Efunctions.R

pancho

Tue Jul 10 14:38:59 2018

```
### functions

update.tree <- function(tree,t_spe,t_ext){
  wt = tree$wt
  E = tree$E
  ct = sum(wt)
  if(t_ext > ct){
    stop('Extinction beyond present!')
  }
  if(t_ext < t_spe){
    stop('Speciation after extinction!')
  }
  # speciation
  K = length(wt)
  k = length(wt[cumsum(wt) < t_spe])
  if((k+1)<K){
    lastbit = E[(k+1):(K-1)]
  }
  else{
    lastbit = NULL
  }
  E = c(E[0:k],1,lastbit)
  # (this can be witten in one line, is convinent?)
  if(k==0){
    wt = c(t_spe, wt[1]-t_spe, wt[2:K])
  }
  if(k > 0 & k < (K-1)){
    wt = c(wt[1:k], t_spe - sum(wt[1:k]), wt[k+1]-(t_spe - sum(wt[1:k])), wt[(k+2):K])
  }
  if(k == (K-1)){
    wt = c(wt[1:(K-1)],t_spe-sum(wt[1:(K-1)]),ct-t_spe)
  }
  #extinction
  K = length(wt)
  k = length(wt[cumsum(wt) < t_ext])
  if((k+1)<K){
    lastbit = E[(k+1):(K-1)]
  }
  else{
    lastbit = NULL
  }
  E = c(E[0:k],0,lastbit)
  # (this can be witten in one line, is convinent?)
  if(k==0){
    wt = c(t_ext, wt[1]-t_ext, wt[2:K])
  }
  if(k > 0 & k < (K-1)){
```

```
    wt = c(wt[1:k], t_ext - sum(wt[1:k]), wt[k+1]-(t_ext - sum(wt[1:k])), wt[(k+2):K])
  }
  if(k == (K-1)){
    wt = c(wt[1:(K-1)],t_ext-sum(wt[1:(K-1)]),ct-t_ext)
  }
  tree = list(wt=wt,E=E)
  return(tree)
}
sampprob <- function(t,s,mu,r){  ## equation (*)
  term1 = s*(1-exp(-mu*(r-t)))
  c = exp(-(s/mu)*exp(-mu*r))
  if(c==0){
    term2 = 0
  }else{
    term2 = c^{-exp(mu*t)+1}
  }
  f = term1*exp(-s*t)*term2
  return(f)
}

# simulation of missing part [NEED TO CLEAN AND SIMPLIFY]
sim.extinct <- function(tree,pars,model='dd',seed=0, adjustment=FALSE){
  if(seed>0) set.seed(seed)
  wt = tree$wt
  ct = sum(wt)
  tree$E = rep(1,(length(wt)-1))
  lambda0=pars[1]
  mu0=pars[2]
  K=pars[3]
  limit = lambda0*K/(lambda0-mu0)
  rs = dim = length(wt)
  if (limit < (dim-1)){
    print('parameters do not make sense, observed tree implies negative rates')
  }
  ms = NULL # missing species, for now we just add time. When we consider topology we do it with specie
  e.lims = NULL # limits on extinctions
  cbt = 0
  N = 2
  nm= 0 # number of missing species
  rprob = NULL # true probability of Missing|observed
  sprob = NULL # sampling probability of Missing|observed
  et = NULL # event type
  h = 1 # index to fill probabilities
  dif1 = vector(mode = 'numeric',length = dim)
  dif2 = vector(mode = 'numeric', length = dim)
  for(i in 1:dim){
    rs = rs-1 # this are the remaning speciations
    cwt = wt[i]
    cbt = sum(wt[0:(i-1)])
    key = 0
    last='nothing'
    while(key == 0){
      if(model == "dd"){  # diversity-dependence model
```

```
  lambda = max(0,lambda0 - (lambda0-mu0)*N/K)
  mu = mu0
  lambda = rep(lambda,N)
}
if(model == 'cr'){ # constant-rate model
  lambda = rep(lambda0,N)
  mu = rep(mu0,N)
}
s = sum(lambda)
if (s == 0){
  t.spe = Inf
}
else{
  t.spe = rexp(1,s)
}
if(nm > 0){ # if there are missing species simulate extinction times
  t.ext = vector(mode = 'numeric',length = nm)
  for(j in 1:nm){
    t.ext[j] = truncdist::rtrunc(1,'exp',a=0,b=(e.lims[j]-cbt),rate=mu)
  }
  extinctedone = which(t.ext == min(t.ext))
  t.ext = min(t.ext)
}
else{
  t.ext = Inf
}
if(t.ext ==Inf & t.spe == Inf){
  print('try another K!')
}
mint = min(t.spe,t.ext)
if(nm > 0){
  probs = vector(mode = 'numeric',length = nm)
  for(j in 1:nm){
    probs[j] = 1-truncdist::ptrunc(mint,'exp',a=0,b=(e.lims[j]-cbt),rate=mu)
  }
}
else{probs = 1}
if(mint < cwt){
  if(mint == t.spe & mint != t.ext){#speciation
    acep = runif(1)
    thre = pexp(ct-cbt,mu)
    if(acep < thre){
      ms = c(ms,cbt+t.spe)
      rprob[h] = dexp(x = t.spe, rate = (s+nm*mu))*(s/(s+nm*mu))
      sp = sampprob(t = mint,s = s,mu = mu,r = ct-cbt)#/integrate(sampprob,lower = 0, upper = ct-
      sprob[h] = prod(probs)*sp
      et[h] = 'speciation'
      h = h + 1
      nm = nm + 1 # number of missing species
      if((N + rs-1) < limit){
        e.lims = c(e.lims,ct)
      }
      else{
```

```
            e.lims = c(e.lims,sum(wt[1:(dim - (N+rs-floor(limit)-1))])) # cual es la interpretacion d
          }
          N = N+1
          #print(paste('at branching time',cbt+t.spe,'a missing species arises, resulting on',N,'curr
          last = 'speciation'
        }
        else{last = 'nothing'}
        cwt = cwt - t.spe
        cbt = cbt + t.spe
      }
      else{#extinction
        pickone = sample(1:nm,1)
        t_spe = ms[pickone]
        t_ext = cbt + t.ext
        tree = update.tree(tree,t_spe=t_spe,t_ext=t_ext)
        rprob[h] = dexp(x = mint, rate = (s+nm*mu))*(mu/(s+nm*mu))
        et[h] = 'extinction'
        probs = probs[-extinctedone]
        sprob[h] = prod(probs)*truncdist::dtrunc(mint,'exp',a=0,b=(e.lims[extinctedone]-cbt),rate=mu)
        ms = ms[-pickone]
        cwt = cwt - t.ext
        cbt = cbt + t.ext
        N = N-1
        h = h+1
        nm = nm - 1
        e.lims = e.lims[-extinctedone]
        last = 'extinction'
      }
    }
    else{
      key = 1
      rprob[h] = pexp(q = cwt, rate = (s+nm*mu0),lower.tail = FALSE)
      et[h] = 'nothing'
      sprob[h] = prod(probs)*(1 - integrate(Vectorize(sampprob),lower = 0, upper = cwt,s=s,mu=mu,r=ct
      h = h+1
      dif1[i] = cwt/wt[i]
      dif2[i] = (mint - cwt)/wt[i]
      if(adjustment & cwt<(mint - cwt) & last=='speciation'){ #Adjusting last speciation
        ms = ms[-length(ms)]
        e.lims = e.lims[-length(e.lims)]
        nm = nm - 1
        N = N-1
      }
    }
  }
  N= N+1
}
tree$rprob = rprob
tree$sprob = sprob
tree$et = et
tree$weight = prod(rprob)/prod(sprob)
logweight = log(rprob)-log(sprob)
tree$logweight = sum(logweight)
```

```
  E = tree$E
  n = c(2,2+cumsum(E)+cumsum(E-1))
  tree$n = n
  return(tree)
}
#negative logLikelihood of a tree
nllik.tree = function(pars,tree){
  b = c(pars[1],(pars[1]-pars[2])/pars[3],pars[2])
  ldt = tree$wt[length(tree$wt)]
  dt = tree$wt[1:(length(tree$wt)-1)]
  E = tree$E
  if(is.null(tree$n)){
    n = c(2,2+cumsum(E)+cumsum(E-1))
    tree$n = n
  }
  lastn = tree$n[length(tree$n)]
  n = tree$n[1:(length(tree$n)-1)]
  sigma = n*(b[1]-b[2]*n + b[3]) #n-dimentional
  lastsigma = lastn*(b[1]-b[2]*lastn + b[3])
  rho = pmax(b[1]*E-b[2]*n*E+b[3]*(1-E),0)
  l = -(sum(-sigma*dt+log(rho))-lastsigma*ldt)
  if(min(b)<0){l = Inf}
  return(l)
}
# negative logLikelihood of a set of trees
nllik.st = function(pars, st){
  m = length(st$rec)
  l = vector(mode = 'numeric',length = m)
  w = vector(mode = 'numeric',length = m)
  for(i in 1:m){
    s = st$rec[[i]]
    w[i] = st$w[i]
    l[i] = nllik.tree(pars,tree=s)
  }
  #L = sum(l*w)/m
  w = w/sum(w)
  L = sum(l*w)
  return(L)
}
# relative likelihood
rel.llik <- function(S1,p0,p1){
  m = length(S1)
  f1 = vector(mode='numeric',length = m)
  f2 = vector(mode='numeric',length = m)
  d = vector(mode='numeric',length = m)
  for(i in 1:m){
    s = S1[[i]]
    f1[i] = nllik.tree(pars=p1,tree=s)
    f2[i] = nllik.tree(pars=p0,tree=s)
    d[i] = length(s$tree$wt)
    if(is.na(f1[i])) print(s)
  }
  Delta = -log(sum(f1/f2)/m)
```

```
  return(Delta)
}
# MLE for a set of trees
mle.st <- function(S,init_par = c(0.5,0.5,100)){
  po = subplex(par = init_par, fn = nllik.st, st=S,hessian = TRUE)
  return(po)
}
# Monte-Carlo sampling / simulation of a set of complete trees
sim.sct <- function(tree,pars,m=10,printHistD=F,parallel=FALSE){
  if(parallel){
    no_cores <- detectCores()- 1
    cl <- makeCluster(no_cores)
    registerDoParallel(cl)
    trees <- foreach(i = 1:m, combine = list) %dopar% {
      ct =  emphasis::sim.extinct(tree = tree,pars = pars) # complete tree
      lw = ct$logweight
      return(list(wt=ct$wt,E=ct$E,logweight=ct$logweight,lw=lw,n=ct$n))
    }
    lw = sapply(trees,function(list) list$lw)
    dim = sapply(trees,function(list) length(list$wt))
    stopCluster(cl)
    lw = lw - max(lw)
    w = exp(lw)
    Rec = trees #esta hay que sacarla
  }
  else{
    Rec = vector(mode = 'list',length = m)
    d = vector(mode='numeric',length = m)
    lw = vector(mode='numeric',length = m)
    rp = vector(mode='numeric',length = m)
    sp = vector(mode='numeric',length = m)
    for(j in 1:m){
      rec = sim.extinct(tree = tree,pars = pars)
      Rec[[j]] = rec
      d[j] = length(rec$wt)
      lw[j] = rec$logweight
      rp[j] = prod(rec$rprob)
      sp[j] = prod(rec$sprob)
    }
    lw = lw - max(lw)
    w = exp(lw)
  }
  return(list(rec = Rec, w=w,dim=dim))
}
# Pilot study
pilot.study <- function(tree,epsilon,m1=10,printprocess=FALSE,init_par=c(1.2,0.3,60),l1=20,parallel = TR
  # pilot study suggested by Chan et. al
  pars = init_par
  M = matrix(ncol = 3,nrow = l1)
  H = matrix(ncol = 3,nrow = l1)
  for(i in 1:l1){
    S = sim.sct(tree,pars,m=m1,printHistD = TRUE,parallel = parallel)
    mle =  mle.st(S = S)
```

```
    pars = mle$par
    H[i,] = try(diag(solve(mle$hessian))/m1)
    M[i,] = pars
    print(paste('Q:',mle$value,'pars:',pars[1],pars[2],pars[3]))
  }
  l = 10
  PM = M[1:(l1-10),]
  PH = H[1:(l1-10),]
  M = M[(l1-9):l1,]
  H = H[(l1-9):l1,]
  Q = vector(mode="numeric",length = (l1-10))
  MLE = list()
  for(i in 1:10){
    Delta = vector(mode="numeric",length = l)
    Me = matrix(ncol = 3,nrow = l)
    if(printprocess) print(paste('iteration',i))
    for(j in 1:l){
      S = sim.sct(tree,M[i,],m=m1,parallel = parallel)
      mle = mle.st(S = S)
      pars = mle$par
      Me[j,] = pars
      Delta[j] = rel.llik(S1 = S$rec,p0 = M[i,], p1 = pars)
    }
    MLE[[i]] = Me
    mD = mean(Delta)
    Q[i] = sum((Delta-mD)^2)
  }
  s2 = sum(Q)/((l-1)*(l1-10+1))
  s1 = sqrt(s2)
  m = m1*s1/epsilon
  m = floor(m) + 1
  return(list(m=m,p=M[10,],s1=s1,M=M,H=H,MLE=MLE,PM=PM,PH=PH))
}
#MCEM
mcem.tree <- function(tree,p,parallel=TRUE){
  m = p$m
  s1 = p$s1
  sig = 100*s1/m
  tol = 2*sig*sqrt(1/5)
  D = Inf
  k = 1
  print("initializing mcem")
  pars = p$p
  PARS = pars
  H = c(NULL,NULL,NULL)
  Me = p$M
  while(abs(D)>tol){
    S = sim.sct(tree,pars,m = m,parallel = parallel)
    M = mle.st(S = S)
    mle = M$par
    h1 = try(diag(solve(M$hessian))/m)
    if(is.numeric(h1)) H =  rbind(H,h1)
    D = rel.llik(S1 = S$rec,p0 = pars,p1 = mle)
```

```
    PARS = rbind(PARS,mle)
    pars = mle
    print(paste("iteration",k,"Q: ",M$value, " lambda: ", pars[1]," mu: ", pars[2], "K:", pars[3]))
    k = k+1
  }
  PARS = data.frame(it=1:(dim(Me)[1]+dim(PARS)[1]),lambda = c(Me[,1],PARS[,1]),mu=c(Me[,2],PARS[,2]),K=
  return(list(pars=pars,PARS=PARS,H=H))
}
############################

### Phylogenetic tree simulation
sim.tree <- function(ct=15, lambda0=0.8, mu0=0.1, K=40, model="dd", printEv=FALSE, seed=0){
  ## Set up
  if(seed>0){set.seed(seed)}
  key=0  # key to go out of the loop
  reboot2=0  # reboot in case that the tree is too small
  while(key==0){
    i = 1
    N = 2 # Starting number of species
    sigmas = NULL # vector with waiting times rates
    Tm = NULL # Waiting times
    E = NULL # vector with 0 if extinction and 1 if speciation
    n = NULL # vector with number of species at time t_i
    S = NULL # vector with species that went extinct/speciation # write this better
    sumt = 0 # time in simulation
    reboot = 0 # this is in case we want to check how many reboots the simulation had.
    newick = paste(sl[1],";",sep="")  # Newick tree
    identf = data.frame(Spec="aa",Time=0) # Labels of species
    L = data.frame(spec='aa', spec_time=0, ext_time=-1, parent = '00') # is this working?
    while (sumt < ct){
      if(model == "dd"){  # diversity-dependence model
        lambda = max(0,lambda0 - (lambda0-mu0)*N/K)
        mu = mu0
        lambda = rep(lambda,N)
        mu = rep(mu,N)
      }
      if(model == 'cr'){ # constant-rate model
        lambda = rep(lambda0,N)
        mu = rep(mu0,N)
      }
      s = sum(lambda)+sum(mu)
      sigmas = c(sigmas,s)
      if (s == 0){break}
      tm = rexp(1,s)  # waiting time of iteration i
      if(tm+sumt>ct){break}
      sumt = tm + sumt
      prob = c(lambda,mu)/s  # Probability of extinctions and speciations
      BD = sample(2*N,1,prob=prob)  # speciation/extinction & identification of the species.
      n[i] = N
      if(BD > N){   # Extinction
        E[i] = 0
        ## for newick output
        species = as.character(identf[BD-N,1])
```

```
      S[i] = species
      ind = regexpr(species,newick)[1] + 2
      atm = sumt-identf[which(identf[,1]==species),2]
      identf = identf[-(BD-N),]
      L[L$spec == species,]$ext_time = sumt #?
      newick = paste(substr(newick,1,ind),as.character(atm),substring(newick,ind+2),sep="")
      N = N-1
      if(printEv){print(paste("extinction in time",sumt, sep=" "))} # put it in a log file
    }else{  # Speciation
      E[i] = 1
      ## for newick output
      species = as.character(identf[BD,1])
      S[i] = species
      ind = regexpr(species,newick)[1]-1
      atm = sumt-identf[which(identf[,1]==species),2]
      newick = paste(substr(newick,1,ind),"(",substr(newick,ind+1,ind+4),",",sl[i+1],"):",as.characte
      identf = rbind(identf,data.frame(Spec=substr(sl[i+1],1,2),Time=sumt))
      identf[identf$Spec == species,2] = sumt
      L = rbind(L,data.frame(spec=substr(sl[i+1],1,2), spec_time=sumt, ext_time=-1, parent=species))
      N = N+1
      if(printEv){print(paste("speciation in time",sumt,sep=" "))}  # put it in a log file
    }
    if (N==0){ # In case all species got extinct: restart
      reboot = reboot + 1
      N = 2 # Number of species
      i = 1
      Tm = NULL
      sumt = 0
      E = NULL # vector with 0 if extinction and 1 if speciation
      n = NULL # vector with number of species at time t_i
      newick = paste(sl[1],";",sep="")  # Newick tree
      identf = data.frame(Spec="aa",Time=0)
      L = data.frame(spec='aa', spec_time=0, ext_time=-1, parent = '00')
    }else { # Otherwise, update values and go to next iteration
      Tm[i] = tm
      i<-i+1
    }
  }
  ####
  if(nchar(newick)<7){
    reboot2=reboot2+reboot+1
  }else{
    newick = compphyl(newi=newick,identf=identf,ct=ct) # set extant species to the present
    phy = read.tree(text=newick)
    dphy = drop.fossil(phy)
    if(Ntip(dphy)>2){ #check if the extant species tree is large enough
      key=1
    }else{
      reboot2=reboot2+reboot+1
    }
  }
}
Tm[i] = ct-sum(Tm)
```

```
  n[i] = n[i-1] + E[i-1] - (1-E[i-1])
  newick.extant = drop.fossil(phy)
  newick.extant.p = phylo2vectors(newick.extant) # is it validated?
  reboot2 = reboot2 + reboot
  return(list(tree=list(wt=Tm, E=E, n=n, S=S, br = cumsum(Tm)), phylo = phy, tree.extant = newick.extan
}

extinction.processes <- function(u,inits,mu0){
  nm = length(u)
  t.ext = vector(mode='numeric',length=nm)
  if(nm > 0){
    for(i in 1:nm){
      t.ext[i] = inits[i] - log(1-u[i])/mu0  #Inverse of the intensity function for constant extinction
    }
  }
  return(t.ext)
}
###  simulation of extincted new version
sim.extinct2 <- function(brts,pars,model='dd',seed=0){
  if(seed>0) set.seed(seed)
  wt = -diff(c(brts,0))
  ct = sum(wt)
  lambda0 = pars[1]
  mu0 = pars[2]
  K = pars[3]
  dim = length(wt)
  ms = NULL # missing speciations, for now we just add time. When we consider topology we do it with sp
  me = NULL # missing extinctions (in the uniform plane)
  bt = NULL
  to = NULL
  cbt = 0
  N = 2
  sprob = NULL # sampling probability of Missing|observed
  h = 1 # index to fill probabilities
  for(i in 1:dim){
    cwt = wt[i]
    cbt = sum(wt[0:(i-1)])
    key = 0
    gosttime = 0
    #last = 'nothing'
    while(key == 0){
      if(model == "dd"){  # diversity-dependence model
        lambda = max(1e-99, lambda0 - (lambda0-mu0)*N/K)
        mu = mu0
        s = N*lambda
      }else{print('Model not implemented yet, try dd')}
      t.spe = rexp(1,s)
      t.ext = extinction.processes(u=me,inits=ms,mu0=mu0)
      #sometimes parameters does not make sense. write a warning when that happens
      t_ext = ifelse(length(t.ext)>0,min(t.ext),Inf)-cbt  # if is not empty gives the waiting time for
      mint = min(t.spe,t_ext)
      if(mint < cwt){
        if(mint == t.spe){#speciation
```

```
          u = runif(1)
          if(u < pexp(ct-(cbt+t.spe),mu)){
            ms = c(ms,cbt+t.spe)
            me = c(me,u)
            bt = c(bt,cbt+t.spe)
            to = c(to,1)
            sprob[h] = sampprob(t = t.spe+gosttime, s = s, mu = mu, r = ct-(cbt-gosttime))
            h = h + 1
            N = N + 1
          }else{gosttime = t.spe + gosttime}
          cwt = cwt - t.spe
          cbt = cbt + t.spe
        }
        else{#extinction
          extinctone = which(t.ext == min(t.ext))
          tspe = ms[extinctone]
          text = t.ext[extinctone]
          bt = c(bt,text)
          to = c(to,0)
          sprob[h] = truncdist::dtrunc(text-tspe,'exp',a=0,b=ct-tspe,rate=mu)*(1-integrate(sampprob,lowe
          ms = ms[-extinctone]
          me = me[-extinctone]
          cwt = cwt - mint
          cbt = cbt + mint
          N = N-1
          h = h+1
          gosttime = 0
        }
      }
      else{
        key = 1
        sprob[h] = (1 - integrate(Vectorize(sampprob),lower = 0, upper = cwt+gosttime,s=s,mu=mu,r=ct-cbt
        h = h+1
      }
    }
    N = N+1
  }
  df = data.frame(bt = c(bt,15-brts),to = c(to,rep(2,length(wt))))
  df = df[order(df$bt),]
  n.tree = list(wt=c(diff(df$bt),ct-df$bt[length(df$bt)]),E=df$to[-length(df$to)])
  if(length(n.tree$E==1) != length(n.tree$E==0)) print('algo mal!!')
  n.tree$E[n.tree$E==2] = 1
  lrprob = -nllik.tree(pars,n.tree) #f
  lsprob = sum(log(sprob)) #g
  logweight = lrprob-lsprob
  n.tree$weight = exp(logweight)
  n.tree$logweight = logweight
  n.tree$f=lrprob
  n.tree$g=lsprob
  return(n.tree)
}
```

```
post.pro <-function(file,extrafile=NULL){
load(file)
#pars = DDD::dd_ML(brts = btdd, idparsopt = 1:3,soc=2,cond=0)
MLE = p$MLE
M = mcem$PARS

it = NULL
lambda = NULL
mu = NULL
K = NULL
for( i in 1:10){
  MM = MLE[[i]]
  it = c(it,rep(i+1,10))
  lambda = c(lambda,MM[,1])
  mu = c(mu,MM[,2])
  K = c(K,MM[,3])
};
col = rep('grey',length(lambda))
SDl = rep(NaN,length(lambda))
SDm = rep(NaN,length(mu))
SDk = rep(NaN,length(K))

it = c(it,1:dim(M)[1])
lambda = c(lambda,M[,2])
mu = c(mu,M[,3])
K = c(K,M[,4])
col = c(col,rep('blue',dim(M)[1]))

MCEMc = data.frame(it=it,lambda=lambda,mu=mu,K=K,col=col)

gamLambda = gam(lambda ~ s(it), data=MCEMc)
gamMu = gam(mu ~ s(it), data=MCEMc)
gamK = gam(K ~ s(it), data=MCEMc)

ex = dim(MCEMc)[1] - dim(mcem$H)[1] - 100
hessL = c(rep(NaN,ex),mcem$H[,1])
SDl = c(SDl,sqrt(hessL+gamLambda$sig2))

hessM = c(rep(NaN,ex),mcem$H[,2])
SDm = c(SDm,sqrt(hessM+gamMu$sig2))

hessK = c(rep(NaN,ex),mcem$H[,3])
SDk = c(SDk,sqrt(hessK+gamK$sig2))

MCEMc$SDl = SDl
MCEMc$SDm = SDm
MCEMc$SDk = SDk

MCEM = rbind(MCEMc,data.frame(it=(-9:0),lambda=p$PM[,1],mu=p$PM[,2],K=p$PM[,3],col=rep('blue',10),SDl=r

gl=ggplot(MCEM) + geom_point(aes(it,lambda),colour=MCEM$col) + geom_errorbar(aes(x=it, y=lambda, ymin =
gm=ggplot(MCEM) + geom_point(aes(it,mu),colour=MCEM$col) + geom_errorbar(aes(x=it, y=mu, ymin = mu-1.96
gk=ggplot(MCEM) + geom_point(aes(it,K),colour=MCEM$col) + geom_errorbar(aes(x=it, y=K, ymin = K-1.96*SD
```

```
gLLmcem = ggplot(data=d)+geom_line(aes(x=it,y=llik))+geom_hline(yintercept = pars$loglik)
gLLmcem2 = ggplot(df, aes(lambda, mu))+ geom_contour(aes(z = llik,color=..level..),bins=100) +geom_poin
if(!is.null(extrafile)){
  load(extrafile)
  gLLmcem2 = gLLmcem2 + geom_point(data=mcem$PARS,aes(x=lambda,y=mu,size=it),col='red')
  gLLmcem = gLLmcem + geom_line(data=d,aes(x=it,y=llik),col='red')
}
return(list(gl=gl,gm=gm,gk=gk, gLLmcem=gLLmcem, gLLmcem3d=gLLmcem2))
}
```