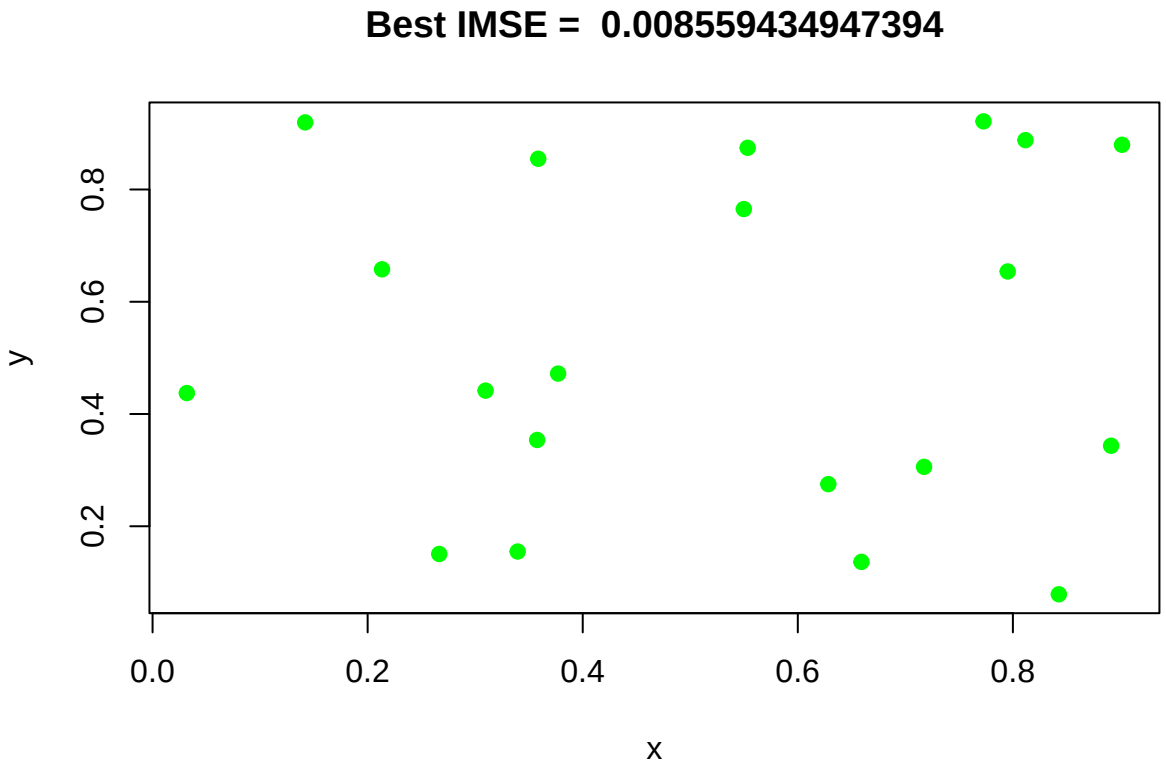
Répéter 1000 fois le tirage aléatoire d'un plan X et le calcul de l'IMSE correspondant. Sauvegarder le meilleur et le pire plan obtenu pour ce critère. Nommez ces deux plans *Xbest* et *Xworse*, tracez ces plans.

Remarque: On pourrait bien sûr améliorer le meilleur plan en partant de plans plus optimisés (l'enjeu était ici de voir aussi de "mauvais" plans).

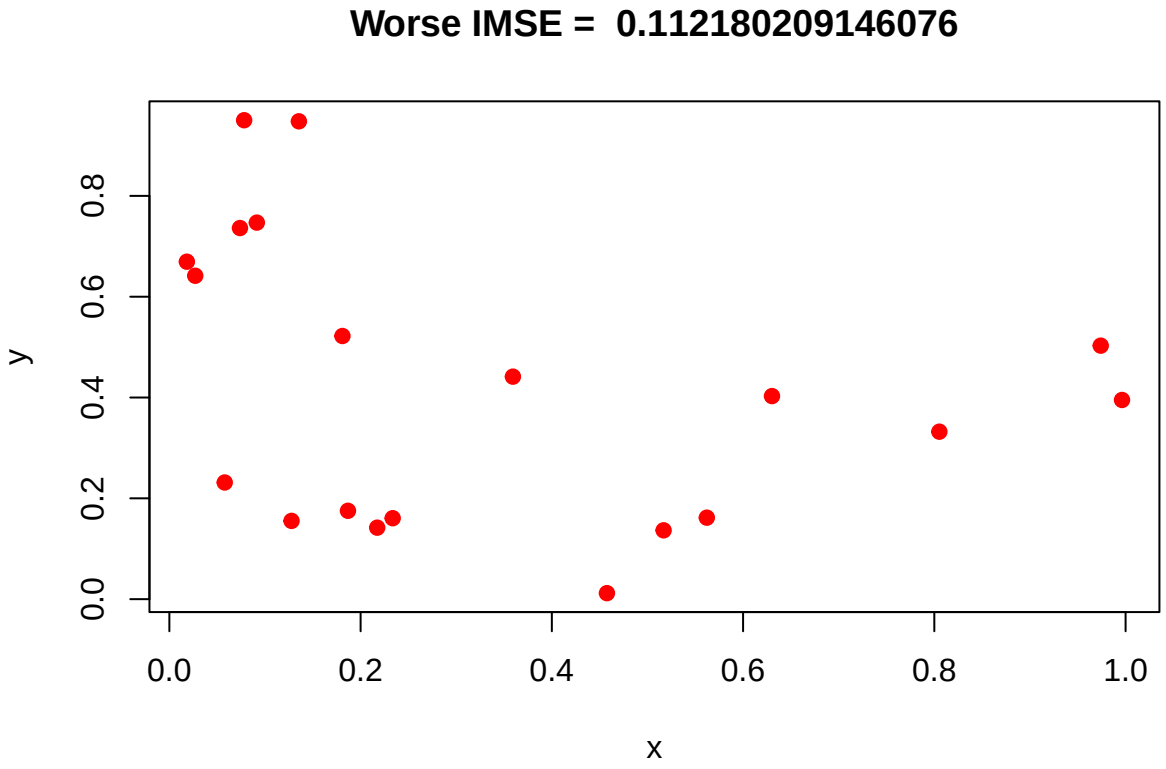
```
# votre code ici
```

Réponse:

```
bestIMSE = 1e10
worseIMSE = 0
for(i in 1:1000) {
  X <- matrix(runif(nbEssais*2), ncol=2, nrow = nbEssais)
  currentIMSE <- IMSE(X)
  if (currentIMSE <= bestIMSE)  {
    Xbest = X
    bestIMSE = currentIMSE
    }
  if (currentIMSE >= worseIMSE) {
    Xworse = X
    worseIMSE = currentIMSE
    }
}
plot(Xbest[,1],Xbest[,2],pch = 21, bg = "green",col  = "green",xlab = "x",ylab='y', main=paste("Best IM
```




```
plot(Xworse[,1],Xworse[,2],pch = 21, bg = "red",col  = "red",xlab = "x",ylab='y', main=paste("Worse IMSE
```



3d. Le grand jour

Après un mois de travaux, de construction de turbines et de tests de performance, l'industriel est en mesure de donner la mesure de performance pour chacun des n points du meilleur plan d'expérience *Xbest*.

Nous supposerons ici que cette performance nommée Y est le résultat de la fonction *branin* du package DiceKriging.

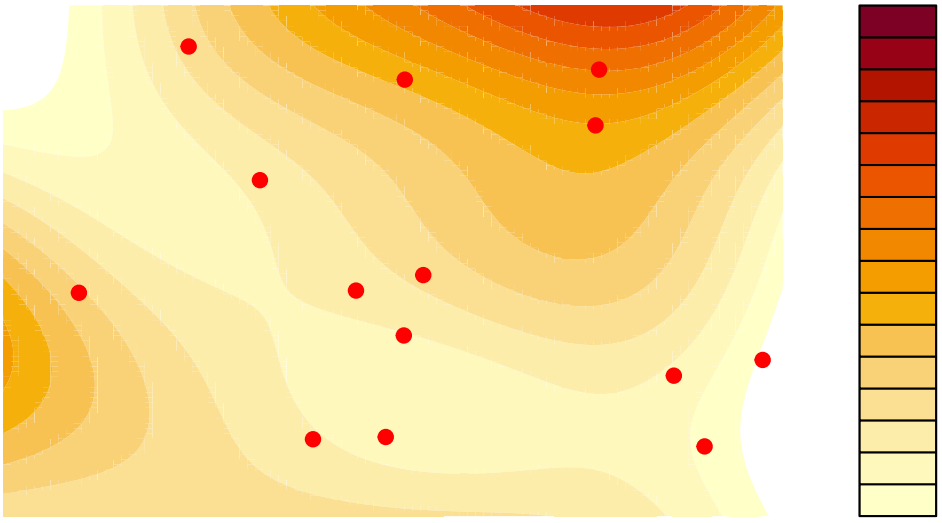
Calculer Y pour chaque point de *Xbest*, puis calculer la moyenne de Krigeage Simple en chacun des points de *xnew*. En vous inspirant de ce code https://rdrr.io/cran/DiceKriging/man/branin.html tracer les courbes de niveau de votre moyenne de Krigeage. On pourra remplacer `contour(x,y,z,40)` par `filled.contour(x,y,z)` pour un résultat plus visuel.

```
# votre code ici
```

Réponse:

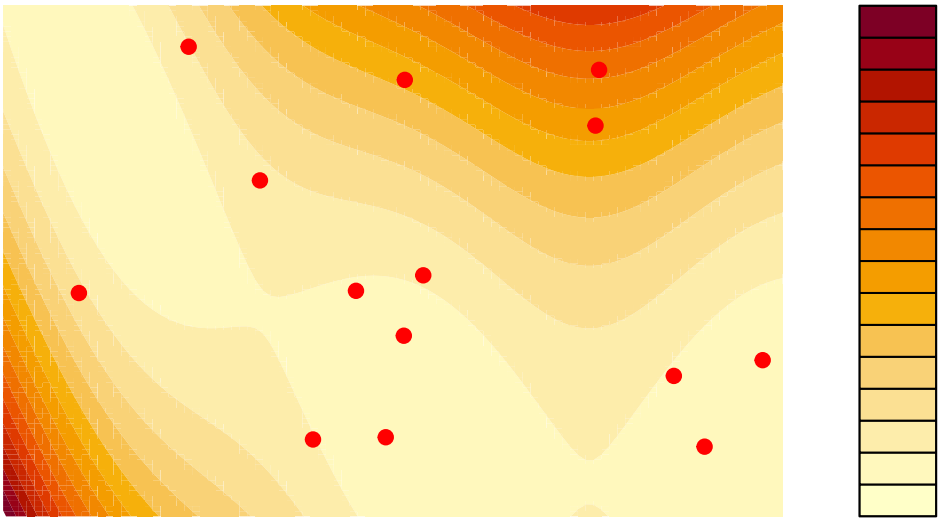
```
Y = apply(Xbest, MARGIN = 1, branin)
sigma2=1

modeleLast <- km(formula = ~1, design = data.frame(Xbest), response = Y, covtype = "gauss", coef.trend =
prediction <- predict(object = modeleLast, newdata = xnew , type="SK" , checkNames=FALSE, se.compute=TRU

n.grid <- 100
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x.grid, y.grid)
response.gridMK <- prediction$mean
z.gridMK <- matrix(response.gridMK, n.grid, n.grid)

response.grid <- apply(design.grid, 1, branin)
z.grid <- matrix(response.grid, n.grid, n.grid)

filled.contour(x.grid,y.grid,z.gridMK, xlim=c(0,1), ylim=c(0,1), zlim=c(-10,300), axes=FALSE)

points(Xbest, pch=19, col="red")
title("Moyenne de Krigeage")
```

Moyenne de Krigeage



```
filled.contour(x.grid,y.grid,z.grid, xlim=c(0,1), ylim=c(0,1), zlim=c(-10,300), axes=FALSE)
points(Xbest, pch=19, col="red")
title("Vraie fonction")
```

Vraie fonction



```
# compte tenu du faible nombre d'observations,
# c'est plutôt pas mal comme ajustement, non?
```
