Importance sampling

We need

$$Q(\theta|\theta^*) = E_{MT}[l_{MT,OT}(\theta)|OT,\theta^*]$$

So MT (missing tree) needs to be sampled from (density)

$$f_{MT|OT}(T;\theta^*)$$

But instead we sample it from

$$g(T;\theta^*,OT)$$

To correct this we re-weigh the expectation by importance scaling:

$$\frac{f_{MT|OT}(T;\theta^*)}{g(T;\theta^*,OT)}$$

Where, in the case of $f_{MT|OT}(T;\theta^*) = -log(\prod_i P_{ms_i}) = -(\sum log(P_{ms_i}))$ and P_{ms_i} corresponds to the probability for the missing species number i. That is

$$P_{ms_i} = \left(\sum\lambda\right)_i e^{t_i(\sum\lambda)_i}\frac{1}{n_i}\frac{\mu e^{t_i\mu}}{1-e^{-\mu rT}}$$

Then, for the case of DD we have

$$P_{ms_i} = n_i(\lambda_0-\beta n_i)e^{t_i(n_i(\lambda_0-\beta n_i))}\frac{1}{n_i}\frac{\mu e^{t_i^{ext}\mu}}{1-e^{-\mu(ct-bt_i)}}$$

Moreover,

$$log(P_{ms_i}) = log(\lambda_0-\beta n_i) + t_i n_i(\lambda_0-\beta n_i) + log(\mu) + t_i^{ext}\mu - log(1-e^{-\mu(ct-bt_i)})$$

I calculate this with the following function

```
num_weigh <- function(rec_tree, pars_rec, ct){
  lambda0 = pars_rec[1]
  beta = pars_rec[2]
  mu = pars_rec[3]
  L = rec_tree$L
  wt = rec_tree$wt
  miss <- L[L[,3]!=(-1),]
  miss_spe <- miss$spec
  time <- rbind(data.frame(brtimes = L[,2], E=1, spec = L[,1]),data.frame(b
rtimes = miss[,3], E=0, spec = miss[,1]))
  time <- time[order(time$brtimes),]
  time$n <- cumsum(time$E)-cumsum(1-time$E)
  time$n <- c(0,time$n[1:length(time$n)-1])
  wait_time = c(diff(time$brtimes))
  time <- time[2:dim(time)[1],]
  time$wt <- wait_time
  missing = time[which(is.element(time$spec,miss_spe)),]
  if(dim(missing)[1]<1){print('THERE IS NOT MISSING SPECIES')}
  m_spec = unique(missing$spec)
  P = matrix(nrow = length(m_spec), ncol=1)
  for(i in 1:length(m_spec)){
    sub_mat = missing[missing$spec == m_spec[i],]
    if(m_spec[i] == 'aa') sub_mat = missing[missing$spec == m_spec[i],]
    lambda = lambda0-beta*sub_mat$n[1]
    t_ext = sub_mat$brtimes[2]-sub_mat$brtimes[1]
    P[i] = log(lambda) + sub_mat$wt[1]*sub_mat$n[1]*lambda +log(mu) + t_ext
*mu-log(1-exp(-mu*(ct-sub_mat$brtimes[1])))
    #P[i] = suml*exp(wt[i]*suml)*(1/ sub_mat$n[1])*(mu*exp(t_ext*mu)/(1-exp
(-mu*(ct-sub_mat$brtimes[1]))))
    #print(suml*exp(wt[i]*suml)*(1/ sub_mat$n[1])*(mu*exp(t_ext*mu)/(1-exp(
-mu*(ct-sub_mat$brtimes[1])))))
  }
  prob = -sum(log(P))
  return(prob)
}
```

Moreoever, $g(T;\theta^*,OT)$ is calculated on the reconstructed process.

Thus, in the `rec_tree` funcion we calculate the weight and we give it to `llik_st`.

Apendix

```
rec_tree <- function(wt, pars=c(0.8,0.0175,0.1), model='dd'){
  lambda0 = pars[1]
  mu0 = pars[3]
  K = (lambda0-mu0)/pars[2]
  n = 1:length(wt)
  i = 1
  E = rep(1,(length(wt)-1))
  fake = FALSE
  ct = sum(wt)
  prob = 0
  while(i < length(wt)){
    N = n[i]
    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)
    if(s==0){
      #print('s=0')
      break
    }
    if(fake){ # in case there was an speciation but not extinction previous
ly
      cwt = cwt - t_spe
      cbt = cbt + t_spe
    }
    else{
      cwt = wt[i]
      cbt = cumsum(wt)[i] - cwt
    }
    t_spe = rexp(1,s)
    if (t_spe < cwt){
      t_ext = rexp(1,mu0) # this is not as general as trees with trait-depe
ndance species yet,
      t_ext = cbt + t_spe + t_ext
      prob = prob + log(1-exp(-s*cwt))
      if (t_ext < ct){
        up = update_tree(wt=wt,t_spe = (cbt + t_spe), t_ext = t_ext, E = E,
 n = n)
        E = up$E
        n = up$n
        wt = up$wt
        fake = FALSE
        prob = prob + log(1-exp(-mu0*(ct-t_spe-cbt)))
      }else{
        prob = prob - mu0*(ct-t_spe-cbt)
        fake = TRUE
        i = i-1
      }
    }else{
      fake = FALSE
      prob = prob - s*cwt
    }
    i = i+1
  }
#  newick = p2phylo(wt,E,)
  L = create_L(wt,E)
  n_prob = num_weigh(rec_tree=list(wt=wt,E=E,n=n,L=L), pars_rec = pars, ct=
ct)
  weight = n_prob/prob
  print(weight)
  return(list(wt=wt,E=E,n=n,weight=weight,L=L))
}
```