```r
########################################
#Funktion für Monte carlo Runs mit Hydrus
#parallelisiert
########################################

mc_parallel2<-function(nr=100,#Anzahl Modellläufe
                       #Parameter Ranges
                       ranges,
                       #Parameter die nicht variiert werden
                       fixed,
                       #wie oft soll das Modell parallel gerechnet werden
                       n_parallel=20,
                       #soll UNSATCHEM verwendet werden
                       UNSAT=T,
                       #maximale Wartezeit nach der das Modell abgebrochen wird
                       sleep=5,
                       #Tiefen die Benutzt werden um Objective Function zu ermitteln
                       fit.tiefe=c(-2,-6,-10,-14),
                       #soll die lower Boundary free drain verwendet werden
                       free_drain=T,# wenn False wird seepage face verwendet
                       #soll  die Objective Function auch für Ca gefittet werden
                       fit.calcium=T,
                       #Anzahl Knoten
                       n_nodes=9,
                       #Verteilung des Bodenmaterials
                       Mat=c(rep(1,3),rep(2,5),3),
                       #Anzahl Print times
                       print_times=100,
                       #maximaler Zeitschritt
                       dtmax=10,
                       #Länge der Warm-up-Periode in Minuten
                       traintime=4500,
                       #soll 'kinetic solution' verwendet werden?
                       kin_sol=T,
                       #wenn recalc = T werden die nicht konvergierten Modellruns
                       #mit niedrigerem dtmax erneut berechnet.
                       recalc=T,
                       #minimal akzeptierte Zeilenanzahl des Modelloutputs
                       #die akzeptiert wird
                       #wenn das Modell nicht vollstädig durchgelaufen ist
                       #wird sonst der RMSE berechnet obwohl
                       #möglicherweise nicht alle Intensitäten repräsentiert werden
                       min_nrows=2500,
                       #welche Messungen werden als Referenz verwendet
                       obs=all_s){
  #Startzeit wird gespeichert um später die Gesamtzeit des MC-laufs ausgeben zu können
  starttime<-Sys.time()

  #Rscript mit Hydrus Functionen ausführen
  #in diesem Skript werden die Funktionen
  #atmos.in, profile.in, selector.in, hydrus.exe,
  #read_hydrus.out und read_conc.out definiert
  source("//FUHYS013/Freiberg/rcode/modellierung/hydrus_input.R")
```

```
  #M als Anzahl der Parameter
  M<-ncol(ranges)
  #für die EE Funktion muss nr  =  r*(M+1) also wird nr angepasst
  r<-round(nr/(M+1))
  nr<-r*(M+1)
  #für die OAT Funktion müssen Parameterranges als Liste vorliegen
  distr_par<-as.list(ranges)
  #Paramatersätze mit  OAT-sampling und Latin Hypercube Sampling (lhs) ziehen
  par<-SAFER::OAT_sampling(r=r,M=M,distr_fun = "unif",distr_par = distr_par,
                           samp_strat = "lhs",des_type = "radial")

  #Parametersätze als data.frame und mit Parameternamen die übergeben wurden
  par<-as.data.frame(par)
  colnames(par)<-colnames(ranges)

#lade Datensatz all.R
load("//FUHYS013/Freiberg/daten/all.R")
#tmax ist die Zeitdifferenz vom ersten zum letzten Messwert in Minuten
tmax<-as.numeric(difftime(max(obs$date),min(obs$date),units = "min"))

#Vektoren für die parallelisiert aufgerufenen Hydrus-Ordner und Dateinamen
file<-paste0("UNSC",1:n_parallel)
projektpfad<-paste0("//FUHYS013/Freiberg/Hydrus/UNSC",1:n_parallel,"/")
programmpfad<-paste0("//FUHYS013/Freiberg/programme/Hydrus-1D_4-",1:n_parallel,"/")

#falls Hydrus gerade noch ausgeführt wird wird es jetzt gestoppt,
#da sonst die Inputdateien nicht bearbeitet werden können
system("taskkill /IM H1D_UNSC.EXE",show.output.on.console=F)
Sys.sleep(2)

#Schleife um inputs für alle parallelisierten Ordner zu schreiben
for (i in 1:n_parallel){
  #Atmospärischen Input mit atmos.in Funktion definieren
  atmos.in(obs=obs,
           total_t = tmax,
           projektpfad = projektpfad[i],mainpath ="//FUHYS013/Freiberg/")

  #Bodenprofil im Modell mittels profile.in Funktion anpassen
  profile.in(projektpfad = projektpfad[i],
             Mat = Mat,n_nodes = n_nodes,th=seq(0.2,0.4,len=n_nodes))
}

#Vektoren für RMSE & NSE anlegen
rmse<-rep(NA,nr)
rmse_ca<-rep(NA,nr)
rmse_both<-rep(NA,nr)
nse<-rep(NA,nr)

print("start of mc loop")
#Monte-Carlo-Schleife mit nr Durchläufen in n_parallel Abständen,
#da bei jedem i n_parallel Modellläufe gerechnet werden
for (i in seq(1,nr,n_parallel)){
```

```
  #zweite Schleife für parallelisiertes ausführen des Modells
  for (j in 1:n_parallel){
    #falls die Anzahl MC-Läufe nicht durch n_parallel teilbar ist
    #werden die überschüssigen parallelisierten Läufe nicht durchgeführt
    if(nrow(par)>=(i+j-1)){

      #Parametersatz i+j-1 mit den fix-Parametern zusammenfügen
      pars<-cbind(par[(i+j-1),],fixed)

      #Den i-ten Parametersatz mit selector.in Funktion dem Modell übergeben
      selector.in(params = pars,
                  projektpfad = projektpfad[j],
                  tmax=tmax,
                  UNSC = UNSAT,
                  free_drain=free_drain,
                  print_times = print_times,
                  dtmax = dtmax,
                  kin_sol = kin_sol)


      #Hydrus mittels hydrus.exe Funktion ausführen
      hydrus.exe(file = file[j],UNSC=UNSAT,hide_hydrus = T,
                 programmpfad = programmpfad[j],wait = T,
                 scriptpath = "//FUHYS013/Freiberg/Hydrus/")
    }#Ende if Schleife
  }#Ende Parallelisierungsschleife

  #kurz verschnaufen
  Sys.sleep(1)

  #interne Funktion um CPU der Prozesse abzufragen
  #dabei gibt "sleep" die maximale Rechenzeit an die Hydrus gegeben wird
  check_CPU<-function(sleep2=sleep){

    #Cmd-line Abfrage für Liste aller tasks mit CPU Angabe
    tasklist<-system(
  "wmic path Win32_PerfFormattedData_PerfProc_Process get Name,PercentProcessorTime",
      intern=T)
    #diese Liste an Stellen mit mindestens zwei Leerzeichen zerschneiden
    tasksplit<-strsplit(tasklist[2:(length(tasklist)-1)]," \\s+")
    #Listenelemente aneinander hängen
    tasks<-do.call("rbind",tasksplit)
    #aktuelle Uhrzeit speichern
    startpoint<-Sys.time()

    #wenn in der taskliste H1D_UNSC vorkommt, also Hydrus gerade ausgeführt wird...
    if(length(grep("H1D_UNSC",tasks))>0){

      #...while-Schleife starten in der abgefragt wird ob Hydrus noch
      #mehr als 2 mal eine CPU über 0 braucht
      #und ob die Schleife schon länger läuft als die maximal erlaubte Zeit
      while(length(which(tasks[grep("H1D_UNSC",tasks),2]>0))>2&
            as.numeric(difftime(Sys.time(),startpoint,units = "sec"))<=sleep2){
```

```
      #kurz verschnaufen
      Sys.sleep(0.1)
      #Taskliste aktualisieren
      tasklist<-system(
  "wmic path Win32_PerfFormattedData_PerfProc_Process get Name,PercentProcessorTime",
        intern=T)
      tasksplit<-strsplit(tasklist[2:(length(tasklist)-1)]," \\s+")
      tasks<-do.call("rbind",tasksplit)
    }#Ende der while-Schleife
    #Zeit die gewartet wurde ausgeben
    print(difftime(Sys.time(),startpoint,units = "sec"))
    }#Ende der if-Schleife
}#Ende der check CPU-Funktion

#CPU checken
check_CPU()
#wenn CPU auf 0 ist oder Zeit überschritten ist wird das Modell beendet
system("taskkill /IM H1D_UNSC.EXE",show.output.on.console=F)

#da immer wieder ein Fehlerfenster auftritt das man wegklicken muss
#wird hier geprüft ob das Fenster schon wieder da ist
fault_check<-shell('tasklist /FI "IMAGENAME eq WerFault.exe"',intern = T)
#wenn es da ist...
if(length(grep("INFORMATION",fault_check))==0){
  #wird es einfach geschlossen
  system("taskkill /IM WerFault.exe",show.output.on.console=F)}

#eventuell nicht mehr nötig aber Redundanz schadet nie:
#also nochmal checken ob Hydrus noch offen ist, da sonst die Funktion abbricht
exe_check<-shell('tasklist /FI "IMAGENAME eq H1D_UNSC.EXE"',intern = T)

#wenn es offen ist dann wird es jetzt geschlossen
while(length(grep("INFORMATION",exe_check))==0){
  Sys.sleep(0.01)
  exe_check<-shell('tasklist /FI "IMAGENAME eq H1D_UNSC.EXE"',intern = T)
  system("taskkill /IM H1D_UNSC.EXE",show.output.on.console=F)
}

#parallelisierte Schleife um in allen Hydrus-Ordnern die Outputs einzulesen
#und den Modellfit zu berechnen
for (j in 1:n_parallel){
  #falls die Anzahl MC-Läufe nicht durch n_parallel teilbar ist
  #werden die überschüssigen parallelisierten Läufe nicht durchgeführt
  if(length(rmse)>=(i+j-1)){

    #wenn auch für Calcium gefittet werden soll
    if(fit.calcium==T){
      #Den Calcium Output mittels read_conc.out einlesen
      outca<-read_conc.out(projektpfad = projektpfad[j],obs=obs,min_nrows=min_nrows)
    }else{
      #sonst NA übergeben
      outca<-list(NA,NA,NA,NA)
    }
```

```
      #Den CO2 Output mittels read_hydrus.out einlesen
      out<-read_hydrus.out(projektpfad=projektpfad[j],
                           UNSC=UNSAT,fit.tiefe = fit.tiefe,
                           traintime=traintime,min_nrows=min_nrows,obs=obs)

      #Die Funktionen übergeben eine Liste
      #das erste Listenelement enthält die gemessenen und modellierten Ganglinien

      ca_vals<-outca[[1]]
      co2_vals<-out[[1]]

      #das dritte Listenelement enthält den NSE
      nse[(i+j-1)]<-out[[3]]

      #mit den outputs wird der normierte RMSE berechnet
      #checken ob die outputs nicht NAs sind
      if(!is.na(out[[2]])&!is.na(outca[[2]])){
        #RMSE norm durch teilen durch sd der Messungen berechnen
        #und dann den Mittelwert von CO2 und Calcium RMSE bilden
        rmse_both[(i+j-1)]<-
          (out[[2]]/sd(co2_vals$CO2_raw,na.rm = T)+
             outca[[2]]/sd(ca_vals$ca_conc,na.rm = T))/2
        }#Ende der if-Schleife

      #das zweite Listenelement enthält den RMSE
      rmse[(i+j-1)]<-out[[2]]
      rmse_ca[(i+j-1)]<-outca[[2]]
      nse[(i+j-1)]<-out[[3]]
    }#Ende if length rmse
    }#Ende for j-parallel

  #Fortschritt der Schleife ausgeben
  print(paste(i/nr*100,"%"))
  #RMSE Werte der parallelen Modellläufe ausgeben
  print(rmse[i:(i+n_parallel-1)])
  #speichern der Daten als Liste falls später ein Fehler auftritt
  mc<-list(rmse,par,nse,rmse_ca,rmse_both)
  save(mc,file="//FUHYS013/Freiberg/Hydrus/montecarlo/mc_temp.R")

  #Müllabfuhr
  gc()
}#Ende Monte-Carlo-Schleife

################################################
#langsamer werden wenn NAs auftraten
################################################

#Faktor um den die Wartezeit bei check CPU verlängert wird
sleep_fac<-1
#neuer dtmax
dtmax2<-dtmax
#Faktor um den dtmax verringert wird
dtmax_fac<-1
```

```
#while-Schleife in der abgefragt wird ob
#NAs im RMSE-Vektor vorkommen
#dtmax2 größer gleich 0.1 ist
#und ob dtmax überhaupt verkleinert werden soll (recalc=T)
while(is.na(mean(rmse))&dtmax2>=0.1&recalc==T){
  #der Faktor dtmax_fac wird erhöht
  dtmax_fac<-10*dtmax_fac
  #dtmax2 wird durch dtmax_fac geteilt
  dtmax2<-  dtmax2/dtmax_fac

  #alle NAs werden aus RMSE-vektoren ausgeschnitten
  rmse_na<-rmse[is.na(rmse)]
  rmse_ca_na<-rmse_ca[is.na(rmse)]
  rmse_both_na<-rmse_both[is.na(rmse)]

  #Ausgabe wie viele NAs nochmal berechnet werden
  print(paste("recalculating",length(rmse_na)," NA models with dtmax =",dtmax2))

  #alle NAs werden aus NSE-vektor ausgeschnitten
  nse_na<-nse[is.na(rmse)]

  #Monte-Carlo-schleife nochmal über NAs laufen lassen
  for (i in seq(1,length(rmse_na),n_parallel)){
    #Parallelisierungsschleife
    for (j in 1:n_parallel){
      #falls die Anzahl MC-Läufe nicht durch n_parallel teilbar ist
      #werden die überschüssigen parallelisierten Läufe nicht durchgeführt
      if(nrow(par[is.na(rmse),])>=(i+j-1)){
        #Parametersatz i mit den fix-Parametern zusammenfügen
        pars<-cbind(par[is.na(rmse),][(i+j-1),],fixed)

        #selector.in Funktion mit dem i-ten Parametersatz
        selector.in(params = pars,
                    projektpfad = projektpfad[j],
                    tmax=tmax,
                    UNSC = UNSAT,
                    free_drain=free_drain,
                    print_times = print_times,
                    dtmax = dtmax2)

        #hydrus ausführen
        hydrus.exe(file = file[j],UNSC=UNSAT,hide_hydrus = T,
                   programmpfad = programmpfad[j],wait = T,
                   scriptpath = "//FUHYS013/Freiberg/Hydrus/")
      }#Ende if nrow par
    }#Ende for j parallel

  Sys.sleep(1)
  #CPU checken
  check_CPU(sleep = sleep+15*sleep_fac)

  #wenn Modelle fertig sind oder Zeit überschritten ist wird Hydrus geschlossen
  system("taskkill /IM H1D_UNSC.EXE",show.output.on.console=F)
```

```r
#schauen ob die Fehlermeldung da ist
fault_check<-shell('tasklist /FI "IMAGENAME eq WerFault.exe"',intern = T)
#wenn ja die Meldung schließen
if(length(grep("INFORMATION",fault_check))==0){
  system("taskkill /IM WerFault.exe",show.output.on.console=F)}

#nochmal schauen ob hydrus wirklich zu ist
exe_check<-shell('tasklist /FI "IMAGENAME eq H1D_UNSC.EXE"',intern = T)
#wenn nein hydrus schließen
while(length(grep("INFORMATION",exe_check))==0){
  Sys.sleep(0.01)
  exe_check<-shell('tasklist /FI "IMAGENAME eq H1D_UNSC.EXE"',intern = T)
  system("taskkill /IM H1D_UNSC.EXE",show.output.on.console=F)
}

#Schleife um Modellfit zu berechnen
for (j in 1:n_parallel){
  if(length(rmse_na)>=(i+j-1)){

      #CO2 und Calcium-Werte mittels Funktionen einlesen
    if(fit.calcium==T){
      outca<-read_conc.out(projektpfad = projektpfad[j],obs=obs,min_nrows=min_nrows)
  }else{
    outca<-list(NA,NA,NA,NA)
  }
  out<-read_hydrus.out(projektpfad=projektpfad[j],
                       UNSC=UNSAT,fit.tiefe = fit.tiefe,
                       traintime=traintime,min_nrows=min_nrows,obs=obs)

  #RMSE norm berechnen
  ca_vals<-outca[[1]]
  co2_vals<-out[[1]]
  #schauen ob der RMSE der über die Funktionen berechnet wurde kein NA ist
  if(!is.na(out[[2]])&!is.na(outca[[2]])){
    #RMSE norm als mittel von CO2 und CA fit
    rmse_both_na[(i+j-1)]<-(out[[2]]/sd(co2_vals$CO2_raw,na.rm = T)+
                              outca[[2]]/sd(ca_vals$ca_conc,na.rm = T))/2
    }#Ende if

  #Objective Functions in Vektoren schreiben
  rmse_na[(i+j-1)]<-out[[2]]
  rmse_ca_na[(i+j-1)]<-outca[[2]]
  nse_na[(i+j-1)]<-out[[3]]
  }#Ende if length rmse
}#Ende for j parallel

#Fortschritt der NA-Schleife ausgeben
print(paste(i/length(rmse_na)*100,"%"))
#RMSE Werte ausgeben
print(rmse_na[i:(i+n_parallel-1)])
#speichern der Daten falls später ein Fehler auftritt
mc<-list(rmse,par,nse,rmse_ca,rmse_both)
save(mc,file="//FUHYS013/Freiberg/Hydrus/montecarlo/mc_temp.R")
```

```
    }#Ende Monte-Carlo NA-Schleife

    #sleep_fac um 2 erhöhen
    sleep_fac<-sleep_fac+2
    #die neu berechneten RMSE werte in die Stellen des RMSE-vektors schreiben an denen NAs waren
    rmse_ca[is.na(rmse)]<-rmse_ca_na
    rmse_both[is.na(rmse)]<-rmse_both_na
    rmse[is.na(rmse)]<-rmse_na

    #dasselbe für NSE
    nse[is.na(nse)]<-nse_na
  }#Ende while NA schleife

  ##################################
  #Ergebnisse des MC-Laufs ausgeben
  ##################################

  #Output in Liste schreiben
  mc<-list(rmse,par,nse,rmse_ca,rmse_both)

  #falls mehr als 100 Modellläufe gemacht wurden
  if(nr>100){
    #eine Datei mit Uhrzeit und Datum im Namen speichern um überschreiben zu verhindern
    filename<-paste0("mc_",nr,"-",format(Sys.time(),"%m-%d_%H.%M"))
    save(mc,file = paste0(mcpfad,filename,".R"))
    #ausgeben welche Datei gespeichert wurde
    print(paste("saved file",filename))}

  #ausgeben wielange der MC-Lauf insgesamt gedauert hat
  print("calculation time:")
  print(Sys.time()-starttime)
  #ausgeben wieviel Prozent  nicht NAs  waren
  print(paste(length(which(!is.na(rmse)))/nr*100,"% succesfully calculated"))

  #Ausgabe der Parameter & entsprechenden Modellfits
  return(mc)
}#Ende
```