ANEXO A: CÓDIGOS DE PROGRAMACIÓN

A.1. CÓDIGO EN R PARA EJECUTAR EL MODELO GR4H

```
# GR4H run model
# By Harold Llauca
run_gr4h <- function(ini,
                     end,
                     pr,
                     pe,
                     param,
                     area,
                     inistate=NULL){

  require(airGR)
  require(lubridate)

  # Prepare database
  Time   <- seq(as.POSIXct(ini, "GMT", format="%Y-%m-%d %H:%M"),
                as.POSIXct(end, "GMT", format="%Y-%m-%d %H:%M"),
                by='hours')
  Inputs <- data.frame(DatesR=Time,
                       P=pr[,-1],
                       E=pe[,-1])

  # Run model
  ntim <- nrow(Inputs)
  nsub <- length(area)
  run  <- list()
  for(w in 1:nsub){

    # Prepare forcing data
    InputsModel <- CreateInputsModel(FUN_MOD=RunModel_GR4H,
                                     DatesR=Inputs$DatesR,
                                     Precip=Inputs[,1+w],
                                     PotEvap=Inputs[,1+nsub+w])

    # Model options
    RunOptions  <- CreateRunOptions(FUN_MOD=RunModel_GR4H,
                                    InputsModel=InputsModel,
                                    IndPeriod_Run=1:ntim,
                                    IniStates=inistate,
                                    verbose=FALSE,
                                    warnings=FALSE)

    # Run model
    OutputsModel <- RunModel(InputsModel=InputsModel,
                             RunOptions=RunOptions,
                             Param=param,
                             FUN=RunModel_GR4H)

    run[[w]] <- as.vector(OutputsModel$Qsim*area[w]/3.6)
  }
  qsim <- round(do.call(cbind, run),3) #in cms
  ans  <- list(Time=Time,
               Qsim=qsim)
  return(ans)
} # end (not run)
```

A.2. CÓDIGO EN R PARA EJECUTAR EL MODELO GR4H CON EL FILTRO DE KALMAN DE CONJUNTOS

```
run_enkf_gr4h <- function(DatesR,
                          Precip,
                          PotEvap,
                          InputsPert,
                          Qobs,
                          Area,
                          IndRun,
                          Param,
                          StateEnKF=c("Prod", "Rout", "UH1", "UH2"),
                          StatePert=c("Prod", "Rout", "UH1", "UH2")) {


  require(airGR)
  require(tictoc)
  tic()

  # Inputs model
  InputsModel <- CreateInputsModel(FUN_MOD=RunModel_GR4H,
                                   DatesR=DatesR,
                                   Precip=Precip,
                                   PotEvap=PotEvap)


  # ------ Checks
  TimeUnit   <- "hourly"
  StateNames <- c("Prod", "Rout", "UH1", "UH2")

  if (length(InputsPert$DatesR) != length(InputsModel$DatesR)) {
      stop("'InputsPert' elements must have the same length as the
'InputsModel'elements")
  }

  ClassInputsModel <- class(InputsModel)
  ClassInputsPert  <- class(InputsPert)[-grep("InputsPert", class(InputsPert))]
  ClassDiffInputsModel <- setdiff(ClassInputsModel, ClassInputsPert)
  ClassDiffInputsPert  <- setdiff(ClassInputsPert, ClassInputsModel)
  if (length(ClassDiffInputsModel) != 0 | length(ClassDiffInputsPert) != 0) {
      msgClassInputs <- "'InputsModel' and 'InputsPert' classes are not consistent:"
    if (length(ClassDiffInputsModel) != 0) {
          msgClassInputs <- sprintf("%s\n\tInputsModel: %s", msgClassInputs,
paste(dQuote(ClassDiffInputsModel), collapse = "\t"))
        }
        if (length(ClassDiffInputsPert) != 0) {
          msgClassInputs <- sprintf("%s\n\tInputsPert:  %s", msgClassInputs,
paste(dQuote(ClassDiffInputsPert), collapse = "\t"))
        }
        stop(msgClassInputs)
    }

  NbMbr <- InputsPert$NbMbr
  if (!(is.atomic(NbMbr) && is.numeric(NbMbr) && length(NbMbr) == 1 && NbMbr >= 2)) {
    stop("'NbMbr' should be a single vector of a numeric value >= 2")
  } else {
      NbMbr <- as.integer(NbMbr)
    }


  # ------ Settings

  # data assimilation method used (not open-loop simulation)
  NbTime <- length(IndRun)

  NbState <- length(StateNames)

  Qobs <- Qobs*3.6/Area
  Qobs[Qobs < 0] <- NaN
  VarThr <- quantile(Qobs, probs = 0.1, na.rm = TRUE)

  # member names
  MbrNames <- sprintf("Mbr_%s", seq_len(NbMbr))

  # time names
  TimeNames <- sprintf("Time_%s", seq_len(NbTime))
```

```r
# InputsModel
InputsModel <- InputsModel[IndRun]

# InputsPert
InputsPert <- InputsPert[IndRun]

# Qobs
Qobs <- Qobs[IndRun]



# ------ Ensemble initializations

ObsPert <- matrix(data = NA,
                  nrow = NbMbr,
                  ncol = NbTime,
                  dimnames = list(MbrNames,
                                  TimeNames))

QsimEns <- ObsPert

IniStatesEns   <- list()
IniStatesEnsNbTime <- list()

EnsStateBkg <- array(data = rep(NaN, times = NbState*NbMbr*NbTime),
                     dim = c(NbState, NbMbr, NbTime),
                     dimnames = list(StateNames,
                                     MbrNames,
                                     TimeNames))

EnsStateA <- EnsStateBkg

ItAssim <- 0

# fake RunOptions
RunOptionsIni <- airGR::CreateRunOptions(FUN_MOD = RunModel_GR4H,
                                         InputsModel = InputsModel,
                                         IndPeriod_Run = 1L,
                                         warning = FALSE, verbose = FALSE)
RunOptionsIter <- airGR::CreateRunOptions(FUN_MOD = RunModel_GR4H,
                                          InputsModel = InputsModel,
                                          IndPeriod_Run = 1L,
                                          IndPeriod_WarmUp = 0L,
                                          IniStates = NULL,
                                          warning = FALSE, verbose = FALSE)


# ------ Run

if (is.null(InputsPert$Precip)) {
  InputsPert$Precip <- replicate(n = ncol(InputsPert$PotEvap),
                                 expr = InputsModel$Precip)
  dimnames(InputsPert$Precip) <- dimnames(InputsPert$PotEvap)
}
if (is.null(InputsPert$PotEvap)) {
  InputsPert$PotEvap <- replicate(n = ncol(InputsPert$Precip),
                                  expr = InputsModel$PotEvap)
  dimnames(InputsPert$PotEvap) <- dimnames(InputsPert$Precip)
}

for (iTime in seq_along(IndRun)) {

  for (iMbr in seq_len(NbMbr)) {

    if (iTime == 1) { # default (one year by default) warmup

      RunOptionsIni$IndPeriod_Run <- iTime
      OutputsModel <- RunModel_GR4H(InputsModel = InputsModel,
                                    RunOptions = RunOptionsIni,
                                    Param = Param)

    } else { # IF iTime > 1

      IniStates <- IniStatesEns[[iMbr]]
      IniStates$Store$Rest <- rep(NA, times = 3)
      IniStates <- unlist(IniStates)
```

```
    IniStates[is.na(IniStates)] <- 0
    RunOptionsIter$IniStates <- IniStates
    RunOptionsIter$IniResLevels <- NULL

    # definition of run options
    InputsPertMbr <- InputsPert
    InputsPertMbr$Precip  <- InputsPert$Precip[, iMbr]
    InputsPertMbr$PotEvap <- InputsPert$PotEvap[, iMbr]
    RunOptionsIter$IndPeriod_Run <- as.integer(iTime)
    InputsModel  <- InputsPertMbr
    OutputsModel <- RunModel_GR4H(InputsModel = InputsModel,
                                  RunOptions = RunOptionsIter,
                                  Param = Param)
  } # END IF(t == 1)

  IniStatesEns[[iMbr]] <- OutputsModel$StateEnd
  names(IniStatesEns)[iMbr] <- sprintf("Mbr_%s", iMbr)

  EnsStateBkg["Prod", iMbr, iTime] <- OutputsModel$Prod
  EnsStateBkg["Rout", iMbr, iTime] <- OutputsModel$Rout
  EnsStateBkg["UH2", iMbr, iTime]  <- OutputsModel$StateEnd$UH$UH2[1]
  EnsStateBkg["UH1", iMbr, iTime]  <- OutputsModel$StateEnd$UH$UH1[1]

  QsimEns[iMbr, iTime]  <- OutputsModel$Qsim

} # END FOR iMbr


# ------ Assimilation [if an observation is available]

if (is.finite(Qobs[iTime])) {

  ItAssim <- ItAssim + 1

    ans <- DA_EnKF(Obs = Qobs[iTime],
                   Qsim = QsimEns[, iTime],
                   EnsState = EnsStateBkg[, , iTime],
                   Param = Param,
                   StateNames = StateNames,
                   StatePert = StatePert,
                   NbMbr = NbMbr,
                   StateEnKF = StateEnKF,
                   VarThr = VarThr)

    for (iMbr in seq_len(NbMbr)) {
      IniStatesEns[[iMbr]]$Store$Prod <- ans$EnsStateEnkf["Prod", iMbr]
      IniStatesEns[[iMbr]]$Store$Rout <- ans$EnsStateEnkf["Rout", iMbr]
      IniStatesEns[[iMbr]]$UH$UH2[1]  <- ans$EnsStateEnkf["UH2" , iMbr]
      IniStatesEns[[iMbr]]$UH$UH1[1]  <- ans$EnsStateEnkf["UH1", iMbr]
    }

    if (iTime < NbTime) {
      IniStatesEnsNbTime[[iTime+1]] <- IniStatesEns
      names(IniStatesEnsNbTime)[iTime+1] <- sprintf("Time_%s",iTime+1)
    }

    if (!is.null(StatePert)) {
      for (iMbr in seq_len(NbMbr)) {
        IniStatesEns[[iMbr]]$Store$Prod <- ans$EnsStatePert["Prod", iMbr]
        IniStatesEns[[iMbr]]$Store$Rout <- ans$EnsStatePert["Rout", iMbr]
        IniStatesEns[[iMbr]]$UH$UH2[1]  <- ans$EnsStatePert["UH2" , iMbr]
        IniStatesEns[[iMbr]]$UH$UH1[1]  <- ans$EnsStatePert["UH1", iMbr]
      }
    }

    EnsStateA[, , iTime] <- ans$EnsStateEnkf

    if (iTime < NbTime) {
      if (!is.null(StatePert)) {
        EnsStateBkg[, , iTime+1] <- ans$EnsStatePert
      } else {
        EnsStateBkg[, , iTime+1] <- ans$EnsStateEnkf
      }
    }
    ObsPert[, iTime] <- ans$ObsPert

}else { # IF no assimilation
```

```
      if (iTime < NbTime) {
        IniStatesEnsNbTime[[iTime+1]] <- IniStatesEns
        names(IniStatesEnsNbTime)[iTime+1] <- sprintf("Time %s", iTime+1)
        EnsStateBkg[, , iTime+1] <- EnsStateBkg[, , iTime]
      }

      EnsStateA[, , iTime] <- EnsStateBkg[, , iTime]
      ObsPert[, iTime] <- rep(Qobs[iTime], times = NbMbr)

    } # END IF assimilation

  } # END FOR time


  # ------ Outputs and class
  QsimEns <- QsimEns*Area/3.6
  Qobs    <- Qobs*Area/3.6
  res <- list(DatesR=InputsModel$DatesR,
              QsimEns=t(QsimEns),
              Qobs=Qobs,
              EnsStateBkg=aperm(EnsStateBkg),
              EnsStateA=aperm(EnsStateA),
              NbTime=NbTime,
              NbMbr=NbMbr,
              NbState=NbState)
  class(res) <- c("OutputsModelDA", "OutputsModel", "EnKF", TimeUnit)
  toc()
  return(res)
}
```

A.3. CÓDIGO EN R PARA EJECUTAR EL MODELO GR4H CON EL FILTRO DE PARTÍCULAS

```
run_pf_gr4h <- function(DatesR,
                        Precip,
                        PotEvap,
                        InputsPert,
                        Qobs,
                        Area,
                        IndRun,
                        Param,
                        StatePert=c("Prod", "Rout", "UH1", "UH2")) {


  # DatesR=time
  # Precip=pr
  # PotEvap=pe
  # InputsPert=InputsPert
  # Qobs=qobs
  # Area=area
  # IndRun=seq_along(time)
  # Param=param
  # StatePert=c("Prod", "Rout")

  require(airGR)
  require(tictoc)
  tic()

  # Inputs model
  InputsModel <- CreateInputsModel(FUN_MOD=RunModel_GR4H,
                                   DatesR=DatesR,
                                   Precip=Precip,
                                   PotEvap=PotEvap)

  # ------ Checks
  TimeUnit   <- "hourly"
  StateNames <- c("Prod", "Rout", "UH1", "UH2")

  if (length(InputsPert$DatesR) != length(InputsModel$DatesR)) {
      stop("'InputsPert' elements must have the same length as the
'InputsModel'elements")
  }

  ClassInputsModel <- class(InputsModel)
  ClassInputsPert  <- class(InputsPert)[-grep("InputsPert", class(InputsPert))]
  ClassDiffInputsModel <- setdiff(ClassInputsModel, ClassInputsPert)
  ClassDiffInputsPert  <- setdiff(ClassInputsPert, ClassInputsModel)
  if (length(ClassDiffInputsModel) != 0 | length(ClassDiffInputsPert) != 0) {
      msgClassInputs <- "'InputsModel' and 'InputsPert' classes are not consistent:"
    if (length(ClassDiffInputsModel) != 0) {
          msgClassInputs <- sprintf("%s\n\tInputsModel: %s", msgClassInputs,
paste(dQuote(ClassDiffInputsModel), collapse = "\t"))
        }
        if (length(ClassDiffInputsPert) != 0) {
          msgClassInputs <- sprintf("%s\n\tInputsPert:  %s", msgClassInputs,
paste(dQuote(ClassDiffInputsPert), collapse = "\t"))
        }
        stop(msgClassInputs)
    }

  NbMbr <- InputsPert$NbMbr
  if (!(is.atomic(NbMbr) && is.numeric(NbMbr) && length(NbMbr) == 1 && NbMbr >= 2)) {
    stop("'NbMbr' should be a single vector of a numeric value >= 2")
  } else {
      NbMbr <- as.integer(NbMbr)
    }


  # ------ Settings

  # data assimilation method used (not open-loop simulation)
  NbTime <- length(IndRun)

  NbState <- length(StateNames)
```

```
Qobs <- Qobs*3.6/Area
Qobs[Qobs < 0] <- NaN
VarThr <- quantile(Qobs, probs = 0.1, na.rm = TRUE)

# member names
MbrNames <- sprintf("Mbr_%s", seq_len(NbMbr))

# time names
TimeNames <- sprintf("Time_%s", seq_len(NbTime))

# InputsModel
InputsModel <- InputsModel[IndRun]

# InputsPert
InputsPert <- InputsPert[IndRun]

# Qobs
Qobs <- Qobs[IndRun]



# ------ Ensemble initializations

ObsPert <- matrix(data = NA,
                  nrow = NbMbr,
                  ncol = NbTime,
                  dimnames = list(MbrNames,
                                  TimeNames))

QsimEns <- ObsPert

IniStatesEns   <- list()
IniStatesEnsNbTime <- list()

EnsStateBkg <- array(data = rep(NaN, times = NbState*NbMbr*NbTime),
                     dim = c(NbState, NbMbr, NbTime),
                     dimnames = list(StateNames,
                                     MbrNames,
                                     TimeNames))

EnsStateA <- EnsStateBkg

ItAssim <- 0

# fake RunOptions
RunOptionsIni <- airGR::CreateRunOptions(FUN_MOD = RunModel_GR4H,
                                         InputsModel = InputsModel,
                                         IndPeriod_Run = 1L,
                                         warning = FALSE, verbose = FALSE)
RunOptionsIter <- airGR::CreateRunOptions(FUN_MOD = RunModel_GR4H,
                                          InputsModel = InputsModel,
                                          IndPeriod_Run = 1L,
                                          IndPeriod_WarmUp = 0L,
                                          IniStates = NULL,
                                          warning = FALSE, verbose = FALSE)


# ------ Run

if (is.null(InputsPert$Precip)) {
  InputsPert$Precip <- replicate(n = ncol(InputsPert$PotEvap),
                                 expr = InputsModel$Precip)
  dimnames(InputsPert$Precip) <- dimnames(InputsPert$PotEvap)
}
if (is.null(InputsPert$PotEvap)) {
  InputsPert$PotEvap <- replicate(n = ncol(InputsPert$Precip),
                                  expr = InputsModel$PotEvap)
  dimnames(InputsPert$PotEvap) <- dimnames(InputsPert$Precip)
}

for (iTime in seq_along(IndRun)) {

  for (iMbr in seq_len(NbMbr)) {

    if (iTime == 1) { # default (one year by default) warmup

      RunOptionsIni$IndPeriod_Run <- iTime
```

```
      OutputsModel <- RunModel_GR4H(InputsModel = InputsModel,
                                    RunOptions = RunOptionsIni,
                                    Param = Param)

    } else { # IF iTime > 1

      IniStates <- IniStatesEns[[iMbr]]
      IniStates$Store$Rest <- rep(NA, times = 3)
      IniStates <- unlist(IniStates)
      IniStates[is.na(IniStates)] <- 0
      RunOptionsIter$IniStates <- IniStates
      RunOptionsIter$IniResLevels <- NULL

      # definition of run options
      InputsPertMbr <- InputsPert
      InputsPertMbr$Precip  <- InputsPert$Precip[, iMbr]
      InputsPertMbr$PotEvap <- InputsPert$PotEvap[, iMbr]
      RunOptionsIter$IndPeriod_Run <- as.integer(iTime)
      InputsModel  <- InputsPertMbr
      OutputsModel <- RunModel_GR4H(InputsModel = InputsModel,
                                    RunOptions = RunOptionsIter,
                                    Param = Param)
    } # END IF(t == 1)

    IniStatesEns[[iMbr]] <- OutputsModel$StateEnd
    names(IniStatesEns)[iMbr] <- sprintf("Mbr_%s", iMbr)

    EnsStateBkg["Prod", iMbr, iTime] <- OutputsModel$Prod
    EnsStateBkg["Rout", iMbr, iTime] <- OutputsModel$Rout
    EnsStateBkg["UH2", iMbr, iTime]  <- OutputsModel$StateEnd$UH$UH2[1]
    EnsStateBkg["UH1", iMbr, iTime]  <- OutputsModel$StateEnd$UH$UH1[1]

    QsimEns[iMbr, iTime]  <- OutputsModel$Qsim

  } # END FOR iMbr


  # ------ Assimilation [if an observation is available]

  if (is.finite(Qobs[iTime])) {

    ItAssim <- ItAssim + 1

      ans <- DA_PF(Obs = Qobs[iTime],
                   Qsim = QsimEns[, iTime],
                   States = IniStatesEns,
                   Param = Param,
                   StateNames = StateNames,
                   NbMbr = NbMbr,
                   StatePert = StatePert,
                   VarThr = VarThr)

      if (!is.null(StatePert)) {
        IniStatesEns <- ans$EnsStatePert
      } else {
        IniStatesEns <- ans$EnsStatePf
      }

      EnsStateA["Prod", , iTime] <- sapply(seq_along(ans$EnsStatePf), function(x)
ans$EnsStatePf[[x]]$Store$Prod)
      EnsStateA["Rout", , iTime] <- sapply(seq_along(ans$EnsStatePf), function(x)
ans$EnsStatePf[[x]]$Store$Rout)
      EnsStateA["UH2" , , iTime] <- sapply(seq_along(ans$EnsStatePf), function(x)
ans$EnsStatePf[[x]]$UH$UH2[1])
      EnsStateA["UH1", , iTime]  <- sapply(seq_along(ans$EnsStatePf), function(x)
ans$EnsStatePf[[x]]$UH$UH1[1])

      if (iTime < NbTime) { # olivier?
        IniStatesEnsNbTime[[iTime+1]] <- ans$EnsStatePf
        names(IniStatesEnsNbTime)[iTime+1] <- sprintf("Time_%s", iTime+1)
      }

  }else { # IF no assimilation

    if (iTime < NbTime) {
      IniStatesEnsNbTime[[iTime+1]] <- IniStatesEns
      names(IniStatesEnsNbTime)[iTime+1] <- sprintf("Time_%s", iTime+1)
```

```
        EnsStateBkg[, , iTime+1] <- EnsStateBkg[, , iTime]
      }
      EnsStateA [, , iTime] <- EnsStateBkg[, , iTime]

    } # END IF assimilation

  } # END FOR time


  # ------ Outputs and class
  QsimEns <- QsimEns*Area/3.6
  Qobs    <- Qobs*Area/3.6
  res <- list(DatesR=InputsModel$DatesR,
              QsimEns=t(QsimEns),
              Qobs=Qobs,
              EnsStateBkg=aperm(EnsStateBkg),
              EnsStateA=aperm(EnsStateA),
              NbTime=NbTime,
              NbMbr=NbMbr,
              NbState=NbState)
  class(res) <- c("OutputsModelDA", "OutputsModel", "PF", TimeUnit)
  toc()
  return(res)
}
```