```r
#construire des données pour la régression logistique quadratique
log_quad <- function(X) {
    X2 <- X

    for (p in 1:(dim(X)[2]-1))
        for (q in (p+1):dim(X)[2])
            X2 <- cbind(X2, X[,p]*X[,q])

    for (p in 1:dim(X)[2])
        X2 <- cbind(X2, X[,p]^2)

    return(X2)
}

# calculer des paramètres de modèle
log_quad.app <- function(Xapp, zapp, intr, epsi) {
  Xapp <- log_quad(Xapp)

    n <- dim(Xapp)[1] # nombre d'individu
    p <- dim(Xapp)[2] # nombre de variable

    Xapp <- as.matrix(Xapp)

    if (intr == T) { # on ajoute une ordonnée à l'origine {
        Xapp <- cbind(rep(1,n),Xapp)
        p <- p + 1
    }

    targ <- matrix(as.numeric(zapp),nrow=n) # ti: la réalisation d'une variable Ti~B(pi)
    targ[which(targ==2),] <- 0 # remplacer la classe 2 à 0
    tXap <- t(Xapp)

    beta <- matrix(0,nrow=p,ncol=1) # w: le paramètre pour calculer
    rownames(beta) <- colnames(Xapp)

    conv <- F
    iter <- 0
    while (conv == F) {
        iter <- iter + 1
        bold <- beta

        prob <- postprob.logquad.explicit(beta, Xapp) # la valeur de pi
        MatW <- diag(as.numeric(prob * (1 - prob)))    # W: la matrice diagonale de terme Wii = pi(1-pi)

        beta <- bold - (solve(-(tXap%*%MatW%*%Xapp)))%*%tXap%*%(targ-prob) # l'équation de nouvelle esimation

        #diagMatW = diag(MatW)
        #nanInMatW = diagMatW[which(is.nan(diagMatW))]
        #if (length(nanInMatW) > 0)
        #    print("Error is comming...")

        nanInProb = prob[which(is.nan(prob))]
        if (length(nanInProb) > 0)
            print("Error is comming...")

        #print("Xapp :")
```

```
        #print(Xapp)
        #print("beta :")
        #print(beta)
        #print("beta old :")
        #print(bold)
        #print("prob : ")
        #print(head(prob))
        #print("MatW")
        #print(head(MatW[,1:5]))
        #print("diag(MatW)")
        #print(diag(MatW))
        #a = diag(MatW)
        #a = a[which(is.nan(a))]
        #print(a)
        #print("targ-prob")
        #print(head(targ-prob))
        #print("tXap")
        #print(head(tXap[,1:5]))
        #print("tXap%*%(targ-prob)")
        #print(head(tXap%*%(targ-prob)))

        #print("-(tXap%*%MatW%*%Xapp)")
        #print(-(tXap%*%MatW%*%Xapp))
        #print("det de -(tXap%*%MatW%*%Xapp)")
        #print(det(-(tXap%*%MatW%*%Xapp)))

        #print("-----------------------------------")

        if (norm(beta-bold)<epsi) {
            conv <- T
        }
    }

    prob <- postprob(beta, Xapp)
    out <- NULL
    out$beta <- beta
    out$iter <- iter
    out$logL <- sum(targ*prob+(1-targ)*(1-prob))
    out$X <- Xapp
    out$z <- targ

    out
}

log_quad.val <- function(beta, Xtst) {
    Xtst <- log_quad(Xtst)
    m <- dim(Xtst)[1] # nombre d'individu des données de test
    p <- dim(beta)[1]
    pX <- dim(Xtst)[2] # nombre de variable des données de test

    Xtst <- as.matrix(Xtst)

    if (pX == (p-1)) {
        Xtst  <- cbind(rep(1,m),Xtst)
    }

    prob_w1 <- postprob(beta, Xtst) # proportions à postériori de la classe 1
    prob_w2 <- 1 - prob_w1
```

```r
    prob <- cbind(prob_w1, prob_w2) # les probabilités a posteriori
    pred <- max.col(prob) # trouver la position de la valeur maximale pour chaque ligne de matrice

    out <- NULL
    out$prob <- prob
    out$pred <- pred

    return(out)
}

postprob.logquad <- function(beta, X) {
    X <- as.matrix(X)
    num = exp(t(beta)%*%t(X))
    den = (1+exp(t(beta)%*%t(X)))
    prob <- t( num / den )
    nb_nan_probs = 0
    for (i in 1:length(prob)) {
        if (is.nan(prob[i])) {
            prob[i] = 0.99
            nb_nan_probs = nb_nan_probs + 1
        }
    }
    pourcentage_proba_indetermine = nb_nan_probs / length(prob) * 100
    if (pourcentage_proba_indetermine > 0)
        cat("Il y a eu", pourcentage_proba_indetermine, "% de probabilités indéterminées.\n")
    return(prob)
}

postprob.logquad.explicit <- function(beta, X) {
    X <- as.matrix(X)
    num = exp(t(beta)%*%t(X))
    den = (1+exp(t(beta)%*%t(X)))
    prob <- t( num / den )

    nanInProb = prob[which(is.nan(prob))]
    if (length(nanInProb) > 0)
        print("nanInProb => Error is comming...")

    for (i in 1:dim(num)[2]) {
        prob_i = num[1,i] / den[1,i]
        if (is.nan(prob_i))
            cat(num[1,i], " / ", den[1,i], " is NaN\n")
    }
    return(prob)
}
```