ANEXOS

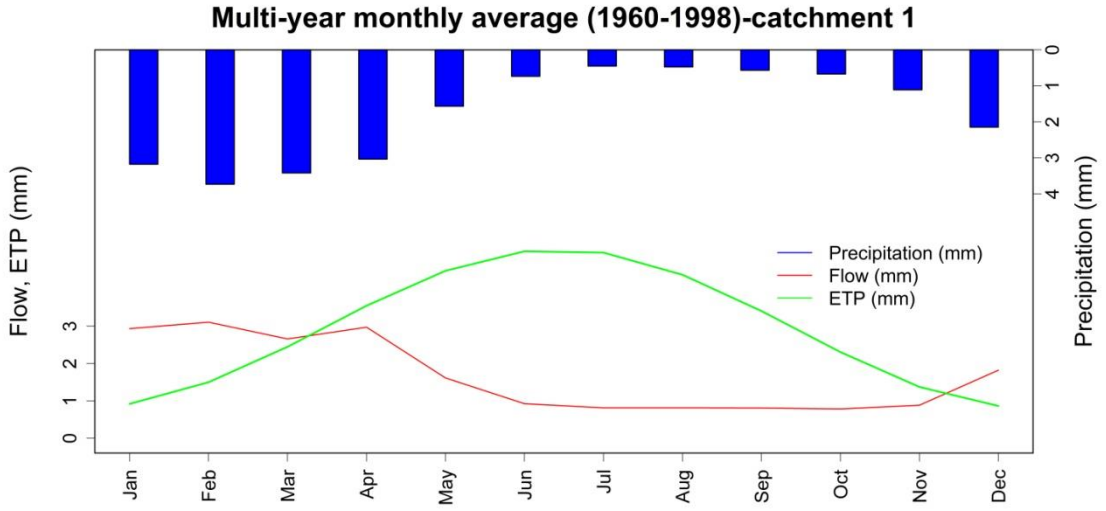

Figura 1. Mensual multianual de precipitación, Flow y ETP de la cuenca 1, para todo el periodo de estudio (1960 a 1998).

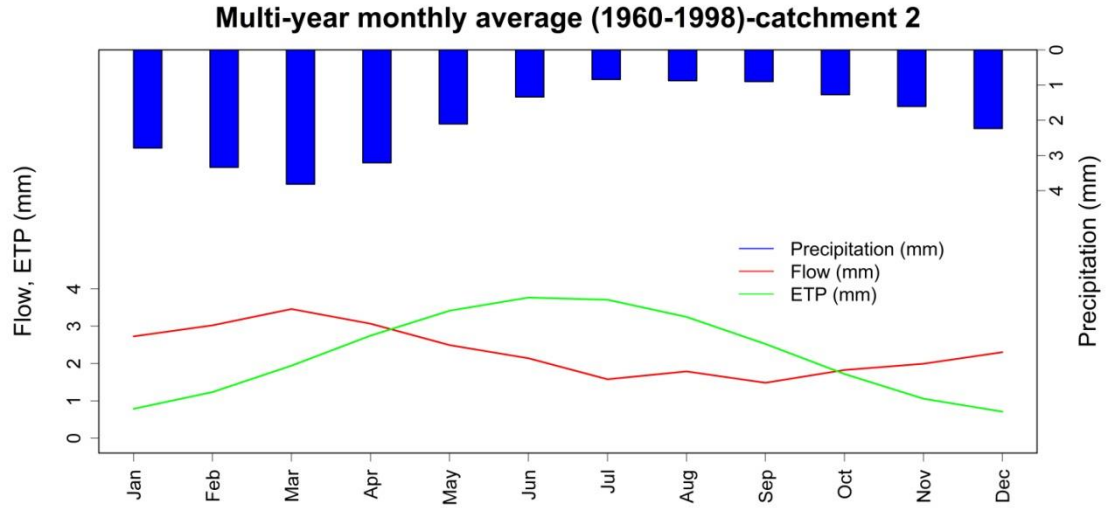

Figura 2. Mensual multianual de precipitación, Flow y ETP de la cuenca 2, para todo el periodo de estudio (1960 a 1998).

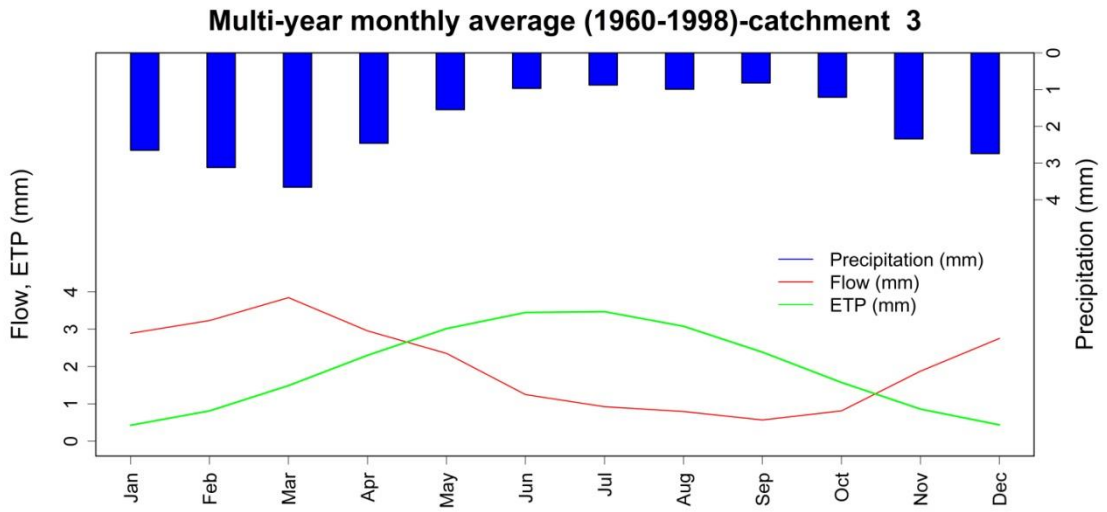

Figura 3. Mensual multianual de precipitación, Flow y ETP de la cuenca 3, para todo el periodo de estudio (1960 a 1998).

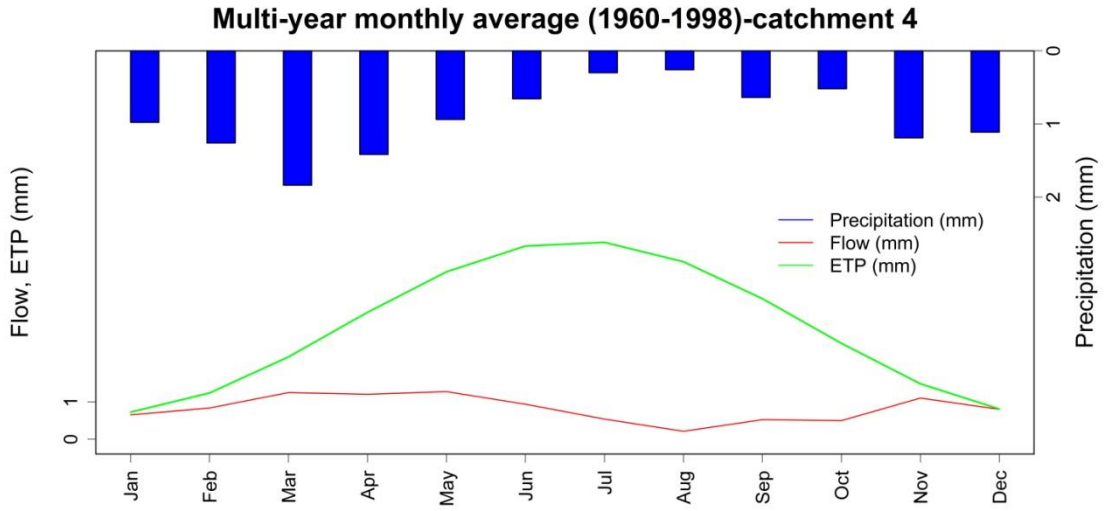

Figura 4. Mensual multianual de precipitación, Flow y ETP de la cuenca 4, para todo el periodo de estudio (1960 a 1998).

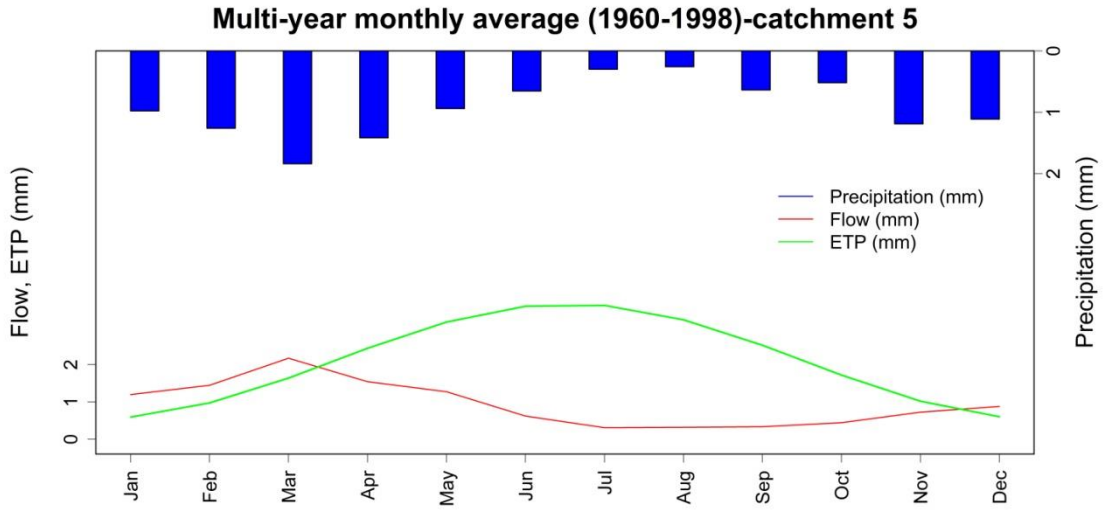

Figura 5. Mensual multianual de precipitación, Flow y ETP de la cuenca 5, para todo el periodo de estudio (1960 a 1998).

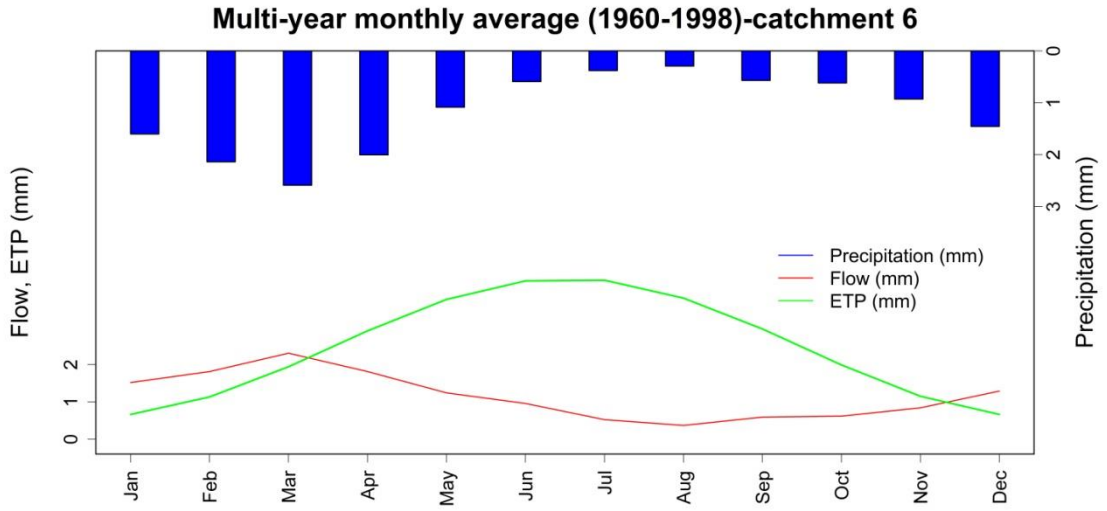

Figura 6. Mensual multianual de precipitación, Flow y ETP de la cuenca 6, para todo el periodo de estudio (1960 a 1998).

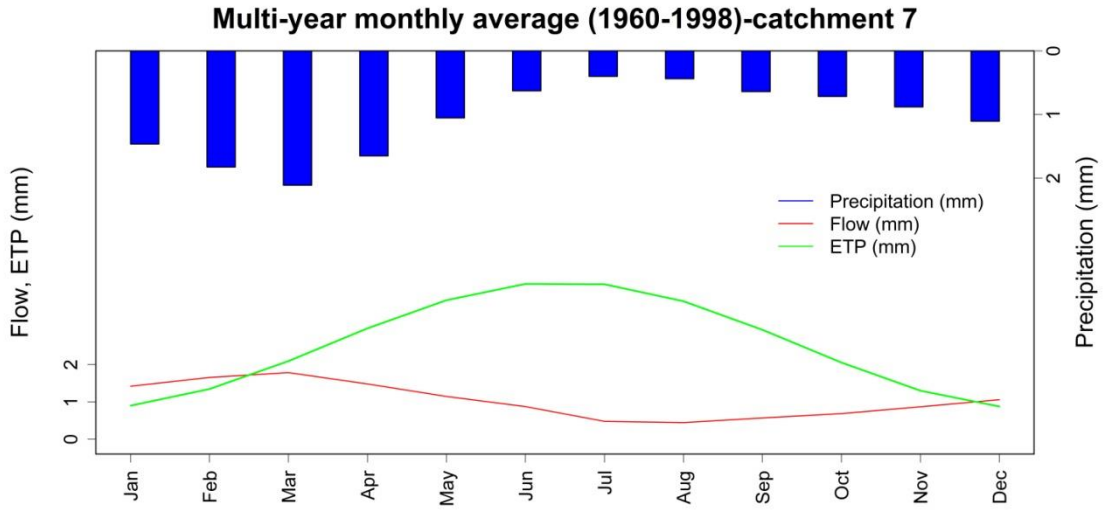

Figura 7. Mensual multianual de precipitación, Flow y ETP de la cuenca 7, para todo el periodo de estudio (1960 a 1998).

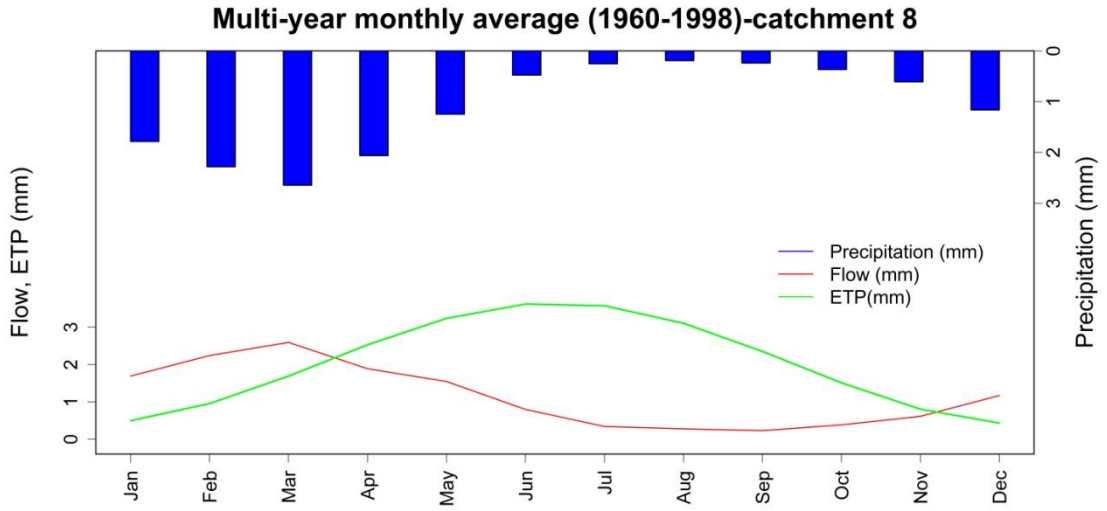

Figura 8. Mensual multianual de precipitación, Flow y ETP de la cuenca 8, para todo el periodo de estudio (1960 a 1998).

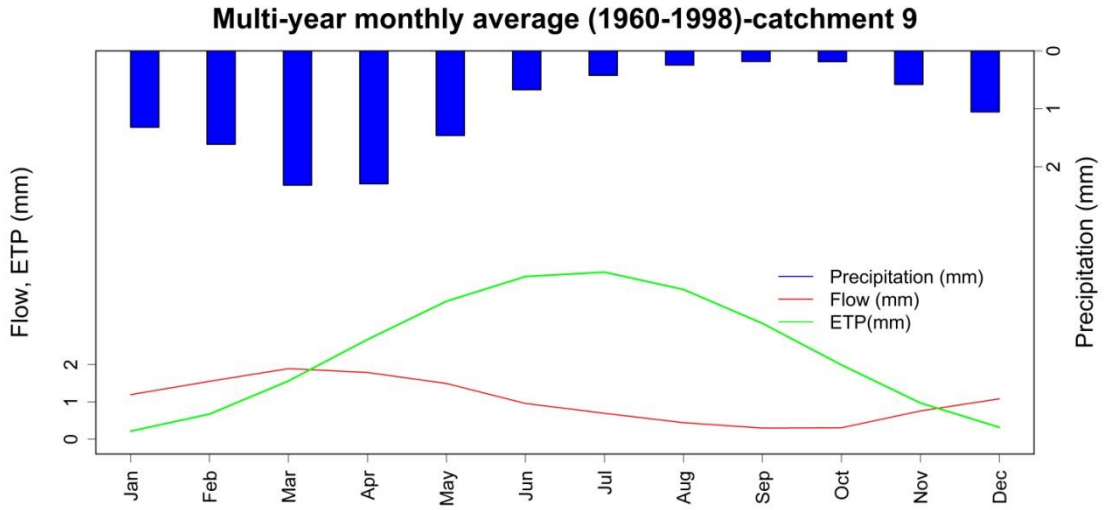

Figura 9. Mensual multianual de precipitación, Flow y ETP de la cuenca 9, para todo el periodo de estudio (1960 a 1998).

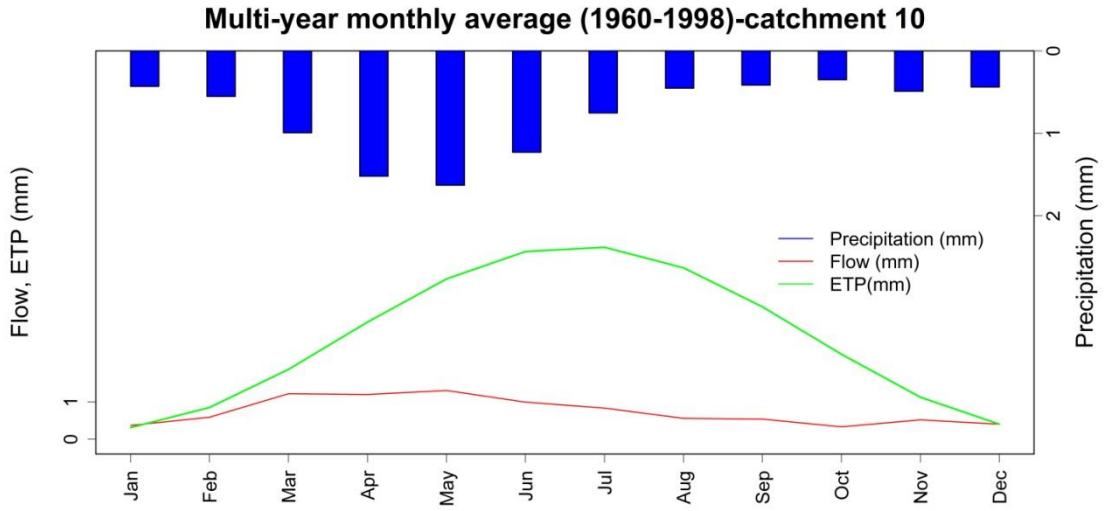

Figura 10. Mensual multianual de precipitación, Flow y ETP de la cuenca 10, para todo el periodo de estudio (1960 a 1998).

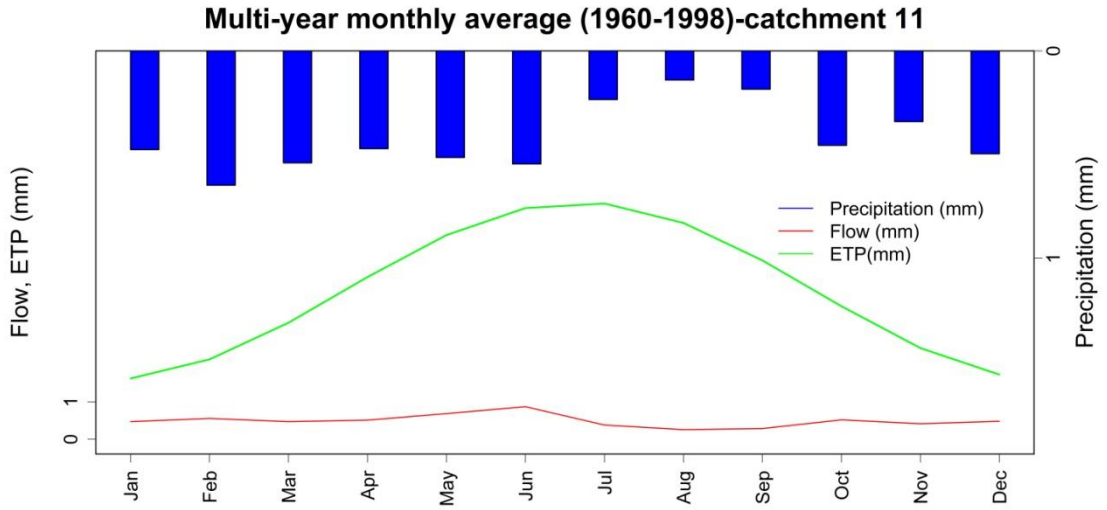

Figura 11. Mensual multianual de precipitación, Flow y ETP de la cuenca 11, para todo el periodo de estudio (1960 a 1998).

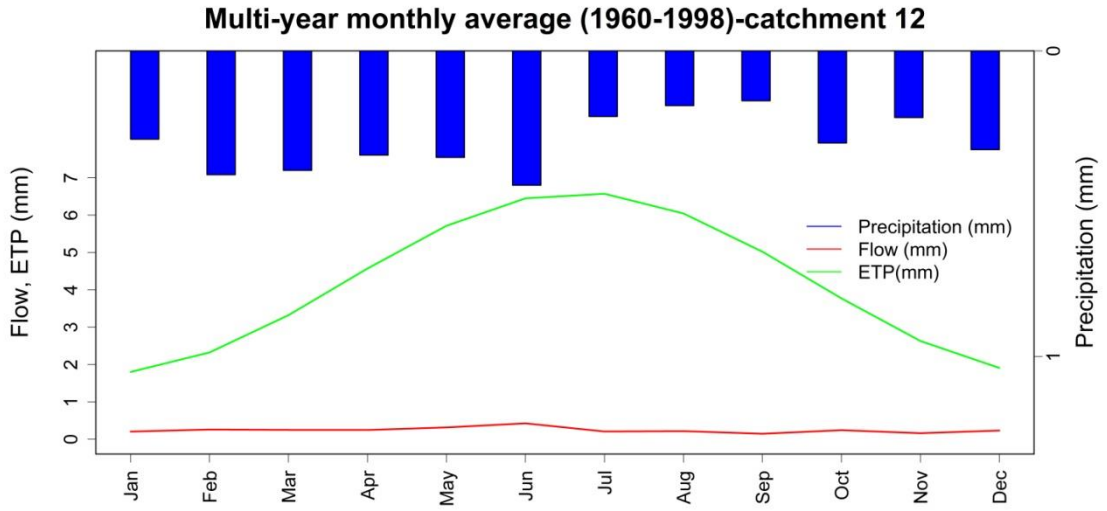

Figura 12. Mensual multianual de precipitación, Flow y ETP de la cuenca 12, para todo el periodo de estudio (1960 a 1998).

Funciones en lenguaje de programación R

 # ASSESSMET PERFORMANCE

NASH

```
Nash <- function(obs, sim, na.rm =
T){
  # inputs : obs = observations, sim =simulations;
  # Outputs: Nash-Sutcliffe efficiency
  NSE <- 1 - sum((obs - sim)^2) / sum((obs - mean(obs))^2)
  return(NSE)
}
```

KLINGUPTA

```
klinggupta <- function(obs, sim, na.rm = T){
  # inputs : obs = observations, sim =simulations;
  # Outputs: Kling-Gupta Efficiency
  r <- cor(obs, sim)              # coeficiente
correlation
  rat.var <- sd(sim) / sd(obs)     # variance rate
  rat.bias <- mean(sim) / mean(obs) # bias rate
  KGE <- 1 - sqrt(((r - 1)^2) + ((rat.var - 1)^2) + ((rat.bias - 1)^2))
  return(list(KGE = KGE, Correlation = r, CV.ratio = rat.var, bias.ratio =
rat.bias))
}
```

 # Normal Quantil Transformation

NQT

```
NQT <- function(N = 1000, factor.sample = 0.5, qobs, qsim, na.rm = T ){
  # Transformation to Normal space N(0,1) through Normal Quantil Transformation. (Nonparametric
approach)
  # inputs: N = numbers samples; factor.sample = double empirical observations; qobs = observations,
sim =simulations;
  # outputs: eta = observations N(0,1); etaS = simulations
N(0,1);
```

```
 # 1. interpolation observations and fitting nonparametric distribution
 # observations
 qmax <- 2 * max(qobs)
 qmin <- min(qobs)
 maxsize <- length(qobs)
 t <- 1 : maxsize
 tsample <- seq(from = 1, to = maxsize, by = factor.sample)
 qsam <- approx(t, qobs, xout = tsample)            # resample observations (interpolation)
 Pq <- pkde(q = qsam$y, fhat = kde(x = qobs, binned = TRUE ))
 # same for simulation
 qmaxs <- 2 * max(qsim)
 qmins <- min(qsim)
 maxsizes <- length(qsim)
 t_s <- 1 : maxsizes
 tsamples <- seq(from = 1, to = maxsizes, by = factor.sample)
 qssam <- approx(t_s, qsim, xout = tsamples)            # resample observations (interpolation)
 Pqs <- pkde(q = qssam$y, fhat = kde(x = qsim, binned = TRUE
))
 # 2. NQT transformation
 eta  <- qnorm(p = Pq, mean = 0, sd = 1)
 etaS <- qnorm(p = Pqs, mean = 0, sd = 1)

 return(list(eta = eta, etaS = etaS ))

}
```

 # COMPUTE PREDICTIVE DISTRIBUTION

```
PostPredDist <- function(forecastN, postparam, factor.sample, forecast){
 # inputs : forecastN = simulations un Normal space, postparam = posterior parameters,
factor.sample= double empirical observations, forecast = simulations or observations in real space
(depent it is calibration[obs], validation[sim])
 # Outputs: posterior predictive distribution
 qkde2 <- function (p, fhat)
 {
  if (any(p > 1) | any(p < 0))
   stop("p must be <= 1 and >= 0")
  cumul.prob <- pkde(q = fhat$eval.points, fhat = fhat)
  ind <- findInterval(x = p, vec = sort(cumul.prob))
  quant <- rep(0, length(ind))
  for (j in 1:length(ind)) {
   i <- ind[j]
   if (i == 0)
    quant[j] <- fhat$eval.points[1]
   else if (i >= length(fhat$eval.points))
    quant[j] <- fhat$eval.points[length(fhat$eval.points)]
   else {
    quant1 <- fhat$eval.points[i]
```

```
      quant2 <- fhat$eval.points[i + 1]
      prob1 <- cumul.prob[i]
      prob2 <- cumul.prob[i + 1]
      alpha <- (p[j] - prob2)/(prob1 - prob2)
      quant[j] <- quant1 * alpha + quant2 * (1 -
alpha)
     }
    }
   return(quant)
  }


  ypred <- matrix(0, length(forecastN), nrow(postparam))  #matrix(observations,posterior), crate a
matrix for predictions
  for(i in  1:length(forecastN)){
   for(j in 1:nrow(postparam)){
    # sample rnorm(n, mean = linear model, sd = error parameter estimate by MCMC )
    ypred[i, j] <- rnorm(1, postparam[j,1] + postparam[j,2] * forecastN[i], postparam[j,3])
   }
  }
  # posterior predictive distribution normal space
  posteriorN <- matrix(data = NA, nrow = nrow(ypred), ncol = ncol(ypred))
  for (k in 1:nrow(ypred)) {
   posteriorN[k,] <- pnorm(q = ypred[k,], mean = 0, sd = 1)
  }
  posteriorR <- matrix(data = NA, nrow = nrow(ypred), ncol = ncol(ypred))
  # browser()
  for (k in 1:nrow(ypred)) {
   posteriorR[k,] <- qkde(p = posteriorN[k,], fhat = kde(x = forecast, binned = TRUE ))
  }
  # invert matrix (posterior,time)
  tsample2 <- seq(from = 1, to = length(forecastN), by = 1/factor.sample)
  posteriorR2 <-t(posteriorR)     #transpose matrix
  for (k in 1:nrow(posterior)) {
   posterior[k,] <- posteriorR2 [k,tsample2]
  }
  return(posterior)
}
```

 # COMPUTE PREDICTIVE DISTRIBUTION FOR VALIDATION PERIOD

```
PostPredDistVal <- function(simvalN, parerrormod,factor.sample, simval, NP){
  # Compute the Posterior Predictive Distribution.
  # inputs : simvalN = simulations in Normal space, NP = number of points for predictive posterior =
1000,factor.sample= double empirical observations, simval = simulations or observations in real space
(depent it is calibration[obs], validation[sim])
  #          parerrormod = vector of parmeter postprocessor (b0,b1,e)
  # Outputs: posterior predictive distribution in validation
```

```
 # function for very low streamflows i.e Guadalupe cathment
 qkde2 <- function (p, fhat)
 {
  if (any(p > 1) | any(p < 0))
    stop("p must be <= 1 and >= 0")
  cumul.prob <- pkde(q = fhat$eval.points, fhat = fhat)
  ind <- findInterval(x = p, vec = sort(cumul.prob))
  quant <- rep(0, length(ind))
  for (j in 1:length(ind)) {
    i <- ind[j]
    if (i == 0)
      quant[j] <- fhat$eval.points[1]
    else if (i >= length(fhat$eval.points))
      quant[j] <- fhat$eval.points[length(fhat$eval.points)]
    else {
      quant1 <- fhat$eval.points[i]
      quant2 <- fhat$eval.points[i + 1]
      prob1 <- cumul.prob[i]
      prob2 <- cumul.prob[i + 1]
      alpha <- (p[j] - prob2)/(prob1 - prob2)
      quant[j] <- quant1 * alpha + quant2 * (1 -
alpha)
    }
  }
  return(quant)
 }


 ypredval <- matrix(data = NA, nrow = NP, ncol = length(simvalN))  #matrix(observations,posterior),
crate a matrix for predictions
 for (i in 1:length(simvalN)){
  ypredval[, i] <- rnorm(NP, parerrormod[1] + parerrormod[2] * simvalN,
parerrormod[3])
 }


 # posterior predictive distribution normal space
 posteriorNval <- matrix(data = NA, nrow = nrow(ypredval), ncol = ncol(ypredval))
 for (k in 1:nrow(ypredval)) {
  posteriorNval[k,] <- pnorm(q = ypredval[k,], mean = 0, sd = 1)
 }
 # posterior predictive distribution real space with resample  ¡¡¡ojoj¡¡¡ change kde for validation
 posteriorRval <- matrix(data = NA, nrow = nrow(ypredval), ncol = ncol(ypredval))
 for (k in 1:nrow(ypredval)) {
  posteriorRval[k,] <- qkde(p = posteriorNval[k,], fhat = kde(x = simval, binned = TRUE ))
  #posteriorRval[k,] <- qkde2(p = posteriorNval[k,], fhat = kde(x = simval, binned = TRUE )) # just for
low flow near to zero
 }
```

```
  # posterior predictive distribution real space without resample  ¡¡¡¡ojoj¡¡¡¡ change lenght to validation
for validation
  # invert matrix (posterior,time)
  tsample2 <- seq(from = 1, to = length(simvalN), by = 1/factor.sample)
  posteriorval <- matrix(data = NA, nrow = ncol(ypredval), ncol = length(tsample2)) #¡¡¡¡ojoj¡¡¡¡
  posteriorR2val <-t(posteriorRval)     #transpose matrix
  for (k in 1:nrow(posteriorval)) {
    posteriorval[k,] <- posteriorR2val [k,tsample2]
  }
  return(posteriorval)
}
```

 # Edit posterior

```
Edit.Posterior <- function(observations, posterior){
  # inputs : discharge = observations, posterior = matrix (posterior,time);
  # Outputs: QQplot, kolmogorov test
  # functions for compute mode as summary statistics
  moda <- function(x, na.rm = T){
    # function compute mode of
vector
    uniqv <- unique(x)
    uniqv[which.max(tabulate(match(x, uniqv)))]

  }
  pos.posterior <- matrix(data = NA, nrow = length(observations), ncol = 10)
  colnames(pos.posterior) <-
c("sd","mean","median","quantile5","quantile95","p","ecdf","quan2_5","quan97_5","moda")
  posterior <-t(posterior)  # transpose data (time, posterior)
  for (i in 1:length(observations)) {
    pos.posterior[i,1] <- sd(posterior[i,], na.rm = T)
    pos.posterior[i,2] <- mean(posterior[i,], na.rm = T)
    pos.posterior[i,3] <- median(posterior[i,], na.rm = T)
    pos.posterior[i,4] <- quantile(posterior[i,], probs = c(0.05),na.rm = T)
    pos.posterior[i,5] <- quantile(posterior[i,], probs = c(0.95),na.rm = T)
    pos.posterior[i,6] <- pnorm(observations[i], pos.posterior[i,2],  pos.posterior[i,1])
    posteriorECDF <- ecdf(pos.posterior[i,])
    pos.posterior[i,7] <- posteriorECDF(observations[i])
    pos.posterior[i,8] <- quantile(posterior[i,], probs = c(0.025),na.rm = T)
    pos.posterior[i,9] <- quantile(posterior[i,], probs = c(0.975),na.rm = T)
    pos.posterior[i,10] <- moda(posterior[i,], na.rm = T)
  }
  return(pos.posterior)

}
```

 # Test Predictive Uncertainty

```
# 1. Cotton test 1
```

```
QQtest <- function(discharge, posterior) {
  # inputs : discharge = observations, posterior = matrix (posterior,time);
  # Outputs: QQplot, kolmogorov test
  # calls  :QQplotout <- QQtest(discharge = Datos$q, posterior = y)
  # QQplotout <- QQtest(discharge = New.datos$q, posterior =
COPfrenchBroad$posterior)
  # QQplotoutPrueba <- QQtest(discharge = DataSitarum$Observation.discharge, posterior =
COPsitarum$posterior)
  # QQplotoutmcplnor <- QQtest(discharge = DataSitarum$Observation.discharge, posterior =
mcpoutputLNORMAL$posterior)
  require(Bolstad2); require(NSM3);
  # 1.1 reduce dimension of monte carlo matix
  y.tran <- t(posterior)                # transpose matrix y (time, posterior)
  zi <-matrix(data = NA, nrow = length(discharge), ncol = 1 )
  for (i in 1:length(discharge)) {
    m.ecdf <-ecdf(y.tran[i,])     # calculated cdf for every
posterior
    zi[i,1] <- m.ecdf(discharge[i])  # evaluate every observation in every cdf
  }
  # without loop
  # m.ecdf <-ecdf(y.tran)
  # zi <- m.ecdf(Datos$q)
  x <- qunif(ppoints(length(discharge)))          # theoretical quantile, observed data length
  # qqplot(x,zi,xlab="Theoretical Quantile U[0,1]",ylab="Quantile of forecast")
  # abline(0,1,lwd=2)
  # line(x, x + 0.05)
  # line(x, x - 0.05)
  qqarea <- sintegral(x, abs(x - zi), n.pts = 525)$int   #256
  #text(paste("Area ",formatC(qqarea,digits=4, format="f"),sep=""), x=0.9 ,y=0.3)
  # dev.copy(png, file="ReliabilityDiagramMCPcitarum.png", res=200, height=12, width=20, units="in")
  # dev.off()
  alphaInd <- 1 - (2 * (qqarea /length(discharge)))  # reliability index
  # Estimate precision or sharpeness or resolution (average precision)
  Esperanza <- apply(posterior, 2, mean, na.rm = T)  # expected posterior
  Desv      <- apply(posterior, 2, sd, na.rm = T)  # standar deviations posterior
  precision <- (sum(Esperanza /Desv) / length(discharge))
  return(list(area = qqarea, alpha = alphaInd, x = x, zi = zi, resolution = precision))
  # Kendall's test of independence
  #cor.test(zi, x, method="kendall")

}
```

 # 2. Cotton test 2: How many points outside the uncertainty band

```
TestUncertBand <- function(observations, quantil5, quantil95){
  # inputs : observations = discharge; quantil5 = vector quantile 5%; quantil95 = vector quantile 95%
  # Outputs: Cotton test II, how many points outside the uncertainty band
  # calls  :CottonTest2 <- TestUncertBand(observations = Datos$q, quantil5 = mcpoutput$q5, quantil95
= mcpoutput$q95)
```

```
  QoutsideBand <- length(which(observations <= quantil5 | observations >= quantil95)) # identified
points outside uncertainty band
  PorOutBand   <- (QoutsideBand/length(observations)) * 100                    # estimate
%
  Coverage95  <- abs(100 - PorOutBand)
  return(Coverage95)
}

# 95% Prediction Probability Uncertainty Band (95PPU)
UncertaintyBand95 <- function(observations, quantil5, quantil95){
  # inputs    : observations = streamflow discharge; quantil5 = vector quantile 5%; quantil95 = vector
quantile 95%
  # Outputs   : B = Average band with of 95PPU; CR = containing ratio; D = Average deviation amplitude,
d_factor = d-factor is the average with fo the prediction interval
  # calls     : PPU95 <- UncertaintyBand95(observations = as.numeric(QCalAipe), quantil5 =
Est.PosteriorMCMC[,4], quantil95 = Est.PosteriorMCMC[,5])
  n  <- length(observations)
  B  <- (1 / n) * sum(quantil95 - quantil5)
  # number of observations outside of this bounds
  nc <- length(which(observations <= quantil5 | observations >= quantil95))
  # number of observations enveloped by this bounds
  CR <- ((n - nc) / n)*100
  D  <- (1 / n) * sum(0.5 * abs((quantil95 + quantil5) - observations))
  d_factor <- ((1 / n) * sum(quantil95 - quantil5)) / sd(observations)
  return(list(B = B, CR = CR, D = D, d_factor = d_factor))
}

moda <- function(x){
  # function compute mode of vector
  uniqv <- unique(x)
  uniqv[which.max(tabulate(match(x, uniqv)))]

}
```

Post-procesadores en lenguaje de programación R

```
#. ABC postprocessor ----------------------------------------------------------------------------

ABCpostprocessor <- function(calobs, calsim, valobs, valsim, n = 100000, linfa, lsupa, linfb, lsupb, linfe,
lsupe){
  # Inputs: calobs, calsim, valobs, valsim are observations and simulations for calibration and validation
period respectively,  are time series object; n = 1000000  # number of simulations
  # Outputs: posterior predictive uncertainty
  ## make NQT transformation
  TransNQT <- NQT(N = 1000, factor.sample = 0.5, qobs = as.numeric(calobs), qsim =
as.numeric(calsim))
  TransNQTval <- NQT(N = 1000, factor.sample = 0.5, qobs = as.numeric(valobs), qsim =
as.numeric(valsim))

  datos <- cbind(TransNQT$eta, TransNQT$etaS)
  # inputs:
```

```
  # 1. vector observed summary statistics
  # 2. matrix of the simulated summary statistics (simulates, summary statistic)
  # 3. matrix of the simulated parameters values (simulates, parameter)
  # prepar my inputs:
  # 1) define summary statistic from observations
  my.stat.obs <- c(mean(datos[,1]))
  names(my.stat.obs) <- c("mean")
  # 2) define prior parameters
  # Uniform prior parameters
  mypar <- data.frame(a =runif(n,linfa,lsupa), b = runif(n,linfb,lsupb), sde =runif(n,linfe,lsupe))
  # gaussian parmeters
  #mypar <- data.frame(a = rnorm(n,mean = -0.0222, sd = 0.005), b = rnorm(n,mean = 1.3287, sd =
0.01), sde =rnorm(n,mean = 0.9, sd = 0.5)) # matirx of parameter, sde = rgamma(100000, shape = 1,
rate = 1/100)
  # eliminate negative values
  # mypar$sde[mypar$sde <0] <- 0
  # summary(mypar)


  # 3) simulated according with sample parameters
  print("Simulating data to ABC")
  ysim <- matrix(NA,dim(datos)[1],dim(mypar)[1])
  for (i in 1:dim(mypar)[1]) {
    ysim[,i] <-  mypar$a[i] + mypar$b[i] * datos[,2] + rnorm(1,0,mypar$sde[i])  # +
rnorm(1,0,mypar$sde[i]) simulation in normal space <- datos[,2]
  }
  # eliminate negative values
  # ysim[ysim <0] <- 0
  # any(is.na(ysim))
  # 4) estimate summary statistics from simulate data
  #print("Compute summary statistics from simulating data (ABC)")
  mysim <-data.frame(mean.sim = apply(ysim, 2, mean))
  summary(mysim)
  # 5) ABC rejection sample
  my.rej <- abc(target=my.stat.obs, param=mypar, sumstat=mysim, tol=0.01, method = "rejection")
#tol=0.0001, tol=0.01, tol=.1
  summary(my.rej)
  sumabcpar <-summary(my.rej)
  parerrormod <- c(median(my.rej$unadj.values[,1]), median(my.rej$unadj.values[,2]),
median(my.rej$unadj.values[,3]))
  print("Compute Posterior Predictive Distribution from ABC")
  PostPredABC <- PostPredDist(forecastN = datos[,2], postparam = my.rej$unadj.values, factor.sample
= 0.5, forecast = as.numeric(calobs))
  save(PostPredABC, file = "PostPredABC.RData")
  #browser()
  # compute statistiscs of posterior predictive
  Est.PosteriorABC <- Edit.Posterior(observations = as.numeric(calobs), posterior = PostPredABC)
  save(Est.PosteriorABC, file = "Est.PosteriorABC.RData")
```

```
  # Predective posterior Validation period
  PostPredVal <- PostPredDistVal(simvalN = TransNQTval$etaS, parerrormod = parerrormod,
factor.sample = 0.5, simval = as.numeric(valobs), NP = 1000)
  EstPostPredVal <- Edit.Posterior(observations = as.numeric(valobs), posterior = PostPredVal)
  save(PostPredVal, file = "PostPredVal.RData")
  save(EstPostPredVal, file = "EstPostPredVal.RData")
  #  Performance metrics
  # calibration period
  QQplot <- QQtest(discharge = as.numeric(calobs), posterior = PostPredABC)
  NSE <- Nash(obs = as.numeric(calobs), sim = Est.PosteriorABC[,"median"])
  KGE <- klinggupta(obs = as.numeric(calobs), sim = Est.PosteriorABC[,"median"])
  CottonTest2 <- TestUncertBand(observations = as.numeric(calobs), quantil5 =
Est.PosteriorABC[,"quan2_5"], quantil95 = Est.PosteriorABC[,"quan97_5"])
  PPU95 <- UncertaintyBand95(observations = as.numeric(calobs), quantil5 =
Est.PosteriorABC[,"quan2_5"], quantil95 = Est.PosteriorABC[,"quan97_5"])
  # Uniformity Test calibration
  UnifTestCalABC <-ks.test(QQplot$zi,'punif')
  # Ho : data are from U(0,1)
  # Ha : data are not from U(0,1)
  # p-value : 0.8106 > 0.05: accept Ho, data from U(0,1)


  # validation period
  QQplotv <- QQtest(discharge = as.numeric(valobs), posterior =
PostPredVal)
  NSEv <- Nash(obs = as.numeric(valobs), sim = EstPostPredVal[,"median"])
  KGEv <- klinggupta(obs = as.numeric(valobs), sim = EstPostPredVal[,"median"])
  CottonTest2v <- TestUncertBand(observations = as.numeric(valobs), quantil5 =
EstPostPredVal[,"quan2_5"], quantil95 = EstPostPredVal[,"quan97_5"])
  PPU95v <- UncertaintyBand95(observations = as.numeric(valobs), quantil5 =
EstPostPredVal[,"quan2_5"], quantil95 = EstPostPredVal[,"quan97_5"])
  # Uniformity Test calibration
  UnifTestValABCv <-ks.test(QQplotv$zi,'punif')


  return(list(posterior = PostPredABC, NSE = NSE, KGE = KGE, UnifTest = UnifTestCalABC$p.value,
PoinsInsideBand = CottonTest2, Est.Posterior = Est.PosteriorABC, PPU95 = PPU95, QQplot = QQplot,
abcpar = my.rej, posteriorv = PostPredVal, NSEv = NSEv, KGEv = KGEv, UnifTestv =
UnifTestValABCv$p.value,  PoinsInsideBandv = CottonTest2v, Est.Posteriorv = EstPostPredVal, PPU95v
= PPU95v, QQplotv = QQplotv))
}


#  MCMC postprocessor----------

MCMCpostprocessor <- function(calobs, calsim, valobs, valsim){
  # Inputs: calobs, calsim, valobs, valsim are observations and simulations for calibration and validation
period respectively,  are time series object;
  # Outputs: posterior predictive uncertainty and metrics
  # Assessmet Performance
```

```
 # Function for MCMC post-
processor
 # Use de same structure datos["Obs", "Sim"]
 # 2.1 Define a Bayesian linear regression model;
 li_reg <- function(pars, data){
   a    <- pars[1]   # intercept
   b    <- pars[2]   # slope
   sd_e <- pars[3]   # error (residuals)
   if(sd_e <= 0){return(NaN)}
   pred <- a + b * data[,2]   # linear regression
model
   # like likelihood function: dnorm(x, mean = 0, sd = 1, log = FALSE
   log_likelihood <- sum(dnorm(x = data[,1], mean = pred, sd = sd_e, log = TRUE))
   prior <- prior_reg(pars)   # define prior of parameters as function
   return(log_likelihood + prior)
 }
 # 2.2 Define the prior distribution
 prior_reg <- function(pars){
   a      <- pars[1]   # intercept
   b      <- pars[2]   # slope
   epsilon <- pars[3]   # error
   # non-informative (flat) priors on
all
   prior_a <- dnorm(x = a, mean = 0, sd = 100, log = T)   # normal but too much sd
   prior_b <- dnorm(x = b, mean = 0, sd = 100, log = T)   # prior b
   prior_epsilon <- dgamma(x = epsilon, shape = 1, rate = 1/100, log = T)
   return(prior_a + prior_b + prior_epsilon)
 }


 ## make NQT transformation
 TransNQT <- NQT(N = 1000, factor.sample = 0.5, qobs = as.numeric(calobs), qsim =
as.numeric(calsim))
 TransNQTval <- NQT(N = 1000, factor.sample = 0.5, qobs = as.numeric(valobs), qsim =
as.numeric(valsim))



 mcmc.lrm <- Metro_Hastings(li_func = li_reg, pars = c(0,1,1), par_names = c('a','b','epsilon'), data =
datos) #pars = c(0,1,1)
 mcmc.lrm <- Metro_Hastings(li_func = li_reg, pars = c(0,1,1), prop_sigma = mcmc.lrm$prop_sigma,
par_names = c('a','b','epsilon'), data = datos)
 ran_aprimcmc <- summary(mcmc.lrm$trace)
 (ran_aprimcmc)
 parerrormod <- c(median(mcmc.lrm$trace[,1]), median(mcmc.lrm$trace[,2]),
median(mcmc.lrm$trace[,3]))


 print("Compute Posterior Predictive Distribution from MCMC for calibration")
 PostPredMCMC <- PostPredDist(forecastN = datos[,2], postparam = mcmc.lrm$trace, factor.sample =
0.5, forecast = as.numeric(calobs))
 save(PostPredMCMC, file = "PostPredMCMC.RData")
```

```
  Est.PosteriorMCMC <- Edit.Posterior(observations = as.numeric(calobs), posterior = PostPredMCMC)
  save(Est.PosteriorMCMC, file = "Est.PosteriorMCMC.RData")

  # Predective posterior Validation period
  print("Compute Posterior Predictive Distribution from MCMC for validation")
  PostPredVal <- PostPredDistVal(simvalN = TransNQTval$etaS, parerrormod = parerrormod,
factor.sample = 0.5, simval = as.numeric(valobs), NP = 1000)
  EstPostPredVal <- Edit.Posterior(observations = as.numeric(valobs), posterior = PostPredVal)
  save(PostPredVal, file = "PostPredVal.RData")
  save(EstPostPredVal, file = "EstPostPredVal.RData")
  # beep(8)
  # browser()
  #  Performance metrics
  QQplot <- QQtest(discharge = as.numeric(calobs), posterior = PostPredMCMC)
  NSE <- Nash(obs = as.numeric(calobs), sim = Est.PosteriorMCMC[,"median"])
  KGE <- klinggupta(obs = as.numeric(calobs), sim = Est.PosteriorMCMC[,"median"])
  CottonTest2 <- TestUncertBand(observations = as.numeric(calobs), quantil5 =
Est.PosteriorMCMC[,"quan2_5"], quantil95 = Est.PosteriorMCMC[,"quan97_5"])
  PPU95 <- UncertaintyBand95(observations = as.numeric(calobs), quantil5 =
Est.PosteriorMCMC[,"quan2_5"], quantil95 = Est.PosteriorMCMC[,"quan97_5"])
  # Uniformity Test calibration
  UnifTestCal <-ks.test(QQplot$zi,'punif')
  # Ho : data are from U(0,1)
  # Ha : data are not from U(0,1)
  # p-value : 0.8106 > 0.05: accept Ho, data from U(0,1)

  # validation period
  QQplotv <- QQtest(discharge = as.numeric(valobs), posterior =
PostPredVal)
  NSEv <- Nash(obs = as.numeric(valobs), sim = EstPostPredVal[,"median"])
  KGEv <- klinggupta(obs = as.numeric(valobs), sim = EstPostPredVal[,"median"])
  CottonTest2v <- TestUncertBand(observations = as.numeric(valobs), quantil5 =
EstPostPredVal[,"quan2_5"], quantil95 = EstPostPredVal[,"quan97_5"])
  PPU95v <- UncertaintyBand95(observations = as.numeric(valobs), quantil5 =
EstPostPredVal[,"quan2_5"], quantil95 = EstPostPredVal[,"quan97_5"])
  # Uniformity Test calibration
  UnifTestValABCv <-ks.test(QQplotv$zi,'punif')

  return(list(posterior = PostPredMCMC, NSE = NSE, KGE = KGE, UnifTest = UnifTestCal$p.value,
PoinsInsideBand = CottonTest2, Est.Posterior = Est.PosteriorMCMC, PPU95 = PPU95, QQplot = QQplot,
mcmcpar =ran_aprimcmc, posteriorv = PostPredVal, NSEv = NSEv, KGEv = KGEv, UnifTestv =
UnifTestValABCv$p.value,  PoinsInsideBandv = CottonTest2v, Est.Posteriorv = EstPostPredVal, PPU95v
= PPU95v, QQplotv = QQplotv))
}

#  MCP postprocessor ----------------------------------------------------------------------------


MCP2D <- function(calobs, calsim, valobs, valsim, N){
```

```
  # Inputs: calobs, calsim, valobs, valsim are observations and simulations for calibration and validation period respectively,  are time series object;
  # Outputs: posterior predictive uncertainty and metrics

  # 1. Make NQT transformation for calibration and validation period
  TransNQT <- NQT(N = 1000, factor.sample = 0.5, qobs = as.numeric(calobs), qsim = as.numeric(calsim))
  TransNQTval <- NQT(N = 1000, factor.sample = 0.5, qobs = as.numeric(valobs), qsim = as.numeric(valsim))

  # 2. Estimating moments from conditional pdf (bivariate gaussian distribution)
  Mat.eta <- cbind(TransNQT$eta, TransNQT$etaS)
  MatCov <- cov(x = Mat.eta)   # Variance - covariance matix
  AuxMat <- MatCov[1,2] * (MatCov[2,2]^ -1)
  GausConMean <- mean(TransNQT$eta) + (TransNQT$etaS - mean(TransNQT$etaS)) * AuxMat
  GausConVar <- var(TransNQT$eta) - MatCov[1,2] * AuxMat

  # 2.1 parmeters for validation
  GausConMeanv <- mean(TransNQT$eta) + (TransNQTval$etaS - mean(TransNQTval$etaS)) * AuxMat

  # 3. Compute condictional predictive distribution (PD)

  PosteriorPred <- function(meancond, varcond, simval, N = 1000, factor.sample = 0.5){
    # Inputs: meancond = mean conditional vector, varcond = variance conditional parameter, simval = observations in calibration or simulations in validation,
    # Outputs: posterior predictive uncertainty
    # Calls: calibration::  PostPredMCP <- PosteriorPred (meancond = GausConMean, varcond = GausConVar, simval = calobs, N = 1000, factor.sample = 0.5)
    # Calls: validation::  PostPredMCPv <- PosteriorPred (meancond = GausConMeanv, varcond = GausConVar, simval = valsim, N = 1000, factor.sample = 0.5)

    # function for very low streamflows i.e Guadalupe cathment
    qkde2 <- function (p, fhat)
    {
      if (any(p > 1) | any(p < 0))
        stop("p must be <= 1 and >= 0")
      cumul.prob <- pkde(q = fhat$eval.points, fhat = fhat)
      ind <- findInterval(x = p, vec = sort(cumul.prob))
      quant <- rep(0, length(ind))
      for (j in 1:length(ind)) {
        i <- ind[j]
        if (i == 0)
          quant[j] <- fhat$eval.points[1]
        else if (i >= length(fhat$eval.points))
          quant[j] <- fhat$eval.points[length(fhat$eval.points)]
        else {
          quant1 <- fhat$eval.points[i]
```

```
        quant2 <- fhat$eval.points[i +
1]
        prob1 <- cumul.prob[i]
        prob2 <- cumul.prob[i + 1]
        alpha <- (p[j] - prob2)/(prob1 - prob2)
        quant[j] <- quant1 * alpha + quant2 * (1 - alpha)
       }
     }
     return(quant)
   }


  # Compute condictional predictive distribution (PD) in Normal space
  # matrix to compute PD matrix(posterior, time(lenght of
data))
  ypred <- matrix(data = NA, nrow = N, ncol =
length(meancond))
  for (i in 1:length(meancond)) {
    ypred[,i] <- rnorm(n = N, mean = meancond[i], sd = (varcond ^ 0.5))
  }

  # 4. Inverse NQT for predictive distribution
  # posterior predictive distribution normal space
  posteriorN <- matrix(data = NA, nrow = nrow(ypred), ncol = ncol(ypred))
  for (k in 1:nrow(ypred)) {
    posteriorN[k,] <- pnorm(q = ypred[k,], mean = 0, sd = 1)
  }
  # posterior predictive distribution real space with resample
  posteriorR <- matrix(data = NA, nrow = nrow(ypred), ncol = ncol(ypred))
  for (k in 1:nrow(ypred)) {
    posteriorR[k,] <- qkde(p = posteriorN[k,], fhat = kde(x = simval, binned = TRUE ))
    #posteriorR[k,] <- qkde2(p = posteriorN[k,], fhat = kde(x = simval, binned = TRUE )) # just for low
flow near to zero
  }
  # posterior predictive distribution real space without resample
  # invert matrix (posterior,time)
  #factor.sample <- 0.5 # factor resample for NQT
  tsample2 <- seq(from = 1, to = length(meancond), by = 1/factor.sample)
  posterior <- matrix(data = NA, nrow = N, ncol = length(tsample2)) #¡¡¡¡ojo¡¡¡¡
  #posteriorR2 <-t(posteriorR)
  #browser()
  for (k in 1:nrow(posterior)) {
    posterior[k,] <- posteriorR [k,tsample2]
  }
  return(posterior)
  }

  # compute predictive distribution calibration
```

```
  print("Compute Posterior Predictive Distribution MCP for calibration")
  PostPredMCP <- PosteriorPred (meancond = GausConMean, varcond = GausConVar, simval =
as.numeric(calobs), N = 1000, factor.sample = 0.5)
  save(PostPredMCP, file = "PostPredMCP.RData")
  # compute statistiscs of posterior predictive
  Est.PosteriorMCP <- Edit.Posterior(observations = as.numeric(calobs), posterior = PostPredMCP)
  save(Est.PosteriorMCP, file = "Est.PosteriorMCP.RData")

  # Predective posterior Validation period
  print("Compute Posterior Predictive Distribution MCP for validation")
  #PostPredMCPval <- PosteriorPred (meancond = GausConMeanv, varcond = GausConVar, simval =
as.numeric(valsim), N = 1000, factor.sample = 0.5)
  PostPredMCPval <- PosteriorPred (meancond = GausConMeanv, varcond = GausConVar, simval =
as.numeric(valobs), N = 1000, factor.sample = 0.5)
  EstPosteriorMCPval <- Edit.Posterior(observations = as.numeric(valobs), posterior = PostPredMCPval)
  save(PostPredMCPval, file = "PostPredMCPval.RData")
  save(EstPosteriorMCPval, file = "EstPosteriorMCPval.RData")
  #browser()
  #  Performance metrics
  QQplot <- QQtest(discharge = as.numeric(calobs), posterior =
PostPredMCP)
  NSE <- Nash(obs = as.numeric(calobs), sim = Est.PosteriorMCP[,"median"])
  KGE <- klinggupta(obs = as.numeric(calobs), sim = Est.PosteriorMCP[,"median"])
  CottonTest2 <- TestUncertBand(observations = as.numeric(calobs), quantil5 =
Est.PosteriorMCP[,"quan2_5"], quantil95 = Est.PosteriorMCP[,"quan97_5"])
  PPU95 <- UncertaintyBand95(observations = as.numeric(calobs), quantil5 =
Est.PosteriorMCP[,"quan2_5"], quantil95 = Est.PosteriorMCP[,"quan97_5"])
  # Uniformity Test calibration
  UnifTestCal <-ks.test(QQplot$zi,'punif')
  # Ho : data are from U(0,1)
  # Ha : data are not from U(0,1)
  # p-value : 0.8106 > 0.05: accept Ho, data from U(0,1)

  # validation period
  QQplotv <- QQtest(discharge = as.numeric(valobs), posterior = PostPredMCPval)
  NSEv <- Nash(obs = as.numeric(valobs), sim = EstPosteriorMCPval[,"median"])
  KGEv <- klinggupta(obs = as.numeric(valobs), sim = EstPosteriorMCPval[,"median"])
  CottonTest2v <- TestUncertBand(observations = as.numeric(valobs), quantil5 =
EstPosteriorMCPval[,"quan2_5"], quantil95 = EstPosteriorMCPval[,"quan97_5"])
  PPU95v <- UncertaintyBand95(observations = as.numeric(valobs), quantil5 =
EstPosteriorMCPval[,"quan2_5"], quantil95 = EstPosteriorMCPval[,"quan97_5"])
  # Uniformity Test calibration
  UnifTestVal <-ks.test(QQplotv$zi,'punif')

  return(list(posterior = PostPredMCP, NSE = NSE, KGE = KGE, UnifTest = UnifTestCal$p.value,
PoinsInsideBand = CottonTest2, Est.Posterior = Est.PosteriorMCP, PPU95 = PPU95, QQplot = QQplot,
posteriorv = PostPredMCPval, NSEv = NSEv, KGEv = KGEv, UnifTestv = UnifTestVal$p.value,
PoinsInsideBandv = CottonTest2v, Est.Posteriorv = EstPosteriorMCPval, PPU95v = PPU95v, QQplotv =
QQplotv))
}
```

 # MCP Truncated Postprocessor ----------

```
Truncamiento <- function(calobs, calsim, inipoints) {
 # Inputs: calobs, calsim,  are observations and simulations for calibration period respectively,  are time series object;
 #        inipoints : range to search truncated point from scatter plot c(-1,1)
 # Outputs: Separate samples up and low with indices

 TransNQT <- NQT(N = 1000, factor.sample = 0.5, qobs = as.numeric(calobs), qsim = as.numeric(calsim))
 a <- NA       # initial value store trunkpoint
 varup <- 1000000 # optimization by brute force so hight value to into the cycle
 delta <- 0.01    # step to move on optimization
 j <- 1
 trunkpoint <- inipoints[1]
 # order the data
 sortetaS <- sort(TransNQT$etaS, index.return=TRUE)
 # extract only the index : sort(TransNQT$etaS, index.return=TRUE)$ix
 sorteta <- TransNQT$eta[sortetaS$ix] # order according sortetaS


 while (trunkpoint < inipoints[2]) {
  trunkidx <- which(sortetaS$x >= trunkpoint)
  upsample <- sorteta[trunkidx]
  upsampleS <- sortetaS$x[trunkidx]

  dist <- length(sorteta) - length(trunkidx)
  lowsample <- sorteta[1:dist]
  lowsampleS <- sortetaS$x[1:dist]

  if (is.na(trunkpoint)){
   print("Change range initial trunked point")
  }else{
   # below sample
   Mat.etalow <- cbind(lowsample, lowsampleS)
   MatCovlow <- cov(x = Mat.etalow)   # Variance - covariance matix
   AuxMatlow <- MatCovlow[1,2] * (MatCovlow[2,2]^ -1)
   GausConMeanlow <- mean(lowsample) + (lowsampleS - mean(lowsampleS)) * AuxMatlow
   GausConVarlow <- var(lowsample) - MatCovlow[1,2] * AuxMatlow
   # Up sample
   Mat.etaup <- cbind(upsample, upsampleS)
   MatCovup <- cov(x = Mat.etaup)   # Variance - covariance matix
   AuxMatup <- MatCovup[1,2] * (MatCovup[2,2]^ -1)
   GausConMeanup <- mean(upsample) + (upsampleS - mean(upsampleS)) * AuxMatup
   GausConVarup <- var(upsample) - MatCovup[1,2] * AuxMatup
```

```
    #
    # varstoreup[j] <- GausConVarup
    # varstorelow(j) <-
GausConVarlow
    # trunkpointstore <- trunkpoint
   }

   if (is.na(GausConVarup)){
    print("the up sample is to small so change range")
    browser()
   }

   if (GausConVarup < varup){
    varup <- GausConVarup
    varlow <- GausConVarlow
    a <- trunkpoint
   }
   j <- j+1
   trunkpoint <- trunkpoint + delta

  }

  idxup <- which(TransNQT$etaS > a)
  idxlow <- which(TransNQT$etaS <=
a)

  upsample <- TransNQT$eta[idxup]
  lowsample <- TransNQT$eta[idxlow]

  upsampleS <-
TransNQT$etaS[idxup]
  lowsampleS <- TransNQT$etaS[idxlow]

  return(list(trunkedpoint = a, upsample = upsample, lowsample = lowsample, upsampleS = upsampleS,
lowsampleS = lowsampleS, idxup = idxup, idxlow = idxlow))

}

MCP2Dtrun <- function(calobs, calsim, valobs, valsim, N, inipoints, factor.sample){
  # Inputs: calobs, calsim, valobs, valsim are observations and simulations for calibration and validation
period respectively,  are time series object;
  #       N : # samples from predictive uncertainty; inipoints : range to search truncated point from
scatter plot c(-1,1)
  # Outputs: posterior predictive uncertainty and metrics

  # 1. Make NQT transformation for calibration and validation period
  TransNQT <- NQT(N = 1000, factor.sample = 0.5, qobs = as.numeric(calobs), qsim =
as.numeric(calsim))
```

```
  TransNQTval <- NQT(N = 1000, factor.sample = 0.5, qobs = as.numeric(valobs), qsim =
as.numeric(valsim))

  # 2. Separate samples up and low
  separate <- Truncamiento(calobs = calobs, calsim = calsim, inipoints = inipoints)

  # 3. Estimating moments from conditional pdf (bivariate gaussian distribution) for up and low sample
  # 3.1 below sample
  Mat.etalow <- cbind(separate$lowsample, separate$lowsampleS)
  MatCovlow <- cov(x = Mat.etalow)   # Variance - covariance matix
  AuxMatlow <- MatCovlow[1,2] * (MatCovlow[2,2]^ -1)
  GausConMeanlow <- mean(separate$lowsample) + (separate$lowsampleS -
mean(separate$lowsampleS)) * AuxMatlow
  GausConVarlow <- var(separate$lowsample) - MatCovlow[1,2] *
AuxMatlow
  # 3.2 Up sample
  Mat.etaup <- cbind(separate$upsample, separate$upsampleS)
  MatCovup <- cov(x = Mat.etaup)   # Variance - covariance
matix
  AuxMatup <- MatCovup[1,2] * (MatCovup[2,2]^ -
1)
  GausConMeanup <- mean(separate$upsample) + (separate$upsampleS -
mean(separate$upsampleS)) * AuxMatup
  GausConVarup <- var(separate$upsample) - MatCovup[1,2] * AuxMatup
  # conditional mean for validation
  GausConMeanv <- mean(TransNQT$eta) + (TransNQTval$etaS - mean(TransNQTval$etaS)) *
AuxMatlow

  # 4. Compute condictional predictive distribution (PD)
  # for validation
  PosteriorPred <- function(meancond, varcond, simval, N = 1000, factor.sample = 0.5){
    # Inputs: meancond = mean conditional vector, varcond = variance conditional parameter, simval =
observations in calibration or simulations in validation,
    # Outputs: posterior predictive uncertainty (posterior, time)
    # Calls: calibration::  PostPredMCP <- PosteriorPred (meancond = GausConMean, varcond =
GausConVar, simval = calobs, N = 1000, factor.sample = 0.5)
    # Calls: validation::  PostPredMCPv <- PosteriorPred (meancond = GausConMeanv, varcond =
GausConVar, simval = valsim, N = 1000, factor.sample = 0.5)

    # function for very low streamflows i.e Guadalupe cathment
    qkde2 <- function (p, fhat)
    {
      if (any(p > 1) | any(p < 0))
        stop("p must be <= 1 and >= 0")
      cumul.prob <- pkde(q = fhat$eval.points, fhat = fhat)
      ind <- findInterval(x = p, vec =
sort(cumul.prob))
      quant <- rep(0, length(ind))
      for (j in 1:length(ind)) {
        i <- ind[j]
```

```
      if (i == 0)
        quant[j] <- fhat$eval.points[1]
      else if (i >= length(fhat$eval.points))
        quant[j] <- fhat$eval.points[length(fhat$eval.points)]
      else {
        quant1 <- fhat$eval.points[i]
        quant2 <- fhat$eval.points[i +
1]
        prob1 <- cumul.prob[i]
        prob2 <- cumul.prob[i + 1]
        alpha <- (p[j] - prob2)/(prob1 - prob2)
        quant[j] <- quant1 * alpha + quant2 * (1 - alpha)
      }
    }
    return(quant)
  }


  # Compute condictional predictive distribution (PD) in Normal space
  # matrix to compute PD matrix(posterior, time(lenght of data))
  ypred <- matrix(data = NA, nrow = N, ncol = length(meancond))
  for (i in 1:length(meancond)) {
    ypred[,i] <- rnorm(n = N, mean = meancond[i], sd = (varcond ^ 0.5))
  }

  # 4. Inverse NQT for predictive distribution
  # posterior predictive distribution normal space
  posteriorN <- matrix(data = NA, nrow = nrow(ypred), ncol = ncol(ypred))
  for (k in 1:nrow(ypred)) {
    posteriorN[k,] <- pnorm(q = ypred[k,], mean = 0, sd = 1)
  }
  # posterior predictive distribution real space with resample
  posteriorR <- matrix(data = NA, nrow = nrow(ypred), ncol = ncol(ypred))
  for (k in 1:nrow(ypred)) {
    posteriorR[k,] <- qkde(p = posteriorN[k,], fhat = kde(x = simval, binned = TRUE ))
    #posteriorR[k,] <- qkde2(p = posteriorN[k,], fhat = kde(x = simval, binned = TRUE )) # just for low
flow near to zero
  }
  # posterior predictive distribution real space without resample
  # invert matrix (posterior,time)
  #factor.sample <- 0.5 # factor resample for NQT
  tsample2 <- seq(from = 1, to = length(meancond), by = 1/factor.sample)
  posterior <- matrix(data = NA, nrow = N, ncol = length(tsample2)) #¡¡¡¡ojoiiii
  #posteriorR2 <-t(posteriorR)
  #browser()
  for (k in 1:nrow(posterior)) {
    posterior[k,] <- posteriorR [k,tsample2]
```

```
  }
  return(posterior)
 }

 PosteriorPredt <- function(meancondlow, varcondlow, meancondup, varcondup, simval, N = 1000,
factor.sample = 0.5, idxlow, idxup){
  # Inputs: meancondlow = mean conditional vector for low sample, varcondlow = variance
conditional parameter for low sample, simval = observations in calibration or simulations in validation,
  #       meancondup, varcondup are the same but for up sample; idxlow, idxup are order index for low
and up samples
  # Outputs: posterior predictive uncertainty (posterior, time)

  # function for very low streamflows i.e Guadalupe cathment
  qkde2 <- function (p, fhat)
  {
   if (any(p > 1) | any(p < 0))
    stop("p must be <= 1 and >= 0")
   cumul.prob <- pkde(q = fhat$eval.points, fhat = fhat)
   ind <- findInterval(x = p, vec =
sort(cumul.prob))
   quant <- rep(0, length(ind))
   for (j in 1:length(ind)) {
    i <- ind[j]
    if (i == 0)
     quant[j] <- fhat$eval.points[1]
    else if (i >= length(fhat$eval.points))
     quant[j] <- fhat$eval.points[length(fhat$eval.points)]
    else {
     quant1 <- fhat$eval.points[i]
     quant2 <- fhat$eval.points[i +
1]
     prob1 <- cumul.prob[i]
     prob2 <- cumul.prob[i + 1]
     alpha <- (p[j] - prob2)/(prob1 - prob2)
     quant[j] <- quant1 * alpha + quant2 * (1 - alpha)
    }
   }
   return(quant)
  }

  # Compute condictional predictive distribution (PD) in Normal space
  # matrix to compute PD matrix(posterior, time(lenght of data))
  # low sample
  ypredlow <- matrix(data = NA, nrow = N, ncol = length(meancondlow))
  for (i in 1:length(meancondlow)) {
   ypredlow[,i] <- rnorm(n = N, mean = meancondlow[i], sd = (varcondlow ^ 0.5))
  }
  # up sample
```

```
  ypredup <- matrix(data = NA, nrow = N, ncol = length(meancondup))
  for (i in 1:length(meancondup)) {
    ypredup[,i] <- rnorm(n = N, mean = meancondup[i], sd = (varcondup ^
0.5))
  }
  # 4. Inverse NQT for predictive distribution
  # posterior predictive distribution normal space
  # low sample
  posteriorNlow <- matrix(data = NA, nrow = nrow(ypredlow), ncol = ncol(ypredlow))
  for (k in 1:nrow(ypredlow)) {
    posteriorNlow[k,] <- pnorm(q = ypredlow[k,], mean = 0, sd = 1)
  }
  # up sample
  posteriorNup <- matrix(data = NA, nrow = nrow(ypredup), ncol = ncol(ypredup))
  for (k in 1:nrow(ypredup)) {
    posteriorNup[k,] <- pnorm(q = ypredup[k,], mean = 0, sd =
1)
  }
  # posterior predictive distribution real space with resample
  # low sample
  posteriorRlow <- matrix(data = NA, nrow = nrow(ypredlow), ncol = ncol(ypredlow))
  for (k in 1:nrow(ypredlow)) {
    posteriorRlow[k,] <- qkde(p = posteriorNlow[k,], fhat = kde(x = simval, binned = TRUE
))
    #posteriorRlow[k,] <- qkde2(p = posteriorNlow[k,], fhat = kde(x = simval, binned = TRUE )) # just for
low flow near to zero
  }
  # up sample
  posteriorRup <- matrix(data = NA, nrow = nrow(ypredup), ncol = ncol(ypredup))
  for (k in 1:nrow(ypredup)) {
    posteriorRup[k,] <- qkde(p = posteriorNup[k,], fhat = kde(x = simval, binned = TRUE ))
    #posteriorRup[k,] <- qkde2(p = posteriorNup[k,], fhat = kde(x = simval, binned = TRUE )) # just for
up flow near to zero
  }

  # posterior predictive distribution real space without resample
  # invert matrix (posterior,time)
  #factor.sample <- 0.5 # factor resample for NQT
  #browser()
  # # transform order to real space
  posteriorAux <- matrix(data = NA, nrow = N, ncol = dim(posteriorRup)[2]+dim(posteriorRlow)[2])
   # low sample
  for (i in 1:length(idxlow)) {
    posteriorAux[,idxlow[i]] <- posteriorRlow[,i]
  }
  # up sample
  for (i in 1:length(idxup)) {
    posteriorAux[,idxup[i]] <- posteriorRup[,i]
```

```
  }

  tsample2 <- seq(from = 1, to = dim(posteriorAux)[2], by = 1/factor.sample)
  posterior <- matrix(data = NA, nrow = N, ncol = length(tsample2)) #¡¡¡¡ojoi¡¡¡
  #posteriorR2 <-t(posteriorR)
  #browser()
  for (k in 1:nrow(posterior)) {
    posterior[k,] <- posteriorAux [k,tsample2]
  }
  return(posterior)
 }


 # compute predictive distribution calibration
 print("Compute Posterior Predictive Distribution MCPt for calibration")
 PostPredMCP <- PosteriorPredt(meancondlow = GausConMeanlow, varcondlow = GausConVarlow,
meancondup = GausConMeanup, varcondup = GausConVarup, simval = calobs, N = 1000,
factor.sample = 0.5, idxlow = separate$idxlow, idxup = separate$idxup)

 save(PostPredMCP, file = "PostPredMCP.RData")
 # compute statistiscs of posterior predictive
 Est.PosteriorMCP <- Edit.Posterior(observations = as.numeric(calobs), posterior = PostPredMCP)
 save(Est.PosteriorMCP, file = "Est.PosteriorMCP.RData")

 # Predective posterior Validation period
 print("Compute Posterior Predictive Distribution MCPt for validation")
 PostPredMCPval <- PosteriorPred (meancond = GausConMeanv, varcond = GausConVarlow, simval =
as.numeric(valobs), N = 1000, factor.sample = 0.5)
 EstPosteriorMCPval <- Edit.Posterior(observations = as.numeric(valobs), posterior = PostPredMCPval)
 save(PostPredMCPval, file = "PostPredMCPval.RData")
 save(EstPosteriorMCPval, file = "EstPosteriorMCPval.RData")
 #browser()
 #  Performance metrics
 QQplot <- QQtest(discharge = as.numeric(calobs), posterior =
PostPredMCP)
 NSE <- Nash(obs = as.numeric(calobs), sim = Est.PosteriorMCP[,"median"])
 KGE <- klinggupta(obs = as.numeric(calobs), sim = Est.PosteriorMCP[,"median"])
 CottonTest2 <- TestUncertBand(observations = as.numeric(calobs), quantil5 =
Est.PosteriorMCP[,"quan2_5"], quantil95 = Est.PosteriorMCP[,"quan97_5"])
 PPU95 <- UncertaintyBand95(observations = as.numeric(calobs), quantil5 =
Est.PosteriorMCP[,"quan2_5"], quantil95 = Est.PosteriorMCP[,"quan97_5"])
 # Uniformity Test calibration
 UnifTestCal <-ks.test(QQplot$zi,'punif')
 # Ho : data are from U(0,1)
 # Ha : data are not from U(0,1)
 # p-value : 0.8106 > 0.05: accept Ho, data from U(0,1)

 # validation period
```

```
  QQplotv <- QQtest(discharge = as.numeric(valobs), posterior = PostPredMCPval)
  NSEv <- Nash(obs = as.numeric(valobs), sim = EstPosteriorMCPval[,"median"])
  KGEv <- klinggupta(obs = as.numeric(valobs), sim = EstPosteriorMCPval[,"median"])
  CottonTest2v <- TestUncertBand(observations = as.numeric(valobs), quantil5 =
EstPosteriorMCPval[,"quan2_5"], quantil95 = EstPosteriorMCPval[,"quan97_5"])
  PPU95v <- UncertaintyBand95(observations = as.numeric(valobs), quantil5 =
EstPosteriorMCPval[,"quan2_5"], quantil95 = EstPosteriorMCPval[,"quan97_5"])
  # Uniformity Test calibration
  UnifTestVal <-ks.test(QQplotv$zi,'punif')

  return(list(posterior = PostPredMCP, NSE = NSE, KGE = KGE, UnifTest = UnifTestCal$p.value,
PoinsInsideBand = CottonTest2, Est.Posterior = Est.PosteriorMCP, PPU95 = PPU95, QQplot = QQplot,
posteriorv = PostPredMCPval, NSEv = NSEv, KGEv = KGEv, UnifTestv = UnifTestVal$p.value,
PoinsInsideBandv = CottonTest2v, Est.Posteriorv = EstPosteriorMCPval, PPU95v = PPU95v, QQplotv =
QQplotv))

}
```

 # MCP Gaussian Mixture Cluster Postprocessor (GMM) ----------

```
# This post-processor run in matlab so we load the predictive posterior  and compute statistics
setwd("F:/TesinaMasterCoDireccion"
)
# load posterior from GMM post-processor for calibration and validation period
load("POSTERIOR-CAL- VAL.RData")
# Save results of GMM post-processor in a new folder
setwd("F:/TesinaMasterCoDireccion/ResultGMM"
)

GMMPostEstMetrics <- function(calobs, calsim, valobs, valsim, posteriorcal, posteriorval){
  # Inputs: calobs, calsim, valobs, valsim are observations and simulations for calibration and validation
period respectively,  are time series object;
  #       posteriorcal: matrix posterior (posteior, time) for calibration, posteriorval for validation period
  # Outputs: statistics of posterior predictive  and metrics

  # compute statistiscs of posterior predictive calibration
  EstPosteriorcal <- Edit.Posterior(observations = as.numeric(calobs), posterior = posteriorcal)
  #save(Est.PosteriorGMM, file = "Est.PosteriorGMM.RData")
  # compute statistiscs of posterior predictive validation
  EstPosteriorval <- Edit.Posterior(observations = as.numeric(valobs), posterior = posteriorval)
  #save(EstPosteriorMCPval, file = "EstPosteriorMCPval.RData")

  #  Performance metrics
  QQplot <- QQtest(discharge = as.numeric(calobs), posterior = posteriorcal)
  NSE <- Nash(obs = as.numeric(calobs), sim = EstPosteriorcal[,"median"])
  KGE <- klinggupta(obs = as.numeric(calobs), sim = EstPosteriorcal[,"median"])
  CottonTest2 <- TestUncertBand(observations = as.numeric(calobs), quantil5 =
EstPosteriorcal[,"quan2_5"], quantil95 = EstPosteriorcal[,"quan97_5"])
  PPU95 <- UncertaintyBand95(observations = as.numeric(calobs), quantil5 =
EstPosteriorcal[,"quan2_5"], quantil95 = EstPosteriorcal[,"quan97_5"])
```

```
  # Uniformity Test calibration
  UnifTestCal <-ks.test(QQplot$zi,'punif')
  # Ho : data are from U(0,1)
  # Ha : data are not from U(0,1)
  # p-value : 0.8106 > 0.05: accept Ho, data from U(0,1)

  # validation period
  QQplotv <- QQtest(discharge = as.numeric(valobs), posterior =
posteriorval)
  NSEv <- Nash(obs = as.numeric(valobs), sim = EstPosteriorval[,"median"])
  KGEv <- klinggupta(obs = as.numeric(valobs), sim = EstPosteriorval[,"median"])
  CottonTest2v <- TestUncertBand(observations = as.numeric(valobs), quantil5 =
EstPosteriorval[,"quan2_5"], quantil95 = EstPosteriorval[,"quan97_5"])
  PPU95v <- UncertaintyBand95(observations = as.numeric(valobs), quantil5 =
EstPosteriorval[,"quan2_5"], quantil95 = EstPosteriorval[,"quan97_5"])
  # Uniformity Test calibration
  UnifTestVal <-ks.test(QQplotv$zi,'punif')

  return(list(posterior = as.matrix(posteriorcal), NSE = NSE, KGE = KGE, UnifTest = UnifTestCal$p.value,
PoinsInsideBand = CottonTest2, EstPosterior = EstPosteriorcal, PPU95 = PPU95, QQplot = QQplot,
posteriorv = as.matrix(posteriorval), NSEv = NSEv, KGEv = KGEv, UnifTestv = UnifTestVal$p.value,
PoinsInsideBandv = CottonTest2v, EstPosteriorv = EstPosteriorval, PPU95v = PPU95v, QQplotv =
QQplotv))
}
```