## VARIOGRAMMES, KRIGEAGE

Le logiciel utilisé est R, avec les packages *geoR* et *fields*.

1. **Données simulées** On dispose d'une simulation d'un champ gaussien sur un carré 101×101. On extrait 100 points du carré et on va reconstruire l'image sur le carré par krigeage. Pour cela il faudra d'abord ajuster un variogramme sur une fonction choisie. On comparera les résultats obtenus avec différents variogrammes à l'image originale.

   (a) Visualisation

   Le fichier *simu1.dat* contient une simulation d'un processus gaussien sur une grille 101×101.
   Charger le fichier avec la fonction *read.table*.
   Les 3 variables : $x$ abcisse, $y$ ordonnée et $z$ valeur du champ sont en colonnes.
   Les fonctions *summary* et *hist* donnent les statistiques élémentaires et l'histogramme d'une variable.
   Visualiser le champ avec la fonction *image.plot*. Il faudra tout d'abord transformer le vecteur $z$ en une matrice avec la fonction *matrix*.

```
> #---------------------------------
> # chargement du fichier de donnees
> #---------------------------------
> aniso = 0
> donnees = read.table("jeu1.dat")
> summary(donnees)
```

```
       V1              V2              V3
 Min.   :  1.00   Min.   :  2.00   Min.   :-4.11096
 1st Qu.: 23.00   1st Qu.: 24.00   1st Qu.:-0.79057
 Median : 51.50   Median : 43.00   Median :-0.06325
 Mean   : 51.07   Mean   : 47.65   Mean   :-0.09107
 3rd Qu.: 75.75   3rd Qu.: 72.25   3rd Qu.: 0.75895
 Max.   :100.00   Max.   :100.00   Max.   : 3.16228
```

```
> donnees.simu = read.table("simu1.dat")
> x=donnees$V1
> y=donnees$V2
> #----graphique du champs--------
> grx = seq(0,100)
> gry = seq(0,100)
> Z = matrix(donnees.simu$V3,nrow=length(grx),ncol=length(gry),
+            byrow=F)
> titre = paste("Gaussian field")
> image.plot(grx,gry,Z,main=titre)
> points(x,y,pch=19)
```

(b) Krigeage

Le fichier *jeu1.dat* contient 100 points tirés au hasard parmi les précédents. Le charger, visualiser les points avec la fonction *points*, et créer un objet *geodata*.

i. Variogramme empirique :

Calculer et visualiser le variogramme empirique avec les fonctions *variog* et *plot*. Faire varier le nombre d'intervalles, les largeurs d'intervalle, la distance maximale.

Le nuage variographique (sous forme de nuage de points ou de boxplot) est une nuée de $n(n-1)/2$ points $(\|s_i - s_j\|, (Z_{s_i} - Z_{s_i})^2/2)$ permettant d'estimer sans biais $\gamma(s_i - s_j)$ (le semi-variogramme aux sites $(s_i, s_j)$). Ce nuage n'est pas très lisible, il ne suffit pas pour avoir une idée sur les caractéristiques de $\gamma$ comme la portée, le palier ou la pépite. On utilise alors le variogramme expérimental pour une représentation de la variabilité spatiale plus visible.
La simulation est isotropique, sinon (cas anisotropique) il peut être utile de représenter plusieurs variogrammes selon plusieurs directions de l'espace (en pratique on regarde les 4 orientations cardinales, S, SE, E, NE, avec la fonction "variog4").

```
> ####
> # variogramme
> #------------
> # 1. Variogramme empirique
> #-------------------------
> geodata = as.geodata(donnees)#convertir les observations en geodata
> m.d = 50                     # distance maximale
> interv = seq(0,m.d,by=5)     # intervalles
> p.m = 10                    # nombre minimal de paire,
> #en pratique on choisit 30 paires
>
> vario.c = variog(geodata,op="cloud")

variog: computing omnidirectional variogram

> plot(vario.c,main = "",pch='+')
> vario.b = variog(geodata,max.dist=m.d,pairs.min=p.m,
+                  breaks=interv)

variog: computing omnidirectional variogram

> plot(vario.b,main = "")
> vario.bc = variog(geodata,max.dist=m.d,pairs.min=p.m,
+                   breaks=interv,bin.cloud=TRUE)

variog: computing omnidirectional variogram

> plot(vario.bc,main = "Box-plot sur le variogramme empirique",
+      bin.cloud=TRUE)
> #ou   sans choix de distance max ou nombre de couples de sites
> vario.b = variog(geodata)

variog: computing omnidirectional variogram
```

```
> plot(vario.b,main = "")
> ### Variogramme pluri-directionnelle
>
> vario.b4 = variog4(geodata)

variog: computing variogram for direction = 0 degrees (0 radians)
        tolerance angle = 22.5 degrees (0.393 radians)
variog: computing variogram for direction = 45 degrees (0.785 radians)
        tolerance angle = 22.5 degrees (0.393 radians)
variog: computing variogram for direction = 90 degrees (1.571 radians)
        tolerance angle = 22.5 degrees (0.393 radians)
variog: computing variogram for direction = 135 degrees (2.356 radians)
        tolerance angle = 22.5 degrees (0.393 radians)
variog: computing omnidirectional variogram

> plot(vario.b4,main = "")
```

ii. Variogramme ajusté :
Ajuster le variogramme expérimental avec les fonctions *variofit* et *lines* selon les modèles sphérique, exponentiel, gaussien, Matern, avec ou sans pépite. La valeur de pépite peut être imposée ou optimisée.
Comparer les différents ajustements obtenus.

On ajuste ici un modèle exponentiel en donnant des paramètres initiaux pour portée (10), palier (1.5) (on suppose qu'à la distance 10 le variogramme converge vers le palier et la covariance est proche de 0) déduits du variogramme experimental. On suppose qu'il n'y a un effet pépite (la limite du variogramme en zéro ou variance d'un terme d'erreur). Une pépite peut être dûe a une erreur de mesure (variabilité de l'instrument de mesure ou un changement brusque dans l'espace de la variable mesurée). Le modèle paramètrique adapté doit être choisi parmi un ensemble de modèles de variagramme admissibles.

```
> # 2. Ajustement du variogramme
> #-----------------------------
> c.m = "exponential"
> i.c = c(1.5,10)
> varioest = variofit(vario.b,cov.model = c.m,fix.kappa=TRUE,
+ ini.cov.pars=i.c,fix.nugget=T) ## on fixe la pepite à 0, "F" estime la pepi

variofit: covariance model used is exponential
variofit: weights used: npairs
variofit: minimisation function used: optim

> x11()
> titre = paste("modele ",c.m,", portee =",round(varioest$cov.pars[2],2),
+ ", palier =",round(varioest$cov.pars[1],2),
+ "nu = ",round(varioest$kappa,2))
> plot(vario.b, main=titre)
> lines(varioest)
```

Le résultat de cette étude variographique nous dit que la variabiloté spatiale du processus est modélisée par un modèle exponentiel de portée 11.78 (au dela d'une distance de 11.78, la dépendance devient faible), un palier de 1.6 (le variogramme converge vers 1.6, la variabilité limite)

iii. Krigeage :
A partir des différents variogrammes obtenus ci-dessus, réaliser les cartes de krigeage (fonction *krige.conv*) et de variance associée. La grille (fonction *expand.grid*) peut être plus ou moins fine (cela influe sur le temps de calcul).
Discuter les résultats obtenus.

```
> # 3. Krigeage
> #------------
> grx = seq(0,100)
> gry = seq(0,100)
> grille = expand.grid(grx,gry)# l'ensemble S
> Kcontrol = krige.control(type.krige="ok",obj.model=varioest)
> #krigeage ordinaire
> Ocontrol = output.control(n.pred=100,simul=TRUE,thres=2)
> K = krige.conv(geodata,loc=grille,krige=Kcontrol)

krige.conv: model with constant mean
krige.conv: Kriging performed using global neighbourhood

> # l'échantillon est geodata (les 100 donnees de jeu1,
> #l'ensemble S de prediction est la grille et la fonction de variogramme est l'
>
> variofit

function (vario, ini.cov.pars, cov.model, fix.nugget = FALSE,
    nugget = 0, fix.kappa = TRUE, kappa = 0.5, simul.number = NULL,
    max.dist = vario$max.dist, weights, minimisation.function,
    limits = pars.limits(), messages, ...)
{
    call.fc <- match.call()
    if (missing(messages))
        messages.screen <- as.logical(ifelse(is.null(getOption("geoR.messages")),
            TRUE, getOption("geoR.messages")))
    else messages.screen <- messages
    if (length(class(vario)) == 0 || all(class(vario) != "variogram"))
        warning("object vario should preferably be of the geoR's class \"variogra
    if (!missing(ini.cov.pars)) {
        if (any(class(ini.cov.pars) == "eyefit"))
            cov.model <- ini.cov.pars[[1]]$cov.model
        if (any(class(ini.cov.pars) == "variomodel"))
            cov.model <- ini.cov.pars$cov.model
    }
    if (missing(cov.model))
        cov.model <- "matern"
    cov.model <- match.arg(cov.model, choices = .geoR.cov.models)
    if (cov.model == "stable")
```

```
    cov.model <- "powered.exponential"
if (cov.model == "powered.exponential")
    if (limits$kappa["upper"] > 2)
        limits$kappa["upper"] <- 2
if (missing(weights)) {
    if (vario$output.type == "cloud")
        weights <- "equal"
    else weights <- "npairs"
}
else weights <- match.arg(weights, choices = c("npairs",
    "equal", "cressie"))
if (messages.screen) {
    cat(paste("variofit: covariance model used is", cov.model,
        "\n"))
    cat(paste("variofit: weights used:", weights, "\n"))
}
if (missing(minimisation.function))
    minimisation.function <- "optim"
if (any(cov.model == c("linear", "power")) & minimisation.function ==
    "nls") {
    cat("warning: minimisation function nls can not be used with given cov.mod
    minimisation.function <- "optim"
}
if (minimisation.function == "nls" & weights != "equal") {
    warning("variofit: minimisation function nls can only be used with weights
    minimisation.function <- "optim"
}
if (is.matrix(vario$v) & is.null(simul.number))
    stop("object in vario$v is a matrix. This function works for only 1 empiri
if (!is.null(simul.number))
    vario$v <- vario$v[, simul.number]
if (mode(max.dist) != "numeric" || length(max.dist) > 1)
    stop("a single numerical value must be provided in the argument max.dist"
if (max.dist == vario$max.dist)
    XY <- list(u = vario$u, v = vario$v, n = vario$n)
else XY <- list(u = vario$u[vario$u <= max.dist], v = vario$v[vario$u <=
    max.dist], n = vario$n[vario$u <= max.dist])
if (cov.model == "pure.nugget") {
    minimisation.function <- "not used"
    message <- "correlation function does not require numerical minimisation"
    if (weights == "equal")
        lm.wei <- rep(1, length(XY$u))
    else lm.wei <- XY$n
    if (cov.model == "pure.nugget") {
        if (fix.nugget) {
            temp <- lm((XY$v - nugget) ~ 1, weights = lm.wei)
            cov.pars <- c(temp$coef, 0)
```

```
        }
        else {
            temp <- lm(XY$v ~ 1, weights = lm.wei)
            nugget <- temp$coef
            cov.pars <- c(0, 0)
        }
    }
    value <- sum((temp$residuals)^2)
}
else {
    if (messages.screen)
        cat(paste("variofit: minimisation function used:",
            minimisation.function, "\n"))
    umax <- max(vario$u)
    vmax <- max(vario$v)
    if (missing(ini.cov.pars)) {
        ini.cov.pars <- as.matrix(expand.grid(c(vmax/2, 3 *
            vmax/4, vmax), seq(0, 0.8 * umax, len = 6)))
        if (!fix.nugget)
            nugget <- unique(c(nugget, vmax/10, vmax/4, vmax/2))
        if (!fix.kappa)
            kappa <- unique(c(kappa, 0.25, 0.5, 1, 1.5, 2))
        if (messages.screen)
            warning("initial values not provided - running the default search
    }
    else {
        if (any(class(ini.cov.pars) == "eyefit")) {
            init <- nugget <- kappa <- NULL
            for (i in 1:length(ini.cov.pars)) {
              init <- drop(rbind(init, ini.cov.pars[[i]]$cov.pars))
              nugget <- c(nugget, ini.cov.pars[[i]]$nugget)
              if (cov.model == "gneiting.matern")
                kappa <- drop(rbind(kappa, ini.cov.pars[[i]]$kappa))
              else kappa <- c(kappa, ini.cov.pars[[i]]$kappa)
            }
            ini.cov.pars <- init
        }
        if (any(class(ini.cov.pars) == "variomodel")) {
            nugget <- ini.cov.pars$nugget
            kappa <- ini.cov.pars$kappa
            ini.cov.pars <- ini.cov.pars$cov.pars
        }
    }
    if (is.matrix(ini.cov.pars) | is.data.frame(ini.cov.pars)) {
        ini.cov.pars <- as.matrix(ini.cov.pars)
        if (nrow(ini.cov.pars) == 1)
            ini.cov.pars <- as.vector(ini.cov.pars)
```

```
    else {
        if (ncol(ini.cov.pars) != 2)
          stop("\nini.cov.pars must be a matrix or data.frame with 2 compo
    }
}
else if (length(ini.cov.pars) > 2)
    stop("\nini.cov.pars must provide initial values for sigmasq and phi\n
if (is.matrix(ini.cov.pars) | (length(nugget) > 1) |
    (length(kappa) > 1)) {
    if (messages.screen)
        cat("variofit: searching for best initial value ...")
    ini.temp <- matrix(ini.cov.pars, ncol = 2)
    grid.ini <- as.matrix(expand.grid(sigmasq = unique(ini.temp[,
        1]), phi = unique(ini.temp[, 2]), tausq = unique(nugget),
        kappa = unique(kappa)))
    v.loss <- function(parms, u, v, n, cov.model, weights) {
        sigmasq <- parms[1]
        phi <- parms[2]
        if (cov.model == "power")
          phi <- 2 * exp(phi)/(1 + exp(phi))
        tausq <- parms[3]
        kappa <- parms[4]
        if (cov.model == "power")
          v.mod <- tausq + cov.spatial(u, cov.pars = c(sigmasq,
            phi), cov.model = "power", kappa = kappa)
        else v.mod <- (sigmasq + tausq) - cov.spatial(u,
          cov.pars = c(sigmasq, phi), cov.model = cov.model,
          kappa = kappa)
        if (weights == "equal")
          loss <- sum((v - v.mod)^2)
        if (weights == "npairs")
          loss <- sum(n * (v - v.mod)^2)
        if (weights == "cressie")
          loss <- sum((n/(v.mod^2)) * (v - v.mod)^2)
        return(loss)
    }
    grid.loss <- apply(grid.ini, 1, v.loss, u = XY$u,
        v = XY$v, n = XY$n, cov.model = cov.model, weights = weights)
    ini.temp <- grid.ini[which(grid.loss == min(grid.loss))[1],
        , drop = FALSE]
    if (is.R())
        rownames(ini.temp) <- "initial.value"
    if (messages.screen) {
        cat(" selected values:\n")
        print(rbind(round(ini.temp, digits = 2), status = ifelse(c(FALSE,
          FALSE, fix.nugget, fix.kappa), "fix", "est")))
        cat(paste("loss value:", min(grid.loss), "\n"))
```

```
    }
    names(ini.temp) <- NULL
    ini.cov.pars <- ini.temp[1:2]
    nugget <- ini.temp[3]
    kappa <- ini.temp[4]
    grid.ini <- NULL
}
if (ini.cov.pars[1] > 2 * vmax)
    warning("unreasonable initial value for sigmasq (too high)")
if (ini.cov.pars[1] + nugget > 3 * vmax)
    warning("unreasonable initial value for sigmasq + nugget (too high)")
if (vario$output.type != "cloud") {
    if (ini.cov.pars[1] + nugget < 0.3 * vmax)
        warning("unreasonable initial value for sigmasq + nugget (too low)
}
if (nugget > 2 * vmax)
    warning("unreasonable initial value for nugget (too high)")
if (ini.cov.pars[2] > 1.5 * umax)
    warning("unreasonable initial value for phi (too high)")
if (!fix.kappa) {
    if (cov.model == "powered.exponential")
        Tkappa.ini <- log(kappa/(2 - kappa))
    else Tkappa.ini <- log(kappa)
}
if (minimisation.function == "nls") {
    if (ini.cov.pars[2] == 0)
        ini.cov.pars <- max(XY$u)/10
    if (kappa == 0)
        kappa <- 0.5
    if (cov.model == "power")
        Tphi.ini <- log(ini.cov.pars[2]/(2 - ini.cov.pars[2]))
    else Tphi.ini <- log(ini.cov.pars[2])
    XY$cov.model <- cov.model
    if (fix.nugget) {
        XY$nugget <- as.vector(nugget)
        if (fix.kappa) {
          XY$kappa <- as.vector(kappa)
          res <- nls((v - nugget) ~ matrix((1 - cov.spatial(u,
            cov.pars = c(1, exp(Tphi)), cov.model = cov.model,
            kappa = kappa)), ncol = 1), start = list(Tphi = Tphi.ini),
            data = XY, algorithm = "plinear", ...)
        }
        else {
          if (cov.model == "powered.exponential")
            res <- nls((v - nugget) ~ matrix((1 - cov.spatial(u,
              cov.pars = c(1, exp(Tphi)), cov.model = cov.model,
              kappa = (2 * exp(Tkappa)/(1 + exp(Tkappa))))),
```

```
          ncol = 1), start = list(Tphi = Tphi.ini,
          Tkappa = Tkappa.ini), data = XY, algorithm = "plinear",
          ...)
      else res <- nls((v - nugget) ~ matrix((1 -
        cov.spatial(u, cov.pars = c(1, exp(Tphi)),
          cov.model = cov.model, kappa = exp(Tkappa))),
        ncol = 1), start = list(Tphi = Tphi.ini,
        Tkappa = Tkappa.ini), data = XY, algorithm = "plinear",
        ...)
      kappa <- exp(coef(res)["Tkappa"])
      names(kappa) <- NULL
    }
    cov.pars <- coef(res)[c(".lin", "Tphi")]
    names(cov.pars) <- NULL
}
else {
    if (fix.kappa) {
      XY$kappa <- kappa
      res <- nls(v ~ cbind(1, (1 - cov.spatial(u,
        cov.pars = c(1, exp(Tphi)), cov.model = cov.model,
        kappa = kappa))), start = list(Tphi = Tphi.ini),
        algorithm = "plinear", data = XY, ...)
    }
    else {
      if (cov.model == "powered.exponential")
        res <- nls(v ~ cbind(1, (1 - cov.spatial(u,
          cov.pars = c(1, exp(Tphi)), cov.model = cov.model,
          kappa = (2 * exp(Tkappa)/(1 + exp(Tkappa))))))),
          start = list(Tphi = Tphi.ini, Tkappa = Tkappa.ini),
          algorithm = "plinear", data = XY, ...)
      else res <- nls(v ~ cbind(1, (1 - cov.spatial(u,
        cov.pars = c(1, exp(Tphi)), cov.model = cov.model,
        kappa = exp(Tkappa)))), start = list(Tphi = Tphi.ini,
        Tkappa = Tkappa.ini), algorithm = "plinear",
        data = XY, ...)
      kappa <- exp(coef(res)["Tkappa"])
      names(kappa) <- NULL
    }
    nugget <- coef(res)[".lin1"]
    names(nugget) <- NULL
    cov.pars <- coef(res)[c(".lin2", "Tphi")]
    names(cov.pars) <- NULL
}
if (cov.model == "power")
    cov.pars[2] <- 2 * exp(cov.pars[2])/(1 + exp(cov.pars[2]))
else cov.pars[2] <- exp(cov.pars[2])
if (nugget < 0 | cov.pars[1] < 0) {
```

```
            warning("\nvariofit: negative variance parameter found using the c
            temp <- c(sigmasq = cov.pars[1], phi = cov.pars[2],
              tausq = nugget, kappa = kappa)
            print(rbind(round(temp, digits = 4), status = ifelse(c(FALSE,
              FALSE, fix.nugget, fix.kappa), "fix", "est")))
            return(invisible())
        }
        value <- sum(resid(res)^2)
        message <- "nls does not provides convergence message"
    }
    if (minimisation.function == "nlm" | minimisation.function ==
        "optim") {
        .global.list <- list(u = XY$u, v = XY$v, n = XY$n,
            fix.nugget = fix.nugget, nugget = nugget, fix.kappa = fix.kappa,
            kappa = kappa, cov.model = cov.model, m.f = minimisation.function
            weights = weights)
        ini <- ini.cov.pars
        if (cov.model == "power")
            ini[2] <- log(ini[2]/(2 - ini[2]))
        if (cov.model == "linear")
            ini <- ini[1]
        if (fix.nugget == FALSE)
            ini <- c(ini, nugget)
        if (!fix.kappa)
            ini <- c(ini, Tkappa.ini)
        names(ini) <- NULL
        if (minimisation.function == "nlm") {
            result <- nlm(.loss.vario, ini, g.l = .global.list,
              ...)
            result$par <- result$estimate
            result$value <- result$minimum
            result$convergence <- result$code
            if (!is.null(get(".temp.theta", pos = .geoR.env)))
              result$par <- get(".temp.theta", pos = .geoR.env)
        }
        else {
            lower.l <- sapply(limits, function(x) x[1])
            upper.l <- sapply(limits, function(x) x[2])
            if (fix.kappa == FALSE) {
              if (fix.nugget) {
                lower <- lower.l[c("sigmasq.lower", "phi.lower",
                  "kappa.lower")]
                upper <- upper.l[c("sigmasq.upper", "phi.upper",
                  "kappa.upper")]
              }
              else {
                lower <- lower.l[c("sigmasq.lower", "phi.lower",
```

```
          "tausq.rel.lower", "kappa.lower")]
        upper <- upper.l[c("sigmasq.upper", "phi.upper",
          "tausq.rel.upper", "kappa.upper")]
      }
    }
    else {
      if (cov.model == "power") {
        if (fix.nugget) {
          lower <- lower.l[c("sigmasq.lower", "phi.lower")]
          upper <- upper.l[c("sigmasq.upper", "phi.upper")]
        }
        else {
          lower <- lower.l[c("sigmasq.lower", "phi.lower",
            "tausq.rel.lower")]
          upper <- upper.l[c("sigmasq.upper", "phi.upper",
            "tausq.rel.upper")]
        }
      }
      else {
        lower <- lower.l["phi.lower"]
        upper <- upper.l["phi.upper"]
      }
    }
    result <- optim(ini, .loss.vario, method = "L-BFGS-B",
      hessian = TRUE, lower = lower, upper = upper,
      g.l = .global.list, ...)
}
value <- result$value
message <- paste(minimisation.function, "convergence code:",
    result$convergence)
if (cov.model == "linear")
    result$par <- c(result$par[1], 1, result$par[-1])
cov.pars <- as.vector(result$par[1:2])
if (cov.model == "power")
    cov.pars[2] <- 2 * exp(cov.pars[2])/(1 + exp(cov.pars[2]))
if (!fix.kappa) {
    if (fix.nugget)
      kappa <- result$par[3]
    else {
      nugget <- result$par[3]
      kappa <- result$par[4]
    }
    if (.global.list$cov.model == "powered.exponential")
      kappa <- 2 * (exp(kappa))/(1 + exp(kappa))
    else kappa <- exp(kappa)
}
else if (!fix.nugget)
```

```
                nugget <- result$par[3]
            }
        }
        estimation <- list(nugget = nugget, cov.pars = cov.pars,
            cov.model = cov.model, kappa = kappa, value = value,
            trend = vario$trend, beta.ols = vario$beta.ols, practicalRange = practical
                phi = cov.pars[2], kappa = kappa), max.dist = max.dist,
            minimisation.function = minimisation.function)
        estimation$weights <- weights
        if (weights == "equal")
            estimation$method <- "OLS"
        else estimation$method <- "WLS"
        estimation$fix.nugget <- fix.nugget
        estimation$fix.kappa <- fix.kappa
        estimation$lambda <- vario$lambda
        estimation$message <- message
        estimation$call <- call.fc
        oldClass(estimation) <- c("variomodel", "variofit")
        return(estimation)
}
<bytecode: 0x7fe7580bc0b0>
<environment: namespace:geoR>
```

```
> #####le resultat du krigeage
>
> Zkrige = matrix(K$predict,nrow=length(grx),ncol=length(gry),byrow=F)
> titre = paste("Krigeage avec un modele",c.m)
> image.plot(grx,gry,Zkrige,zlim=c(-5,5),main=titre)
> contour(grx,gry,Zkrige,levels=seq(-5,5),add=TRUE)
> points(x,y,pch=19)
>
```

Avec le variogramme optimal choisi (supposons que c'est le modèle exponentiel ci-dessus), on réalise une prédiction du processus spatial en plusieurs points $s_0$ dans une grille contenant l'échantillon ou pas (le krigeage donne une interpolation exacte ; autrement dit $\hat{Z}_{s_0} = Z_{s_0}$ si $s_0$ est un site de l'échantillon).
La variance de l'erreur de prédiction donne la qualité de l'interpolation, plus elle est proche de 0 meilleure est la qualité de prédiction.

```
> #####variance de l'erreur de prévision
>
> s=apply(cbind(K$krige.var,rep(0,length(K$krige.var))),1,max)
> Sigma = sqrt(matrix(s,nrow=length(grx),ncol=length(gry),
+                     byrow=F))
> titre = "Ecart-type de krigeage"
> image.plot(grx,gry,Sigma,zlim=c(0,2),main=titre)
> contour(grx,gry,Sigma,levels=seq(0,2,0.5),add=TRUE)
> points(x,y,pch=19)
>
```

```
>
```

(c) Automatisation du krigeage avec Automap

```
> ### choix du variogramme automatisé
> donnees = read.table("jeu1.dat")
> grx = seq(0,100)
> gry = seq(0,100)
> grille = expand.grid(grx,gry)# l'ensemble S
> colnames(donnees)=c("x","y","z")
> coordinates(donnees) = ~x+y
> # Ici le modèle optimal est de Sphérique de paramètres
> #de portee 40.20053, portee  1.5887879 et pepite
>
> variogram = autofitVariogram(z~1,donnees)
> #variogram
> plot(variogram)
> ###Krigeage avec choix automatique du variogramme
> grille2=as.data.frame(grille)
> colnames(grille2)=c("x","y")
> coords=SpatialPoints(grille2)
> kriging_result = autoKrige(z~1,donnees, coords)

[using ordinary kriging]

> plot(kriging_result)
> #Sans grille
>
> kriging_result = autoKrige(z~1,donnees)

[using ordinary kriging]

> plot(kriging_result)
```

La prédiction est très bonne (en particulier autour des points de l'échantillon), au regard de la carte des erreurs de prédiction.

2. **Etude de cas : pollution de l'air**

On souhaite réaliser une carte quotidienne de concentration d'ozone sur la région Parisienne. Pour cela on dispose chaque jour des sorties d'un modèle déterministe mis au point au Laboratoire de Météorologie Dynamique (Ecole Polytechnique) et des mesures de concentration d'ozone effectuées par AirParif en 21 stations.

Le fichier *stationsKm4.txt* en format ascii contient un tableau formé des colonnes suivantes :

- colonne 1 : absisses (en km) des stations
- colonne 2 : ordonnées (en km) des stations
- colonne 3 : mesures aux stations (en $\mu/m^3$)
- colonne 4 : valeur du modèle aux stations

Le fichier *grilleKm4.txt* contient

- colonne 1 : absisses (en km) des points de grille
- colonne 2 : ordonnées (en km) des points de grille
- colonne 3 : valeur du modèle des points de grille

(a) Tracer la carte des concentrations données par le modèle.

```
> summary(donnees)
       V1              V2              V3             V4
 Min.   : 31.17   Min.   : 31.68   Min.   :116.0   Min.   : 98.58
 1st Qu.: 75.25   1st Qu.: 75.76   1st Qu.:130.0   1st Qu.:109.44
 Median : 80.38   Median : 84.34   Median :140.0   Median :118.62
 Mean   : 79.54   Mean   : 81.11   Mean   :148.9   Mean   :128.01
 3rd Qu.: 88.93   3rd Qu.: 89.98   3rd Qu.:157.0   3rd Qu.:131.35
 Max.   :133.30   Max.   :107.17   Max.   :219.0   Max.   :209.67
> x = donnees[,1]
> y = donnees[,2]
> z = donnees[,3]
> #hist(z)
> #-----------
> # modele
> #-------
> #View(grille)
> grxy = grille[,1:2]
> grx = seq(5,130,5)
> gry = seq(5,130,5)
> Zmod = matrix(grille[,3],nrow=length(grx),
+               ncol=length(gry),byrow=FALSE)
> titre = "Modele"
> image.plot(grx,gry,Zmod,zlim=c(100,250),main=titre,
+            col=tim.colors(64))
> contour(grx,gry,Zmod,levels=seq(100,250,50),add=TRUE)
> points(x,y,pch=19)
>
>
```

(b) Faire une carte en estimant la concentration en chaque point de la grille par krigeage à partir des mesures aux 21 stations. Comparer avec la carte précédente.

```
> #------------
> # variogramme
> #------------
> # 1. Variogramme empirique
> #-------------------------
> geodata = as.geodata(donnees)
> m.d = 100                      # distance maximale
> interv = seq(10,m.d,by=20)     # intervalles
> p.m = 2                        # nombre minimal de couples de stations
> ### nuée variographique
> #vario.c = variog(geodata,op="cloud")
> #plot(vario.c,main = "",pch='+')
>
> vario.b = variog(geodata,max.dist=m.d,pairs.min=p.m,
+                  breaks=interv)
```

```
variog: computing omnidirectional variogram
> plot(vario.b,main = "Variogramme empirique")
> # 2. Ajustement du variogramme
> #----------------------------
>
> c.m = "gaussian"
> i.c = c(1000,50)
> varioest = variofit(vario.b,cov.model = c.m,ini.cov.pars=i.c)
variofit: covariance model used is gaussian
variofit: weights used: npairs
variofit: minimisation function used: optim
> x11()
> titre = paste("modele ",c.m,", portee =",round(varioest$cov.pars[2],2),
+               ", palier =",round(varioest$cov.pars[1],2))
> plot(vario.b,main=titre)
> lines(varioest)
>
```

On peut également autiomatiser :

```
> # Avec choix automatique du variogramme
> ###Krigeage avec choix automatique du variogramme
> donnees2=as.data.frame(donnees[,1:3])
> colnames(donnees2)=c("x","y","z")
> grillexy=as.data.frame(grxy)
> colnames(grillexy)=c("x","y")
> coords=SpatialPoints(grillexy)
> coordinates(donnees2) = ~x+y
> kriging_result = autoKrige(z~1,donnees2, coords)
[using ordinary kriging]
> x11()
> plot(kriging_result)
> #ou
> Zkrige = matrix(kriging_result$krige_output$var1.pred,
+ nrow=length(grx),ncol=length(gry),byrow=FALSE)
> image.plot(grx,gry,Zkrige,zlim=c(100,300),main=titre,
+            col =tim.colors(64))
> titre = "Krigeage ordinaire"
> s=apply(cbind(kriging_result$krige_output$var1.var,
+ rep(0,length(kriging_result$krige_output$var1.var))),1,max)
> Sigma = sqrt(matrix(s,nrow=length(grx),ncol=length(gry),byrow=FALSE))
> titre = "Ecart-type de krigeage"
> image.plot(grx,gry,Sigma,zlim=c(0,ceiling(max(Sigma))),
+            main=titre,col =tim.colors(64))
> points(x,y,pch=19)
```

(c) On désire combiner les 2 approches. Pour cela on corrige le modèle déterministe en chaque point de la grille par une estimation de la différence concentration-modèle obte-

nue en krigeant les différences observation-modèle aux stations.
Comparer aux deux cartes précédentes.

```
> K=krige.conv(geodata,loc=grxy,krige=
+ krige.control(type.krige="ok",obj.model=varioest))

krige.conv: model with constant mean
krige.conv: Kriging performed using global neighbourhood

> Zkrige = matrix(K$predict,nrow=length(grx),ncol=length(gry),
+                 byrow=FALSE)
> titre = paste("Krigeage avec un modele",c.m)
> image.plot(grx,gry,Zkrige,zlim=c(100,250),main=titre)
> contour(grx,gry,Zkrige,levels=seq(100,250,50),add=TRUE)
> points(x,y,pch=19)
> s=apply(cbind(K$krige.var,rep(0,length(K$krige.var))),1,max)
> Sigma = sqrt(matrix(s,nrow=length(grx),ncol=length(gry),byrow=FALSE))
> titre = "Ecart-type de krigeage"
> image.plot(grx,gry,Sigma,zlim=c(0,ceiling(max(Sigma))),
+ main=titre,col =tim.colors(64))
> contour(grx,gry,Sigma,levels=seq(0,ceiling(max(Sigma)),5),
+         add=TRUE)
> points(x,y,pch=19)
> #### A t-on raison de faire une interpolation ordinaire
> summary(lm(z~x+y))

Call:
lm(formula = z ~ x + y)

Residuals:
    Min      1Q  Median      3Q     Max
-17.982  -9.323  -2.564   4.059  28.920

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 198.6574    19.7081  10.080 7.90e-09 ***
x            -0.9526     0.1238  -7.696 4.24e-07 ***
y             0.3207     0.1723   1.861   0.0791 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 13.24 on 18 degrees of freedom
Multiple R-squared:  0.8162,        Adjusted R-squared:  0.7958
F-statistic: 39.98 on 2 and 18 DF,  p-value: 2.389e-07

>
```

Au regard de la variance de l'erreur, on voit que la prédiction par krigeage ordinaire est très bonne en particulier aux alentours des points de l'échantillon.
Cependant, regardons si la moyenne est constante ou une fonction linéaire des sites ("summary(lm(z x+y))"), si le modèle linéaire est validé alors il faut un krieage universel.

On a bien une tendance (moyenne non constante), un krigeage universel est plus adapté que l'ordinaire.

```
> zerr=z-donnees[,4]
> geodata = as.geodata(cbind(x,y,zerr))
> m.d = 100                      # distance maximale
> interv = seq(10,m.d,by=20)    # intervalles
> p.m = 2                      # nombre minimal de paire
> vario.b = variog(geodata,max.dist=m.d,pairs.min=p.m,
+ uvec=interv,messages.screen=FALSE)

variog: computing omnidirectional variogram

> plot(vario.b,main = "Variogramme empirique")
> # 2. Ajustement du variogramme
> #----------------------------
> c.m = "exponential"
> i.c = c(500,50)
> varioest = variofit(vario.b,cov.model = c.m,
+ minimisation.function = "nls",ini.cov.pars=i.c,fix.nugget=T,
+ fix.kappa=TRUE,max.dist=vario.b$max.dist)

variofit: covariance model used is exponential
variofit: weights used: npairs
variofit: minimisation function used: optim

> titre = paste("modele ",c.m,", portee =",
+ round(varioest$cov.pars[2]*100)/100,", palier =",
+ round(varioest$cov.pars[1]*100)/100)
> plot(vario.b,main=titre)
> lines(varioest)
> # 3. Krigeage
> #------------
> grxy = grille[,1:2]
> grx = seq(5,130,5)
> gry = seq(5,130,5)
> K = krige.conv(geodata,loc=grxy,
+ krige=krige.control(type.krige="ok",obj.model=varioest))

krige.conv: model with constant mean
krige.conv: Kriging performed using global neighbourhood

> ## prediction combinée=krigeage de l'erreur+modele deterministe
>
> Zkrige = matrix(K$predict+grille[,3],nrow=length(grx),
+                 ncol=length(gry),byrow=FALSE)
> titre = paste("Krigeage avec un modele",c.m)
> x11()
> titre = paste("Krigeage avec un modele",c.m)
> image.plot(grx,gry,Zkrige,zlim=c(0,250),main=titre,
+            col=tim.colors(64))
> contour(grx,gry,Zkrige,levels=seq(0,250,10),add=TRUE)
```

```
> points(x,y,pch=19)
> s=apply(cbind(K$krige.var,rep(0,length(K$krige.var))),1,max)
> Sigma = sqrt(matrix(s,nrow=length(grx),ncol=length(gry),byrow=FALSE))
> titre = "Ecart-type de krigeage"
> image.plot(grx,gry,Sigma,zlim=c(0,ceiling(max(Sigma))),
+            main=titre,col =tim.colors(64))
> #contour(grx,gry,Sigma,levels=seq(0,ceiling(max(Sigma))),add=TRUE)
> points(x,y,pch=19)
>
```

En automatisant :

```
> # Choix automatique
>
> donnees3=donnees2
> donnees3$z=zerr
> kriging_result = autoKrige(z~1,donnees3, coords)
[using ordinary kriging]
> plot(kriging_result)
> Zkrige = matrix(kriging_result$krige_output$var1.pred+
+ grille[,3],nrow=length(grx),
+ ncol=length(gry),byrow=FALSE)
> image.plot(grx,gry,Zkrige,zlim=c(100,250),main=titre,
+            col =tim.colors(64))
> points(x,y,pch=19)
> s=apply(cbind(kriging_result$krige_output$var1.var,
+ rep(0,length(kriging_result$krige_output$var1.var))),1,max)
> Sigma = sqrt(matrix(s,nrow=length(grx),ncol=length(gry),
+                     byrow=FALSE))
> titre = "Ecart-type de krigeage"
> image.plot(grx,gry,Sigma,zlim=c(0,ceiling(max(Sigma))),
+            main=titre,col =tim.colors(64))
> points(x,y,pch=19)
```

La méthode combinée donne une meilleure prédiction que le modèle par krigeage avec les données aux stations ou le modèle déterministe.