Modelling Evolutionary Trees
CSC8622

Pedro Pinto da Silva   Alexander Kell

November 5, 2016

Part 1

Question (i)

Below we present our functions for generating a Yule tree. The algorithm for building the structure of the tree is straightforward:

1. Initialize a vector of size $2n - 2$ with zeros.
2. For each value j in 1 to $n - 1$:
   (a) Find the non-zero elements of the vector whose index (represent extant species).
   Special case for $j = 1$.
   (b) Sample one value from the set of candidates (parent).
   Special case for $j = 1$ where parent is the root with value $2 \times n - 1$.
   (c) Sample up to two positions in the vector with value zero (childs).
   (d) Set the value of childs in the vector as the index of the parent.

In the end of the for loop we have a single numeric vector v containing the structure of tree. Species or edge childs are given by the indexes of v whereas the parent of a species stored at i is obtained by simply accessing $v[i]$. Children of species i are obtained by searching the vector for the element whose value is equal to i. If v does not contain such value then i is an extant species. The root is represented by value $2 \times n - 1$. Although this was a neat and resource-wise cheap way to store and represent the tree, we are also required to store the length of each edge. Therefore we augmented this basic representation of the tree to a matrix that also includes the birth, termination and length of each species. Even though that for a given edge the value of the child is given by the index of the row, we added a column "Child" to the generated matrix in order to increase readibility (as we are not worried about the space tradeoff).

However, we did not label the nodes according to the example given, i.e. the parent node labelled as $n + 1$, the inner nodes between $n + 2$ and $2xn - 1$ and the leaf nodes between 1 and n. To that extent, we perform on a simple reordering of the labels which does not affect the structure of the tree. Finally we wrapped our method in a function that returns a data frame with further information, namely whether the species (or child) is extant or not.

```
buildTree = function(n=10, lambda=0.5) {
  nspecies = (2*n - 2)
  cols = c("parent", "child", "birth", "termination", "length")
  tree = matrix(NA, nrow = nspecies, ncol = length(cols))
  colnames(tree) = cols
  tree[,c("parent", "child")] = 0

  t = 0
  for(k in 1:(n-1)) {
    if(k == 1) {
      parent = 2*n - 1
    } else {
      # A candidate edge is an edge of which the child is an extant
      # species. In this case we look for children which do not
      # have an entry in the vector of parents.
      candidates = which( ! (tree[, "child"] %in% tree[, "parent"]))
      # Length of candidates is always > 1 otherwise we would
      #  have to be careful with the behavior of sample
      #  (undesired behavior for length == 1)
      parent = sample(candidates, 1)
      t = t + rexp(1, rate = k*lambda)
    }

    childs = sample(which(tree[,"parent"]==0), 2)
    tree[childs, "child"] = childs
    tree[childs, "parent"] = parent
    tree[childs, "birth"] = t
    if(k > 1) {
      tree[parent, "termination"] = t
    }
  }
  t = t + rexp(1, rate = n*lambda)
  tree[is.na(tree[, "termination"]), "termination"] = t

  tree[, "length"] = tree[, "termination"] - tree[, "birth"]
  return(tree)
}

isExtant = function(tree, index=1:nrow(tree)) {
  ! (tree[index, "child"] %in% tree[, "parent"])
}

loadSpecies = function(path="../aux/species.txt") {
  species = read.table(path, header=FALSE, sep = "+", stringsAsFactors = FALSE)$V1
  species[-which(species=="unavailable")]
}

# Yet Another Yule (YAY)
yay = function(n=10, lambda=0.5) {
  tree = buildTree(n, lambda)
  species = loadSpecies()
  if(length(species) > 0) {
    nomes = species[sample(1:length(species), nrow(tree)+1)]
  } else {
    nomes = paste("poney", 1:(nrow(tree)+1), sep="")
  }

  yule = data.frame(Parent      = tree[, "parent"],
                    ParentName  = nomes[tree[, "parent"]],
                    Child       = tree[, "child"],
                    ChildName   = nomes[tree[, "child"]],
                    isExtant    = isExtant(tree),
                    Birth       = tree[, "birth"],
                    Termination = tree[, "termination"],
                    Length      = tree[, "length"])
  yule[yule$Parent == 2*n-1, ]$ParentName = nomes[2*n-1]
  return(yule)
}

yule = yay()
head(yule, 1)

##   Parent  ParentName Child              ChildName isExtant     Birth
## 1     18 Bison bison     1 Limnocorax flavirostra    FALSE 0.3013138
##   Termination   Length
## 1    1.669321 1.368007

yule[, -c(2,4)]

##    Parent Child isExtant     Birth Termination    Length
## 1      18     1    FALSE 0.3013138   1.6693207 1.3680069
## 2      11     2     TRUE 1.3888936   2.7824920 1.3935984
## 3      16     3     TRUE 2.4640271   2.7824920 0.3184649
## 4       6     4     TRUE 1.8110572   2.7824920 0.9714348
## 5      11     5     TRUE 1.3888936   2.7824920 1.3935984
## 6       1     6    FALSE 1.6693207   1.8110572 0.1417365
## 7      19     7    FALSE 0.0000000   0.2899637 0.2899637
## 8      18     8    FALSE 0.3013138   2.4844599 2.1831461
## 9       7     9     TRUE 0.2899637   2.7824920 2.4925283
## 10      8    10     TRUE 2.4844599   2.7824920 0.2980321
## 11     17    11    FALSE 0.3463737   1.3888936 1.0425198
## 12     16    12     TRUE 2.4640271   2.7824920 0.3184649
## 13      8    13     TRUE 2.4844599   2.7824920 0.2980321
## 14     17    14     TRUE 0.3463737   2.7824920 2.4361182
## 15      6    15     TRUE 1.8110572   2.7824920 0.9714348
## 16      1    16    FALSE 1.6693207   2.4640271 0.7947064
## 17     19    17    FALSE 0.0000000   0.3463737 0.3463737
## 18      7    18    FALSE 0.2899637   0.3013138 0.0113501
```

Question (ii)

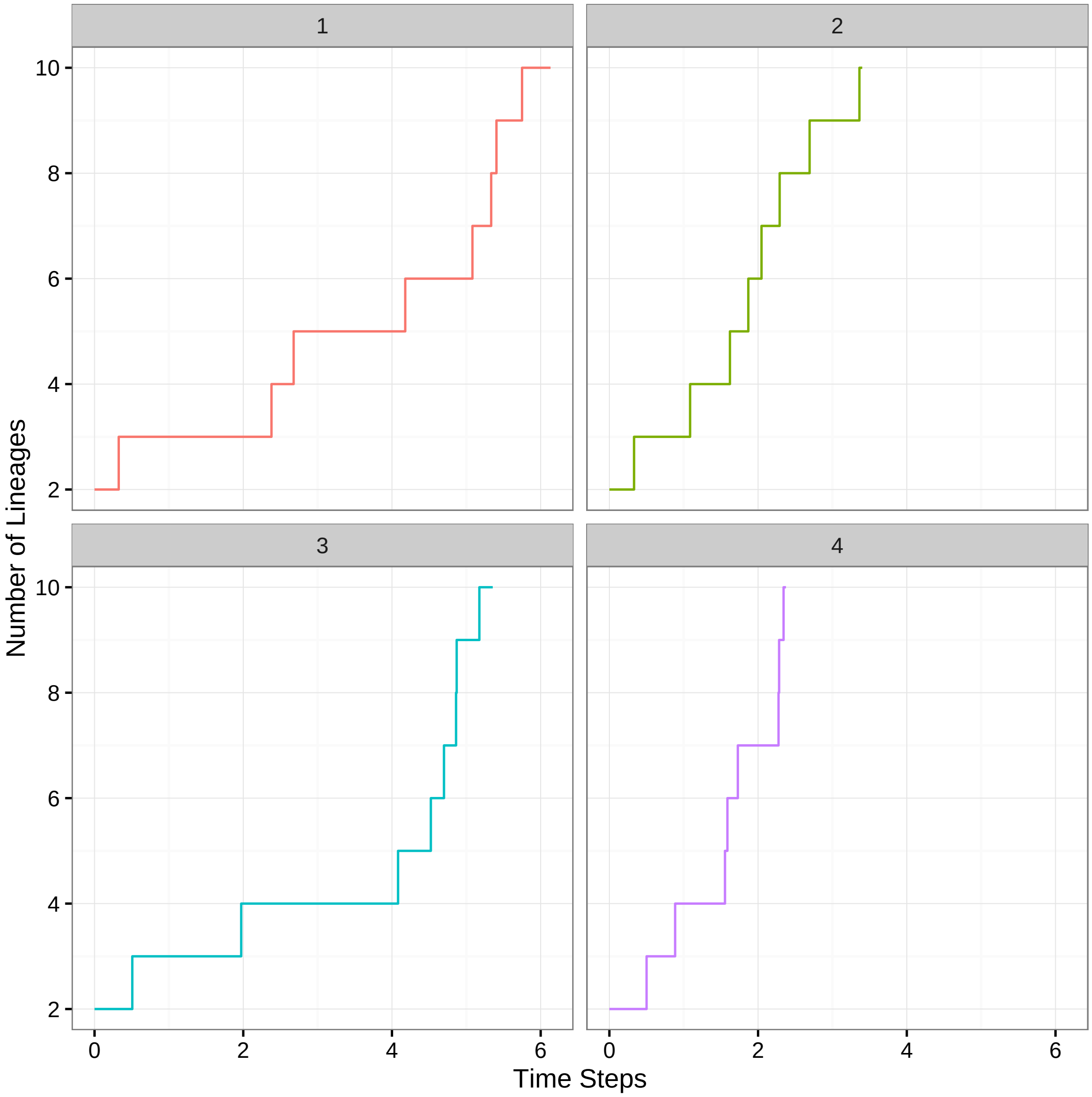
Below is the function for computing the time-step changes in the number of extant lineages:

```
yuleSteps = function(yule) {
  tstep = unique(sort(yule$Birth))
  tstep = c(tstep, max(yule$Termination))
  return(data.frame(tstep=tstep, nlineages=c(2:length(tstep), length(tstep))))
}

yuleSteps(yule)

##        tstep nlineages
## 1  0.0000000         2
## 2  0.2899637         3
## 3  0.3013138         4
## 4  0.3463737         5
## 5  1.3888936         6
## 6  1.6693207         7
## 7  1.8110572         8
## 8  2.4640271         9
## 9  2.4844599        10
## 10 2.7824920        10
```

Question (iii)

Using the function previously defined we created a time series step-plot for four different yule trees. It's worth noting that we coded this task so that more plots could be generated using the same code (only required to change the value of the variable *nPlots*).

```
n = 10
lambda=0.5
nPlots = 4

yays = lapply(1:nPlots, function(i) yuleSteps(yay(n, lambda)))
yays = rbind.fill(yays)
yays$group = ((as.numeric(rownames(yays)) - 1) %/% n) + 1

ggplot(yays, aes(x = tstep, y = nlineages))   +
  geom_step(aes(colour=factor(group))) +
  facet_wrap(~ group, ncol=nPlots %/% 2) +
  ylab("Number of Lineages") +
  xlab("Time Steps") +
  theme_bw() +
  scale_colour_discrete(guide = FALSE)
```



Part 2

Question (i)

At this stage, we introduced a new function to relabel the edges as required and return a phylo object instead. The relabelling, aparently tricky at first, consisted in a simple re-order of parent and child labels based on whether the edge represented an extant or extinct species.

We created a new function instead of changing our previously defined *yay* function, because each returns a different representation of the yule tree, even though the underlying structure is the same. Consider that we might be interested later in creating our own class for representing phylogentic trees, called *yay*. Then, separating concerns now across different functions without altering behavior would prove beneficial later.

```
# Yet Another Phylo
yaPhylo = function(n=10, lambda=0.5) {
  yule = yay(n, lambda)

  # Relabelling the nodes
  yule[yule$isExtant==TRUE, ]$Child = 1:n
  yule[yule$isExtant==FALSE, ]$Child = (n+2):(2*n-1)
  yule$Parent = yule$Child[yule$Parent]
  yule[is.na(yule$Parent), ]$Parent = n + 1

  phylo = list(edge = matrix(c(yule$Parent, yule$Child), ncol = 2),
               edge.length = yule$Length,
               tip.label = paste("t", 1:10, sep=""),
               Nnode = n - 1)
  class(phylo) = "phylo"
  return(phylo)
}
```

Question (ii)

The *length* function for the phylo class consists in summing the lengths of all phylo edges:

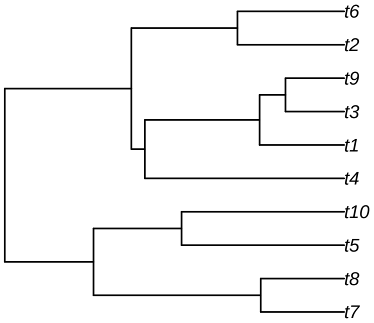
```
length.phylo = function(phylo) {
  sum(phylo$edge.length)
}
```

Question (iii)

Using the plot method for the phylo class we created following four tree plots:

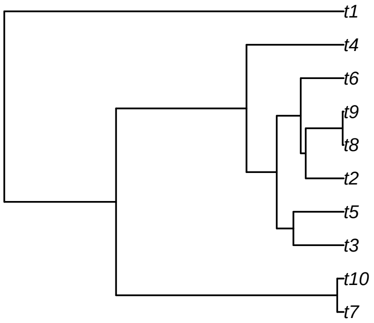
```
n = 10
lambda =  0.5
par(mfrow=c(2,2))

for (i in 1:4) {
  phylo = yaPhylo(n, lambda)
  plot(phylo)
  title(main=paste("Tree", i),
        sub=paste("Phylogenetic Diversity:", round(length(phylo),2)),
        line = 1)
}
```

Tree 1



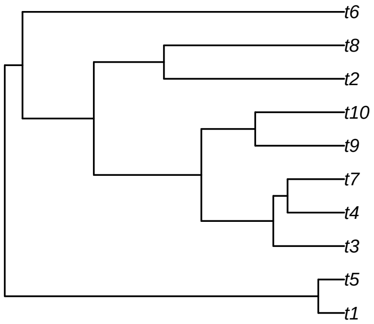
Phylogenetic Diversity: 16.31

Tree 2



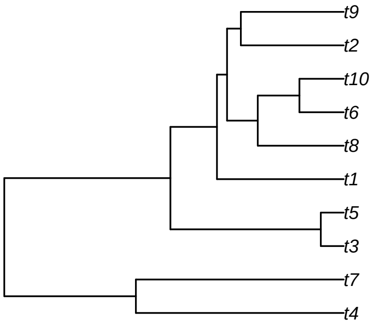
Phylogenetic Diversity: 25.18

Tree 3



Phylogenetic Diversity: 9.36

Tree 4



Phylogenetic Diversity: 9.07

Part 3