This documents my exploration of the differences between `OUwie` and `ouch` when each is asked to fit OU models with a single α and σ operating on the entire tree. The motivation for this is that I noticed that the parameter estimates and likelihoods reached by `OUwie` and `ouch` are often quite different. I originally did this is 2014, but wanted to update it now to explore how the programs differ from one another now.

The procedure I will follow here is very straightforward: generate Yule trees of different sizes; specify two selective regimes arising out of the branching event at the root, so that there is a single monophyletic clade in each selective regime - this is the simplest possible multiple-optimum OU scenario; simulate phenotypic data with a known α and σ; fit an OU model to the phenotypic data and tree using both `OUwie` and `ouch` and record the parameter estimates and likelihoods.

Trees were generated by assuming that branching follows a Poisson process, so that branching events are exponentially distributed. To generate trees of different sizes, I set the rate parameter of the exponential distribution to a value of 0.3 and found random number generator seeds that would generate trees of size from 10 to 100 tips, with the only constraint being that the smaller of the two clades originating at the root had at least 1/3 of the total tip species. This guarantees that both selective regimes are reasonably well-represented, thereby improving both power and parameter estimates.

Phenotypic data was generated assuming $\alpha = 3$ and $\sigma^2 = 1$, with selective optima of $\theta_1 = -1$ and $\theta_2 = 1$. The root was always assumed to be in regime 1. For any given tree size, 20 different phenotypic datasets were generated and fit to the OU model. This allows me to calculate how bias/variance in parameter estimates differ between the two algorithms. All of the code, including random number generator seeds, needed to reproduce the results shown here can be found at the end of this document.

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs
= "cs")'
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs
= "cs")'
```

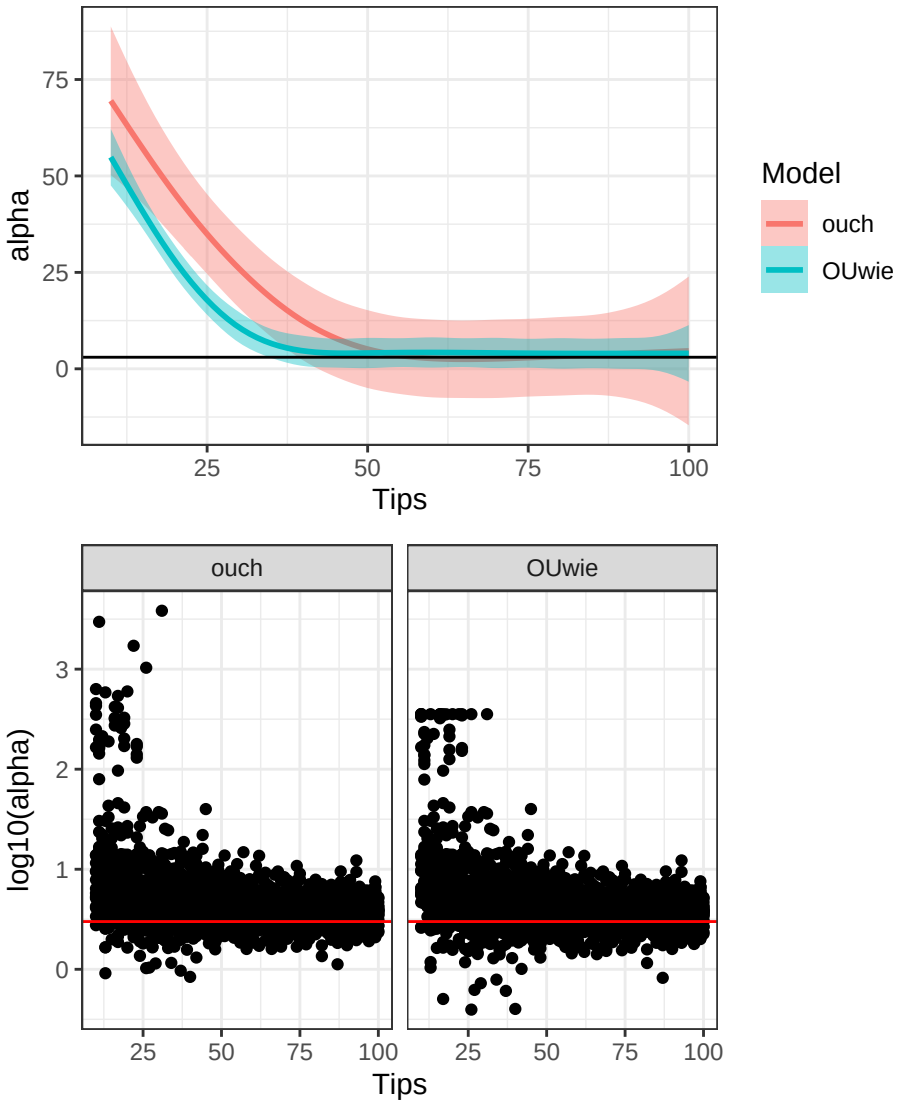

FIGURE 1. Comparing the α estimates of `ouch` and `OUwie` for trees of 10-100 tips. In the top panel, the lines and shaded region come from fitting a generalized additive model to the α estimates, with the size of the tree as the dependent variable. The line shows the expectation of the GAM, whereas the shaded regions capture the variability across the 20 stochastically generated phenotypic datasets. The black line shows the true α value. From this figure, it appears that the estimates are converging on the true value, but the error around the `ouch` estimates is higher than for `OUwie`. However, an examination of the estimates for each dataset (bottom panels) shows that variability is actually still fairly large. It is clear, however, that neither method is particularly good or bad at estimating α. If anything, `OUwie` is slightly better than `ouch`, which is a change from 2014, when I did this analysis.

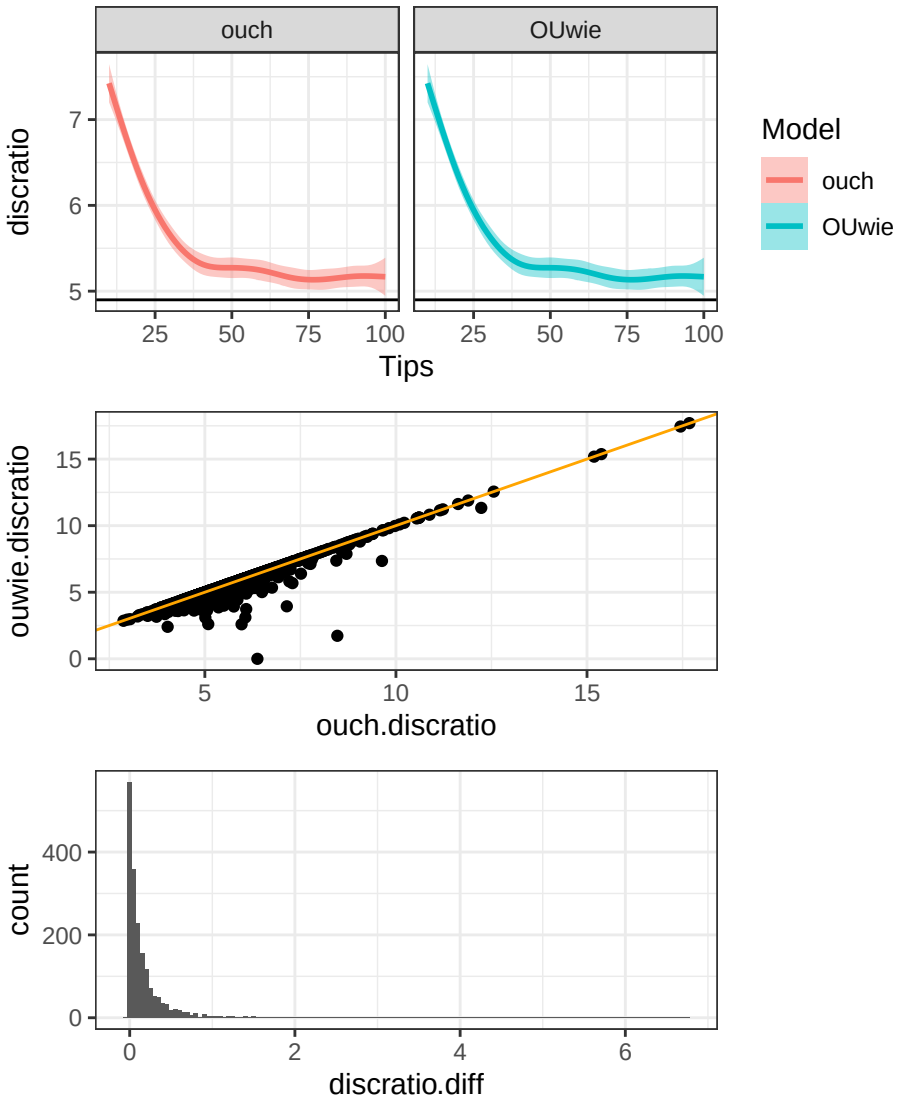

FIGURE 2. Estimates of the discriminability ratio, $\sqrt{2\alpha}\Delta\theta/\sigma$ (Cressler, Butler, and King 2015). This dimensionless combination of parameters may be better estimated than any individual parameter. This dimensionless parameter also captures the linked ability to estimate the selective optima, selection strength, and drift. You can see from all three panels that `ouch` and `OUwie` often estimate this parameter nearly identically: the top panel shows results from a generalized additive model fit to the estimates for each number of tips; the middle panel shows a scatterplot of the estimates for each method; the bottom panel shows a histogram of the difference between the `ouch` estimate and the `OUwie` estimate. Interestingly, the estimate returned by `ouch` is always larger than the estimate returned by `OUwie`.

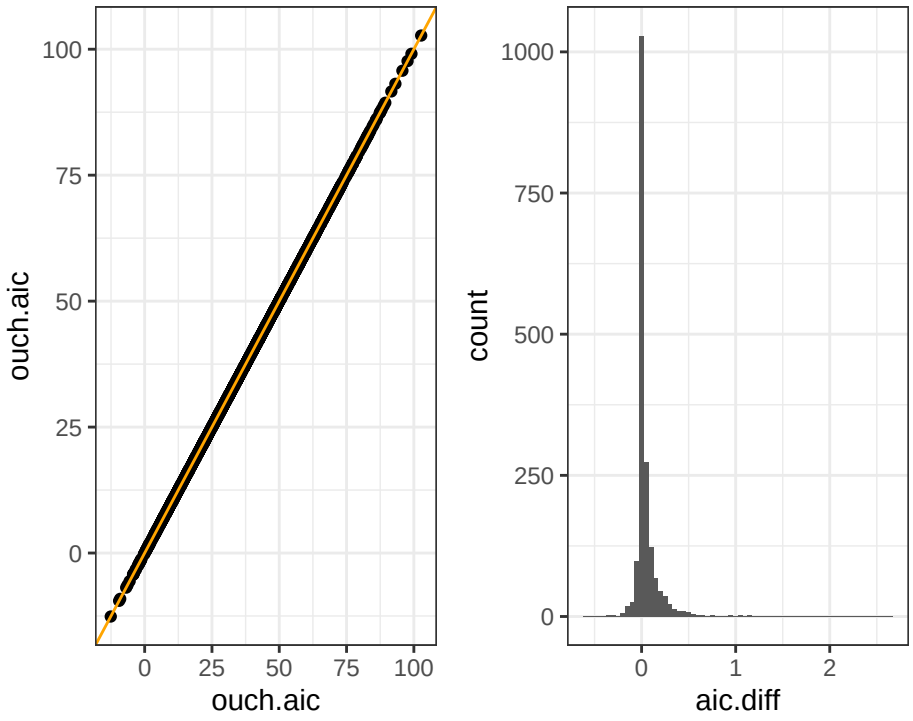

FIGURE 3. Estimated AICs for `ouch` and `OUwie` are essentially identical. The histogram shows the difference between the AIC of the parameters estimated by `ouch` and the AIC of the `OUwie` parameter estimates; positive values suggest that extttOUwie has found parameter estimates with a higher likelihood.

```
require(ouch)
## Creates an ouchtree object.
##
## 'waittime' must be specified by the user, and is the
## expected time to a branching event, assuming the branching
## follows a Poisson process (that is, that branching events
## are exponentially distributed)
##
## 'endtime' must also be specified and determines how long to
## let the branching process proceed.
##
## The smaller the value of waittime, or the larger the value
## of endtime, the larger the tree that will be produced. If
## only these two variables are set, the tree that is produced
## will be ultrametric; specifically, it will be a Yule tree,
## corresponding to a pure birth process.
##
## 'time' can also be specified, setting the time of the root
```

```
## node.

## A birth-death branching process can be simulated by setting
## the value of extrate, which determines the rate at which
## lineages go extinct. This rate is also assumed to be
## exponentially distributed.
##
gen.tree <- function(waittime, endtime,
                     time = 0, extrate = 0,
                     anc.node = NA, tree) {
    if (missing(tree))
        tree <- matrix(NA,0,3,
                       dimnames=list(NULL,c("node","ancestor","time")))
    curr.node <- nrow(tree)+1
    if (time < endtime) {
        tree <- rbind(tree,c(curr.node,anc.node,time))
        extinct.time <- if (extrate > 0) {
            time+rexp(2,rate=extrate)
        } else {
            c(Inf,Inf)
        }
        time <- pmin(time+rexp(2,rate=1/waittime),endtime)
        for (k in 1:2) {
            if (time[k] < extinct.time[k]) {
                tree <- Recall(time=time[k],
                               waittime=waittime,endtime=endtime,extrate=ext
            } else {
                tree <- rbind(tree,c(nrow(tree)+1,curr.node,extinct.time[k]]
            }
        }
    } else {
        tree <- rbind(tree,c(curr.node,anc.node,time))
    }
    return(as.data.frame(tree))
}

## Generate phenotypic data according to an Ornstein-Uhlenbeck
## process, given an ouchtree object specified by tree, regimes
## specified by regimes, and OU parameters theta, alpha, and sigma.
## This is done by recursion, which gen.phenotypic.values acting as
## the "parent" function that calls set.phenotypic.values, the
## function that is recursed over.
gen.phenotypic.values <- function(tree, regimes, theta, alpha, sigma, remove
  if (class(tree) == 'data.frame')
```

```
    tree <- with(tree, ouchtree(node,ancestor,time))
  if (tree@nnodes != length(regimes))
    stop('Length of regime specification != number of nodes in tree; each no
  if (length(theta) != length(levels(regimes)))
    stop('Each regime must have a unique theta value specified as the select

  root.reg <- as.numeric(regimes[1])
  root.pheno <- calc.root.pheno(theta[root.reg], alpha, sigma)
  pheno <<- vector(mode='numeric', length=tree@nnodes)
  set.phenotypic.values(curr.node=1, anc.node=0, anc.pheno=root.pheno, tree=

  ## if T, set the phenotypic values of internal nodes to NA
  if (remove.anc==T)
    pheno[as.numeric(setdiff(tree@nodes, tree@term))] <<- NA

  pheno
}
## Calculate the phenotypic value of the ancestral node as being drawn
## from the stationary distribution of the OU process.
calc.root.pheno <- function(theta, alpha, sigma) {
  ## mean and variance of stationary distribution
  mean <- theta
  var <- sigma^2/(2*alpha)
  rnorm(1, mean=mean, sd=sqrt(var))
}
## This function recursively sets the phenotypic values of the nodes
## in an ouchtree object specified by tree.  The values are, by
## default, set for a vector called 'pheno'
set.phenotypic.values <- function(curr.node, anc.node, anc.pheno, tree, regi
  ## set the phenotype of the current node
  if (curr.node == 1) {
    pheno[curr.node] <<- anc.pheno

    ## get the descendents of the root node and set their phenotypes
    desc <- which(tree@ancestors==curr.node)
    Recall(curr.node=desc[1], anc.node=curr.node, anc.pheno=anc.pheno, tree=
    Recall(curr.node=desc[2], anc.node=curr.node, anc.pheno=anc.pheno, tree=
  }
  else {
    ## generate a new phenotypic value
    x0 <- anc.pheno
    t <- as.numeric(tree@times[curr.node])-as.numeric(tree@times[anc.node])
    th <- theta[as.numeric(levels(regimes)[regimes[curr.node]])]
    a <- alpha
```

```
    p <- x0*exp(-a*t) + th*(1-exp(-a*t)) +
      sigma*sqrt((1-exp(-2*a*t))/(2*a))*rnorm(1, 0, 1)

    ## set the value of the phenotype
    pheno[curr.node] <<- p

    ## if the curr.node is not a tip, get its ancestors and set their
    ## phenotypes
    if (!(curr.node %in% tree@term)) {
      desc <- which(tree@ancestors==curr.node)

      ## set the phenotypes for all descendents
      for (n in 1:length(desc))
        Recall(curr.node=desc[n], anc.node=curr.node, anc.pheno=p, tree=tree
               regimes=regimes, theta=theta, alpha=alpha, sigma=sigma)
    }
  }
}

run <- FALSE;
if (run) {

    ## RNG seeds used to generate the trees
    seeds <- c(35250,48305,512,9741,10942,82785,3690754,52887,2694830,31665,

    ## RNG seeds for generating phenotypic datasets
    pseeds <- c(429561,582711,910097,733148,536289,732902,397913,193437,466

    results <- array(NA, dim=c(length(seeds)*length(pseeds),13))
    row <- 1
    for (i in 1:length(seeds)) {
        print(i)
        seed <- seeds[i]
        set.seed(seed)
        tree <- gen.tree(waittime=0.3, endtime=1)
        labels <- rep('', nrow(tree))
        labels[which(tree$time==1)] <- paste0('Sp',1:sum(tree$time==1))
        tree$labels <- labels
        ou.tree <- ouchtree(nodes=tree$node, ancestors=tree$ancestor, times=
        ou.reg <- paint(ou.tree, subtree=c('1'='1','2'='2'), branch=c('1'='1

        source("convert.R") ## for converting ouchtree to apetree
        ape.tree <- convert(ot=ou.tree)
        ## Label the internal nodes with their regime
```

```
        ## figure out which clade is in regime 2
        ## root is ntips+1, find its daughters
        daughters <- which(ape.tree$edge[,1]==ou.tree@nterm+1)
        ## which daughter corresponds to node 2 in the ouchtree?
        node2 <- which(round(ape.tree$edge.length[daughters],4)==round(ou.t
        ## what is the nodelabel for this daughter
        start <- ape.tree$edge[daughters[node2],2]
        ## what is the nodelabel for the other daughter?
        finish <- ape.tree$edge[daughters[-node2],2]
        ## create node.labels
        nodes <- (ou.tree@nterm+1):(ape.tree$Nnode+ou.tree@nterm)
        regs <- rep('1', length(nodes))
        reg2 <- start:(finish-1)
        regs[which(nodes%in%reg2)] <- '2'
        ape.tree$node.label <- regs

        ## Plot trees to ensure that the trees and paintings are identical
        ## plot(ou.tree, regimes=ou.reg)
        ## par(ask=T)
        ## plot(ape.tree)
        ## nodelabels(pch=21, bg=ape.tree$node.label)

        for (j in 1:length(pseeds)) {
            set.seed(pseeds[j])
            ## simulate phenotypic data
            x <- gen.phenotypic.values(ou.tree, ou.reg, theta=c(-1,1), alpha
            ## create data frames of phenotypic data
            ou.dat <- data.frame(val=x)
            ape.dat <- as(ou.tree, 'data.frame')[,c(4,1,2,3)]
            ape.dat$val <- x
            ape.dat$reg <- ou.reg
            ape.dat <- ape.dat[!is.na(ape.dat$val),c('labels','reg','val')]

            ## fit OU model using ouch and OUwie and record
            ou.OU <- hansen(ou.dat, ou.tree, ou.reg, sqrt.alpha=1, sigma=1)
            ape.OUM <- OUwie(ape.tree, ape.dat, model='OUM')
            results[row,] <-
                c(ou.tree@nterm,
                  as.numeric(sapply(c('1','2'),function(x)sum(ou.reg[ou.tree
                  as.numeric(summary(ou.OU)$aic),
                  as.numeric(summary(ou.OU)$alpha),
                  as.numeric(summary(ou.OU)$sigma.sq),
                  as.numeric(summary(ou.OU)$optima$val),
                  ape.OUM$AIC,
```

```
                    ape.OUM$solution[1,1],
                    ape.OUM$solution[2,2],
                    ape.OUM$theta[1:2,1])
                row <- row+1
            }
        }
        colnames(results) <- c('ntips','nreg1','nreg2','ouch.aic','ouch.alpha',
        results <- as.data.frame(results)
        save(results, file='Comparing_OUwie_and_ouch_results.rda')
}
```