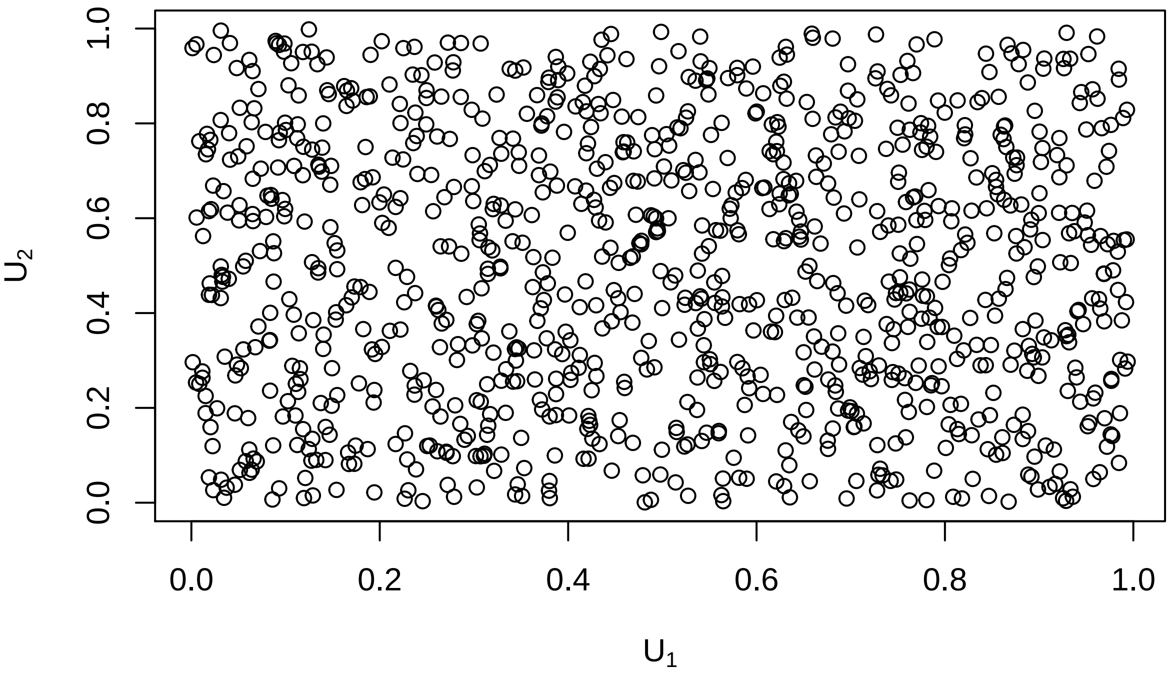
Simulation et fonction de répartition des copules

Voici un document qui inclus des codes R pour simuler différentes copules et des fonctions de répartition.

Copule d'indépendance

Simulation

```r
nsim <- 1000
vV <- matrix(runif(2 * nsim), nsim, 2, byrow = T)
plot(vV, xlab = expression(U[1]), ylab = expression(U[2]))
```



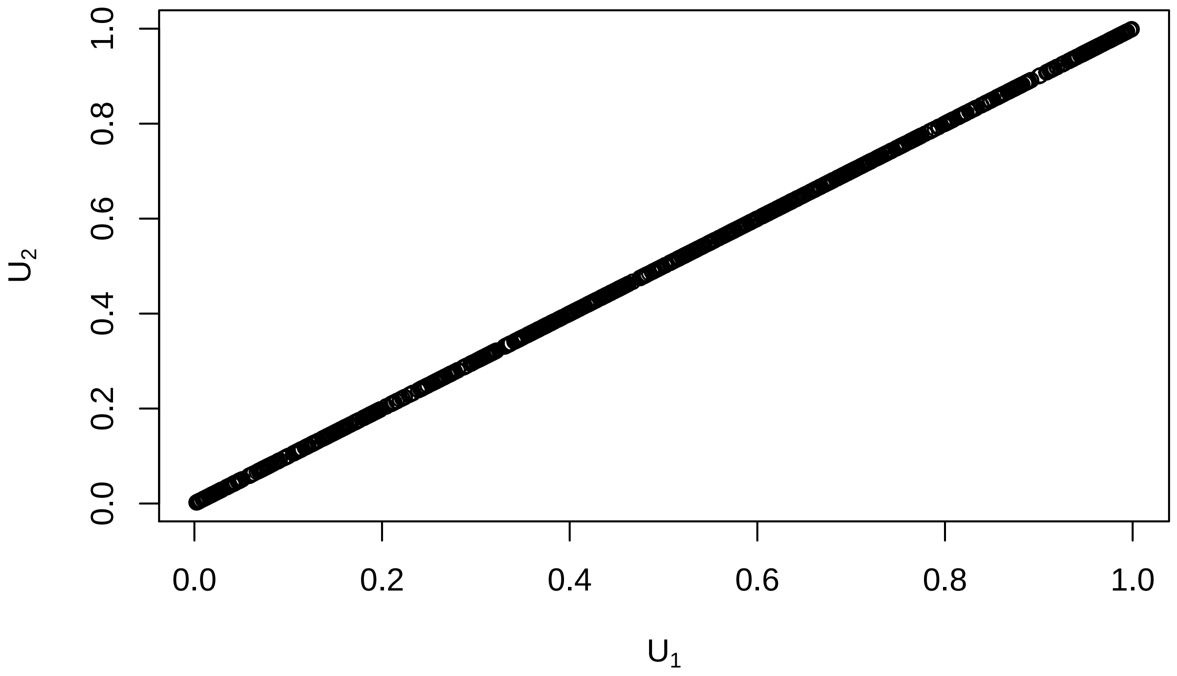
Fonction de répartition

```r
C <- function(u1, u2) u1 * u2
```

Copule borne supérieure de Fréchet

Simulation

```r
vV <- matrix(runif(nsim), nsim, 2)
plot(vV, xlab = expression(U[1]), ylab = expression(U[2]))
```



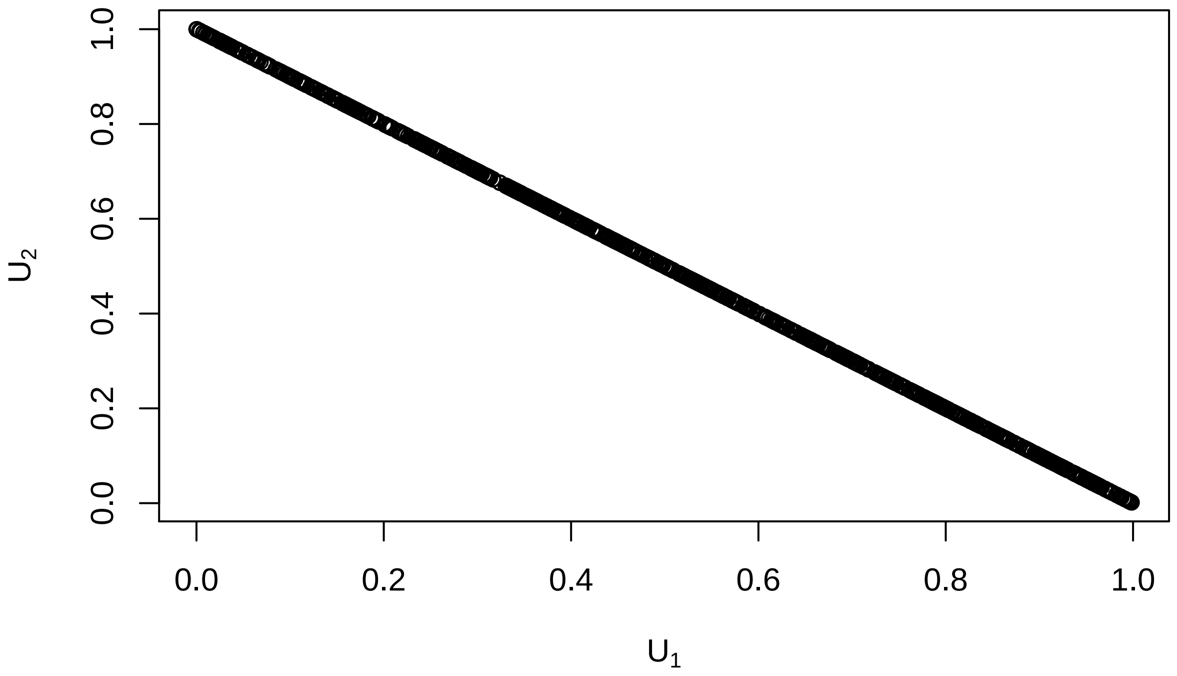
Fonction de répartition

```
C <- function(u1, u2) min(u1, u2)
```

Copule borne inférieure de Fréchet

Simulation

```
vV <- matrix(runif(nsim), nsim, 2)
vV[, 2] <- 1 - vV[,1]
plot(vV, xlab = expression(U[1]), ylab = expression(U[2]))
```



Fonction de répartition

```
C <- function(u1, u2) max(u1 + u2 - 1, 0)
```

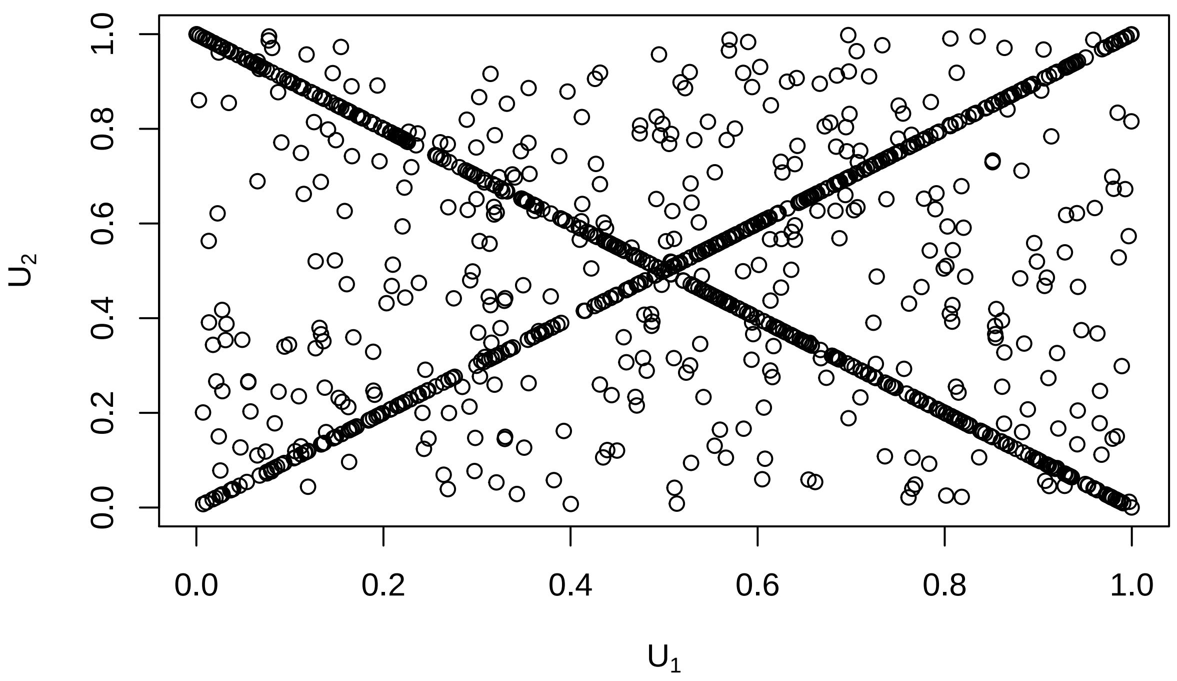
Copule de Fréchet

Simulation

```
alphabeta <- c(.3, .3, 1-.3-.3)
vSup <- matrix(runif(nsim), nsim, 2)
vInf <- matrix(runif(nsim), nsim, 2)
vInf[,2]<-1 - vInf[,1]
vInd <- matrix(runif(2 * nsim), nsim, 2, byrow = T)
fb <- sample(c(1, 2, 3), 1000, replace = TRUE, prob = alphabeta)
vV <- matrix(numeric(), nsim, 2)

for(i in 1:nsim) {
  if (fb[i] == 1)
    vV[i, ] <- vInf[i, ]
  else if (fb[i] == 2)
    vV[i, ] <- vSup[i,]
  else vV[i, ] <- vInd[i, ]
}

plot(vV, xlab = expression(U[1]), ylab = expression(U[2]))
```



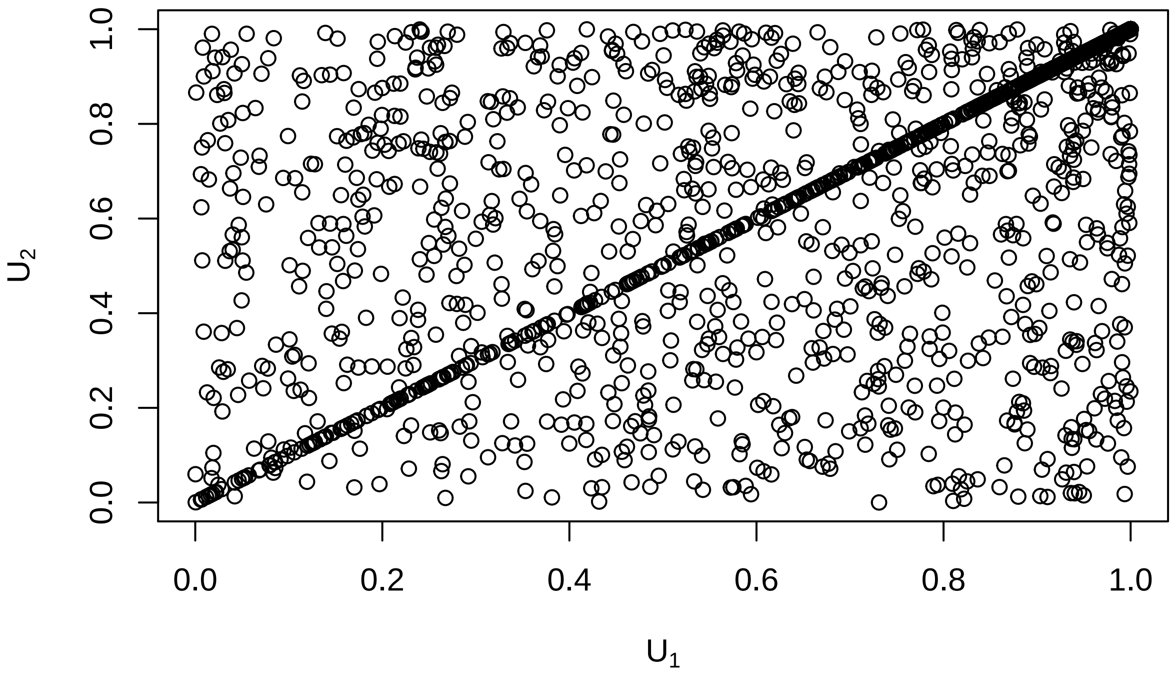
Fonction de répartition

```
C <- function(u1, u2, alph, bet) (u1 * u2)*(1 - alph - bet) + alph * min(u1, u2) + bet * max(u1 + u2 -
```

Copule Cuadras-Augé

Simulation

```
alph <- 0.2
vV <- matrix(runif(3 * nsim), nsim, 3, byrow = T)
vY <- cbind(qexp(vV[ ,1], alph), sapply(1:2, function(t) qexp(vV[ ,t + 1], 1 - alph)))
vU <- sapply(1:2, function(t) 1-exp(-pmin(vY[ ,c(1, t + 1)])))
plot(vU, xlab = expression(U[1]), ylab = expression(U[2]))
```



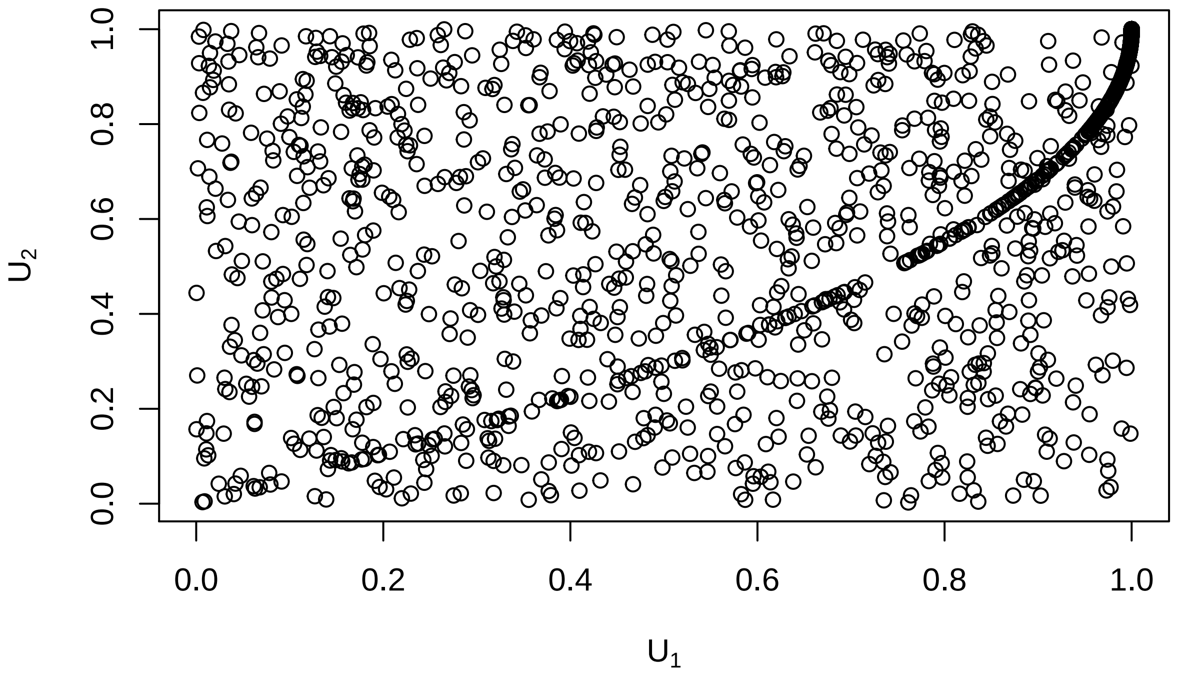
Fonction de répartition

```
C <- function(u1, u2) (u1 * u2) ** (1 - alph) * min(u1, u2) ** alph
```

Copule Marshall-Olkin

Simulation

```
alphabeta <- c(0.05, 0.1)
vV <- matrix(runif(3 * nsim), nsim, 3, byrow = T)
vY <- cbind(qexp(vV[ ,1], 1), sapply(1:2, function(t) qexp(vV[ ,t + 1], ((1 - alphabeta[t]) / alphabeta
vX <- sapply(1:2, function(t) pmin(vY[ ,1], vY[ ,(t + 1)]))
vU <- sapply(1:2, function(t) 1 - exp(-pmin(vY[ ,c(1, (t + 1))]) / alphabeta[t]))
plot(vU, xlab = expression(U[1]), ylab = expression(U[2]))
```



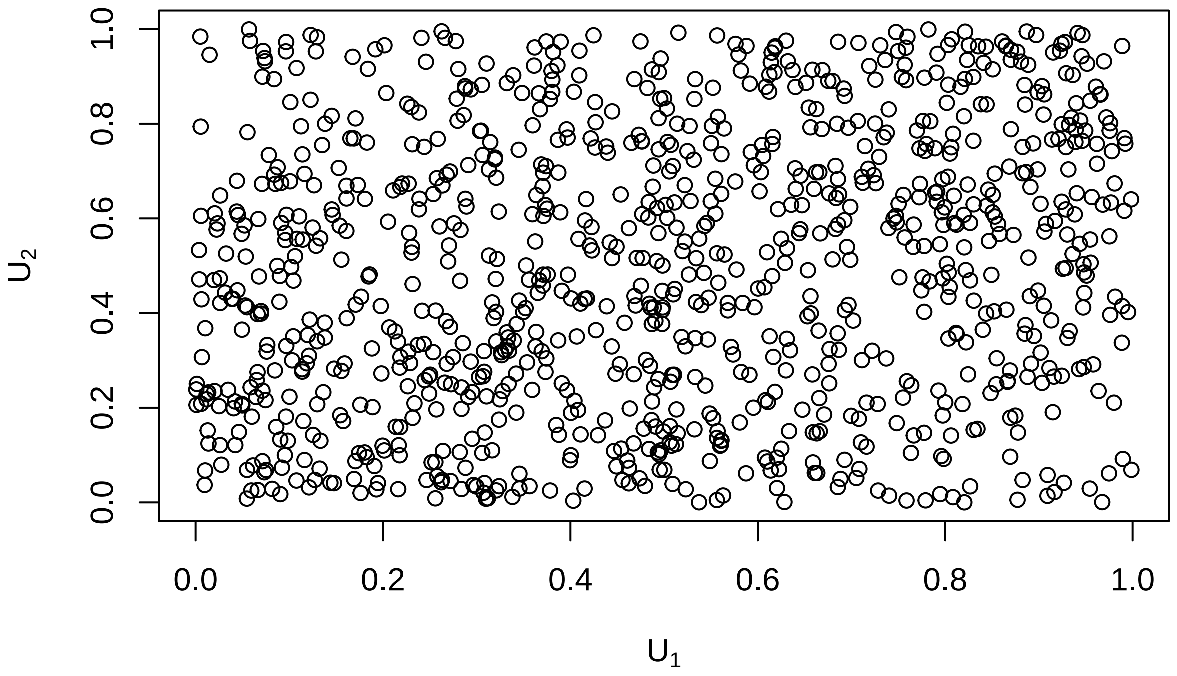
Fonction de répartition

```r
C <- function(u1, u2, alph, bet) min(u1 ** (1 - alph) * u2, u1 * u2 ** (1 - bet))
```

Copule EFGM

Simulation

```r
alph <- 0.5
vV <- matrix(runif(nsim * 2), nsim, 2, byrow = T)
W1 <- alph * (2 * vV[ ,1] - 1) - 1
W2 <- (1 - alph * (2 * vV[ ,1] - 1)) ** 2 + 4 * alph * vV[ ,2] * (2 * vV[ ,1] - 1)
vU <- cbind(vV[ ,1], 2 * vV[ ,2] / (sqrt(W2) - W1))
plot(vU, xlab = expression(U[1]), ylab = expression(U[2]))
```



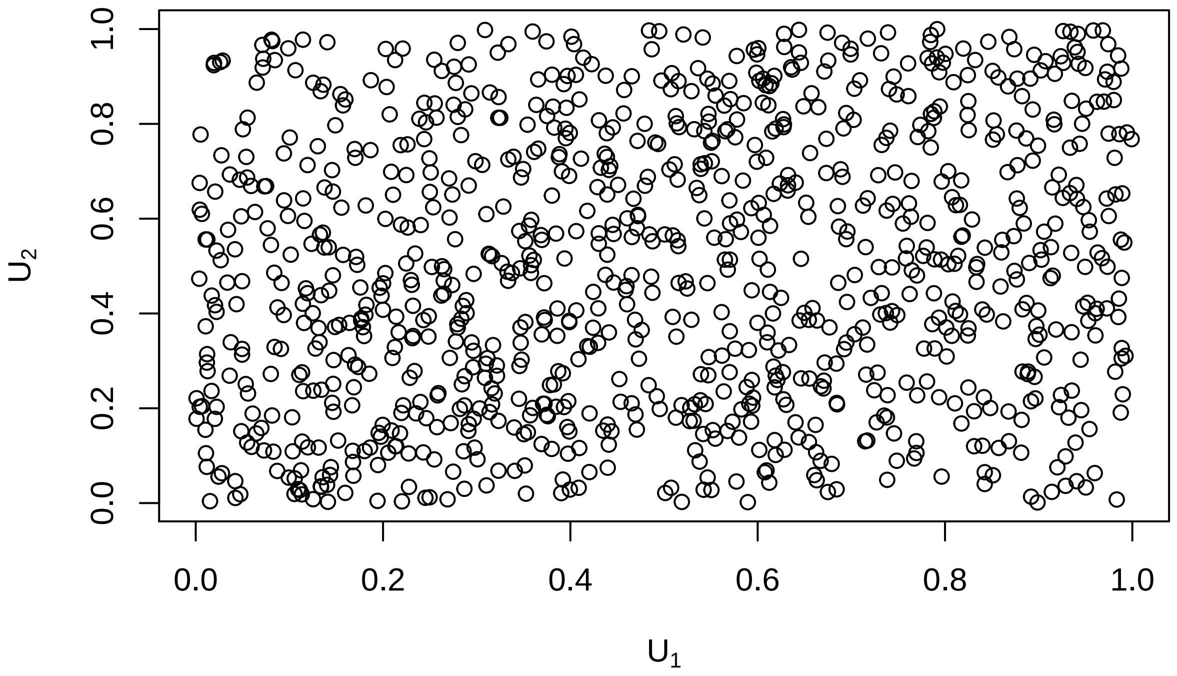
Fonction de répartition

```r
C <- function(u1, u2, alph) u1 * u2 + alph * (1 - u1) * (1 - u2) * u1 * u2
```

Copule Ali-Mikhail-Haq

Simulation

```r
alph <- 0.5
fg2 <- c(0, sapply(1:1000, function(t) alph * (1 - alph) ** (t - 1)))
vTheta <- sample(0:1000, nsim, replace = TRUE, prob = fg2)
vV <- matrix(runif(2 * nsim), nsim, 2)
vW <- sapply(1:2, function(t) -1 / vTheta * log(1 - vV[ ,t]))
vU <- sapply(1:2, function(t) (1 - alph) / (exp(vW[ ,t]) - alph))
plot(vU, xlab = expression(U[1]), ylab = expression(U[2]))
```



Fonction de répartition

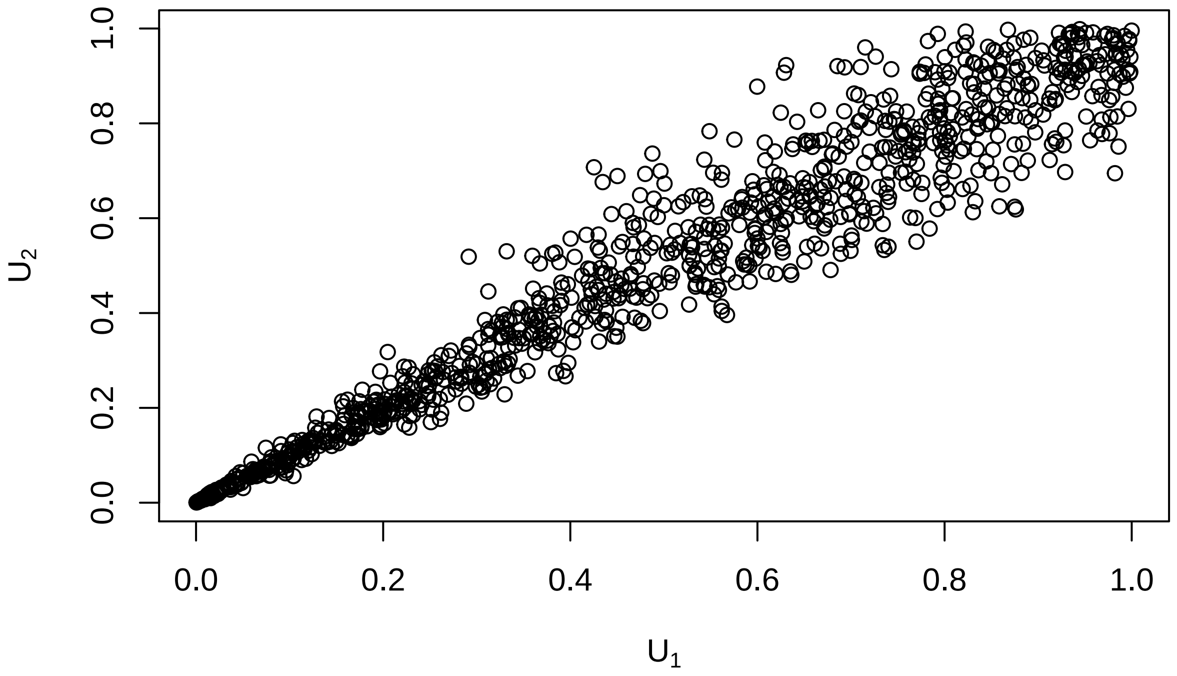
```
C <- function(u1, u2, alph) u1 * u2 / (1 - alph * (1 - u1) * (1 - u2))
```

Copule de Clayton

Simulation

```
alph <- 10

vV <- matrix(runif(nsim * 3), nsim, 3, byrow = T)
vTheta <- qgamma(vV[, 1], 1 / alph, 1)
vY <- sapply(1:2, function(t) qexp(vV[, t + 1], vTheta))
vU <- (1 + vY) ** (-1 / alph)
plot(vU, xlab = expression(U[1]), ylab = expression(U[2]))
```



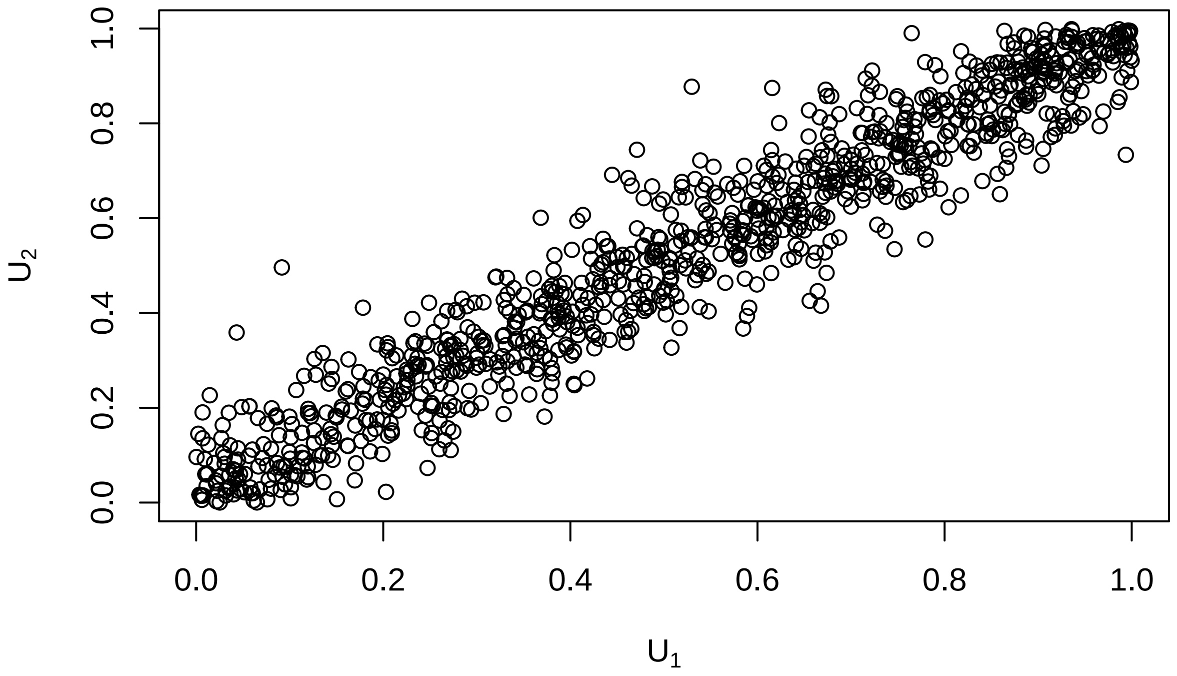
Fonction de répartition

```
C <- function(u1, u2, alph) (u1 ** -alph + u2 ** -alph - 1)** -(1 / alph)
```

Copule de Frank

Simulation

```
alph <- 20
vV <- matrix(runif(nsim * 2), nsim, 2)
vU <- cbind(vV[ ,1], -1 / alph * log(1 + vV[ ,2] * (exp(-alph) - 1) / (exp(-alph * vV[ ,1]) * (1 - vV[
plot(vU, xlab = expression(U[1]), ylab = expression(U[2]))
```



Fonction de répartition

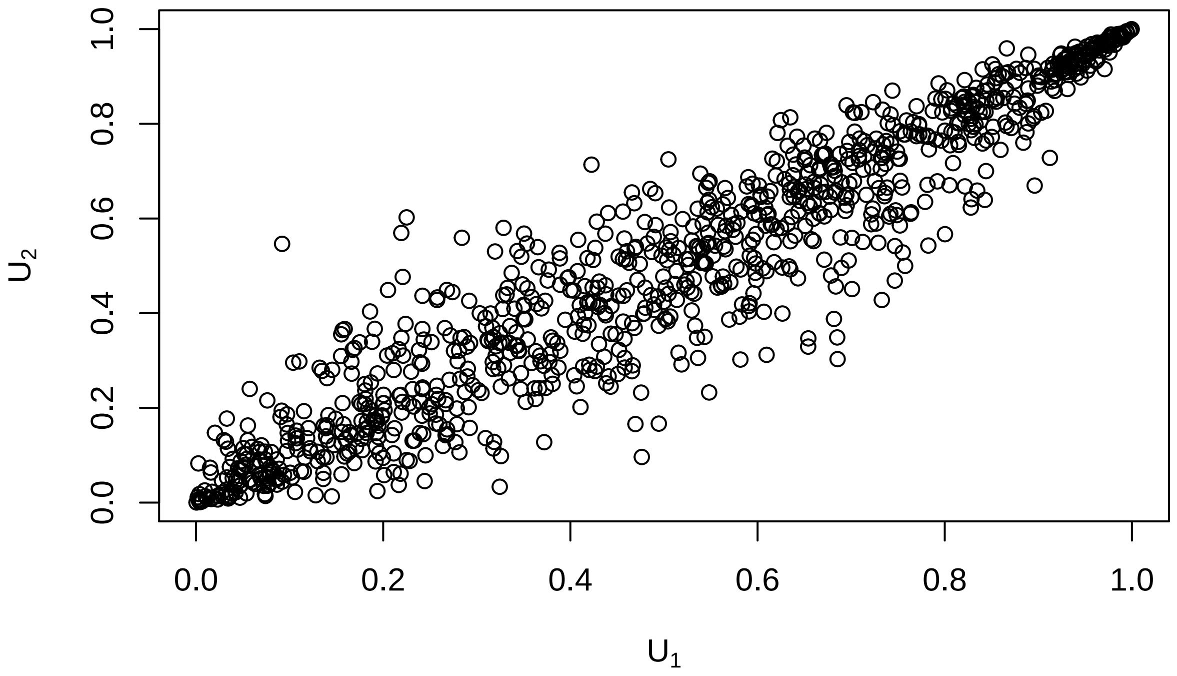
```
C <- function(u1, u2, alph) -1 / alph * log(1 + (exp(-alph * u1) - 1) * (exp(-alph * u2) - 1) / (exp(-al
```

Copule de Gumbel

Simulation

```
rstable <- function(n, phi){
  # algo de Marceau (2013) page 351
  R <- runif(n, min = - pi / 2, max = pi / 2)
  V <- rexp(n, 1)
  A <- sin(phi * (R + pi / 2)) / cos(R) ** (1 / phi)
  B <- (cos(phi * pi / 2 + (phi - 1) * R) / V) ** ((1 - phi) / phi)
  A * B
}

alph <- 5
vThet <- rstable(nsim, phi = alph ** -1)
vV<-matrix(runif(nsim * 2), nsim, 2)
vZ <- qexp(vV, 1)
vU <- exp(- (vZ / vThet) ** (1 / alph))
plot(vU, xlab = expression(U[1]), ylab = expression(U[2]))
```



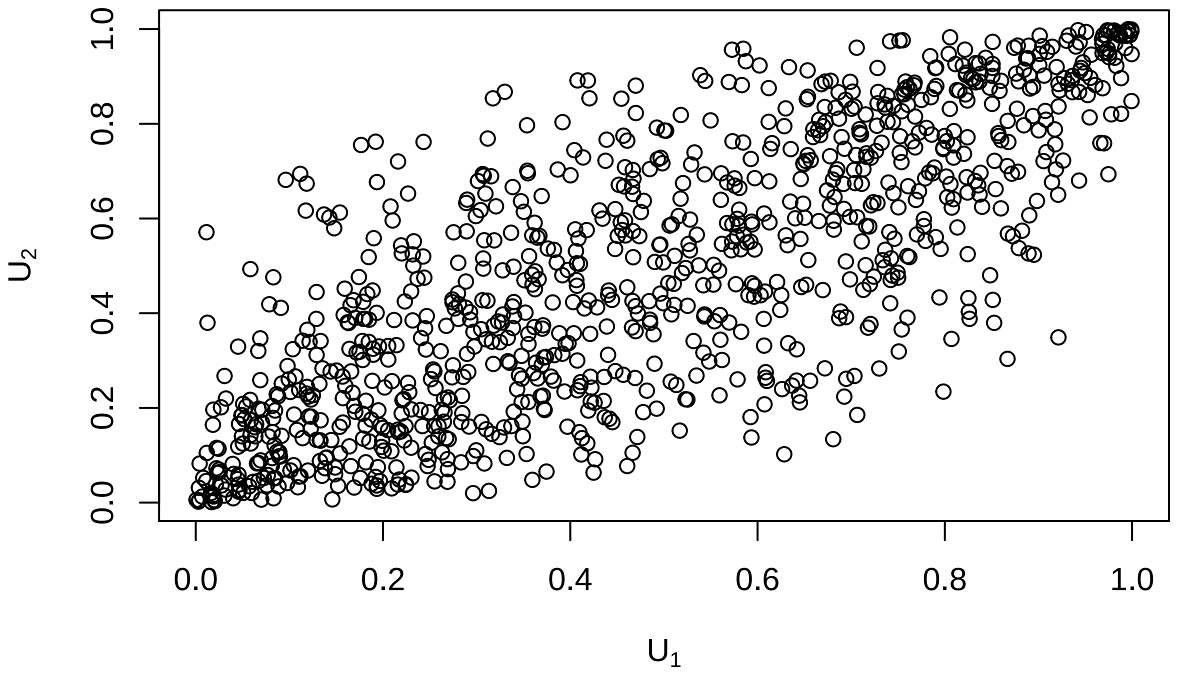
Fonction de répartition

```
C <- function(u1, u2, alph) exp(-((-log(u1) ** alph) + (-log(u2)) ** alph) ** (1 / alph))
```

Copule normale

Simulation

```
alph<-.8
vV<-MASS::mvrnorm(nsim, c(0, 0), matrix(c(1, alph, alph, 1), 2))
vU<-sapply(1:2, function(t) pnorm(vV[ ,t], 0, 1))
plot(vU, xlab = expression(U[1]), ylab = expression(U[2]))
```


U_2
U_1
0.0
0.2
0.4
0.6
0.8
1.0