Lab 7

Peter Antonaros

#YARF

For the next labs, I want you to make some use of my package. Make sure you have a JDK installed first

https://www.oracle.com/java/technologies/downloads/

Then try to install rJava

```
#install.packages("rJava")
options(java.parameters = "-Xmx8000m")
library(rJava)
.jinit()
```

If you have error, messages, try to google them. Everyone has trouble with rJava!

If you made it past that, please try to run the following:

```
if (!pacman::p_isinstalled(YARF)){
  pacman::p_install_gh("kapelner/YARF/YARFJARs", ref = "dev")
  pacman::p_install_gh("kapelner/YARF/YARF", ref = "dev", force = TRUE)
}
pacman::p_load(YARF)
```

Please try to fix the error messages (if they exist) as best as you can. I can help on slack.

#Rcpp

We will get some experience with speeding up R code using C++ via the `Rcpp` package.

First, clear the workspace and load the `Rcpp` package.

```
pacman::p_load("Rcpp")
```

Create a variable `n` to be 10 and a vaiable `Nvec` to be 100 initially. Create a random vector via `rnorm` `Nvec` times and load it into a `Nvec` x `n` dimensional matrix.

```
n=10
Nvec = 100
X = matrix(rnorm(n * Nvec), nrow=Nvec)
```

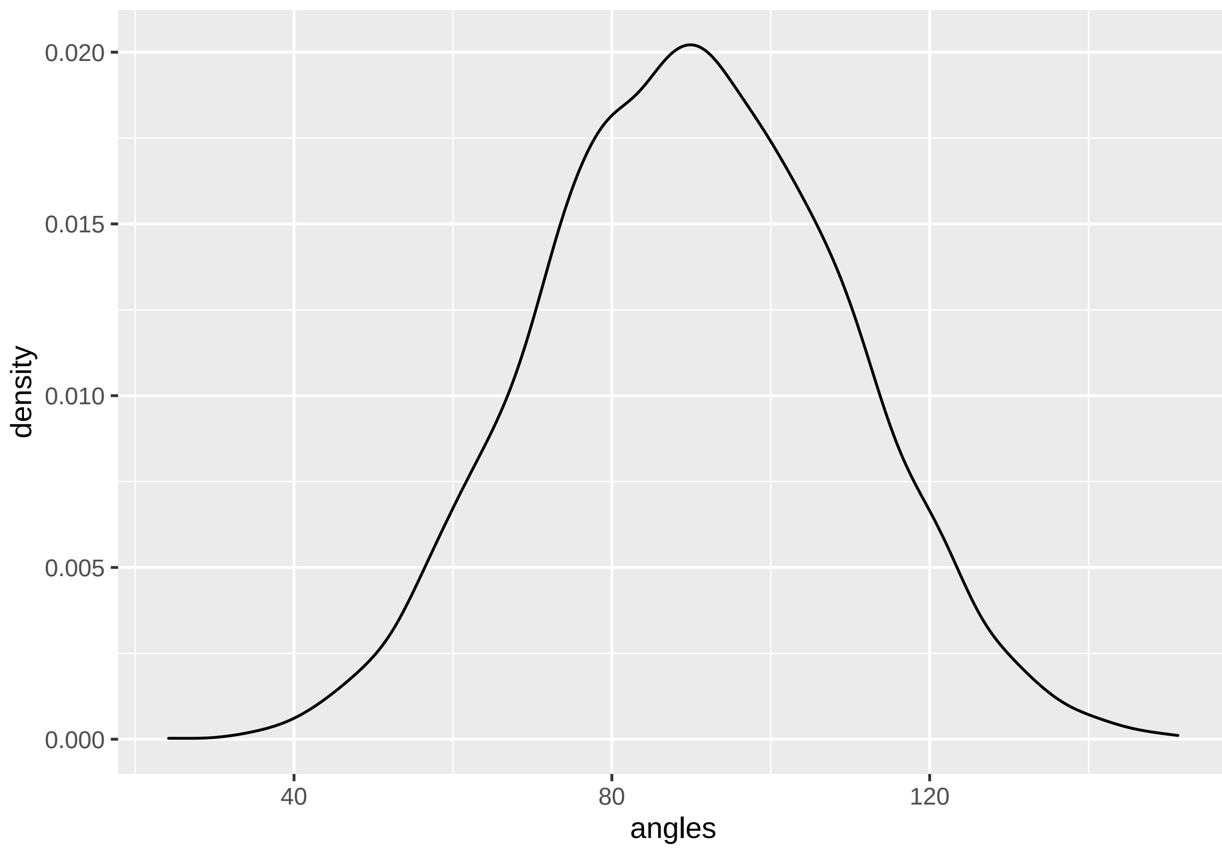
Write a function `all_angles` that measures the angle between each of the pairs of vectors. You should measure the vector on a scale of 0 to 180 degrees with negative angles coerced to be positive.

```r
all_angles = function(X){

  n = nrow(X)
  D = matrix(NA, nrow=n, ncol=n)


  for(i in 1:(n-1)){

    for(j in (i+1):n){

      x_i = X[i,]
      x_j = X[j,]

      D[i,j] = acos(sum(x_i*x_j)/sqrt(sum(x_i^2)*sum(x_j^2)))*(180/pi)
    }
  }
  D
}
```

Plot the density of these angles.

```r
D = all_angles(X)
pacman::p_load(ggplot2)
ggplot(data.frame(angles = c(D)))+
  geom_density(aes(x=angles))
```

```
## Warning: Removed 5050 rows containing non-finite values (stat_density).
```



Write an Rcpp function `all_angles_cpp` that does the same thing. Use an IDE if you want, but write it below in-line.

```
cppFunction('

  NumericMatrix all_angles_cpp(NumericMatrix X){

    int n = X.nrow();
    int p = X.ncol();
    NumericMatrix D(n,n);
    std::fill(D.begin(), D.end(), NA_REAL);

    for(int i =0; i<(n-1); i++){

      for(int j=i+1; j<n; j++){

        double dot_product = 0;
        double length_x_i_sq = 0;
        double length_x_j_sq = 0;

        for(int k=0; k<p; k++){

          dot_product += X(i,k) * X(j,k);
          length_x_i_sq += pow(X(i,k),2);
          length_x_j_sq += pow(X(j,k),2);
        }
```

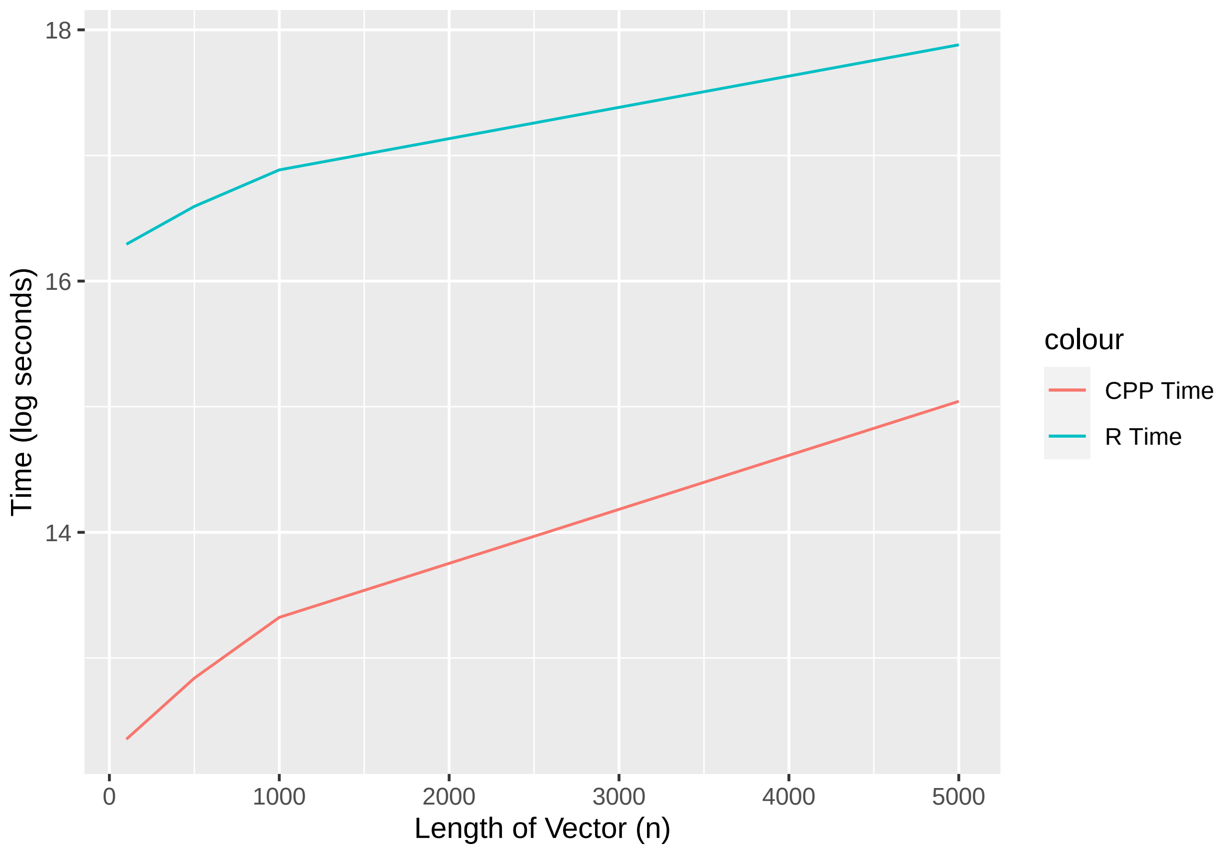
```
      D(i,j) = acos(dot_product/sqrt(length_x_i_sq * length_x_j_sq))*(180.0/M_PI);
    }
  }

  return D;
}
')
```

Test the time difference between these functions for `n = 1000` and `Nvec = 100, 500, 1000, 5000` using the package `microbenchmark`. Store the results in a matrix with rows representing `Nvec` and two columns for base R and Rcpp.

```r
#install.packages("microbenchmark")
library(microbenchmark)
Nvecs = c(100, 500, 1000, 5000)

results_for_time = data.frame(
  Nvec = Nvecs,
  time_for_base_R = numeric(length = length(Nvecs)),
  time_for_cpp = numeric(length = length(Nvecs))
)

for (i in 1 : length(Nvecs)){
  X = matrix(rnorm(n * Nvecs[i]), nrow = Nvec)

  results_for_time$time_for_base_R[i] = mean(microbenchmark(r_angles = all_angles(X), unit = "s")$time)

  results_for_time$time_for_cpp[i] = mean(microbenchmark(cpp_angles = all_angles_cpp(X), unit = "s")$ti

}
```

Plot the divergence of performance (in log seconds) over n using a line geometry. Use two different colors for the R and CPP functions. Make sure there's a color legend on your plot. We will see later how to create "long" matrices that make such plots easier.

```r
ggplot(results_for_time) +
  geom_line(aes(x = Nvec, y = log(time_for_base_R),col = "R Time")) +
  geom_line(aes(x = Nvec, y = log(time_for_cpp), col = "CPP Time")) +
  xlab("Length of Vector (n)")+
  ylab("Time (log seconds)")
```

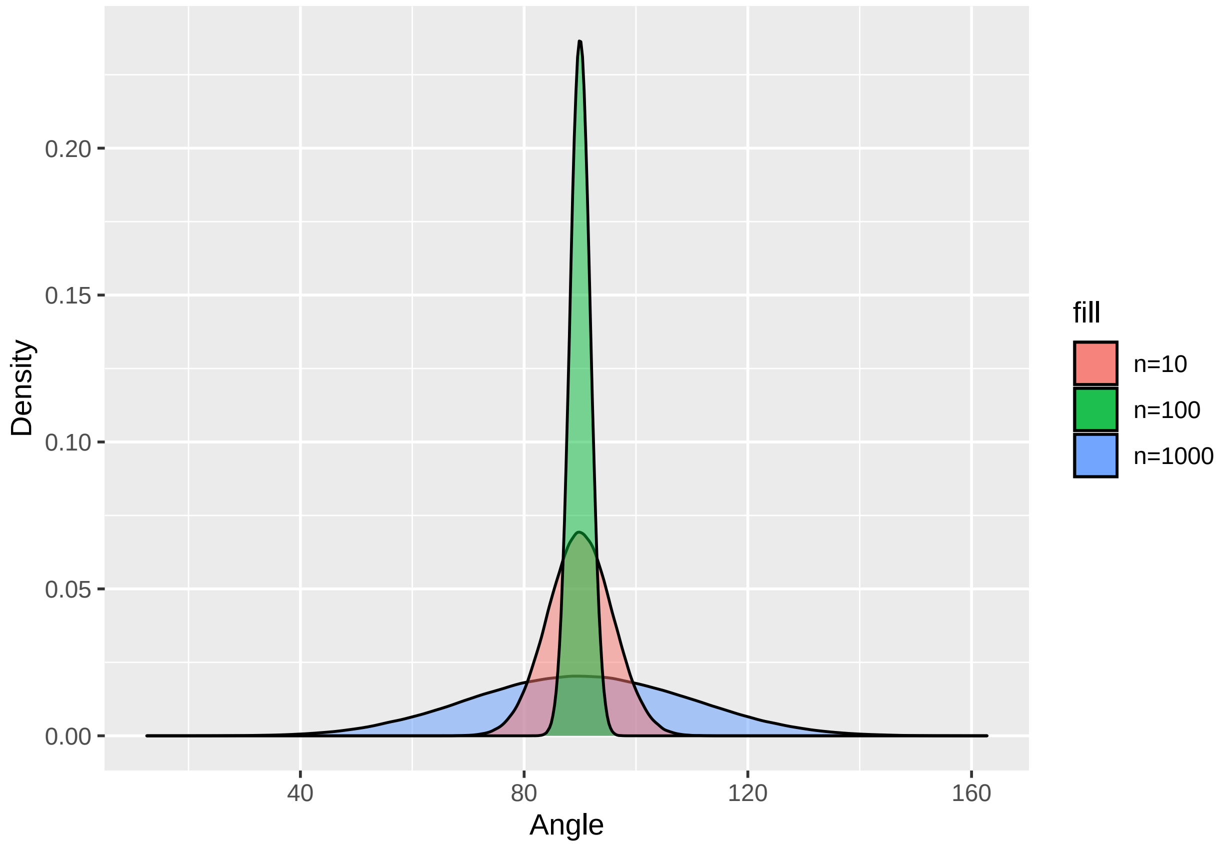


Let `Nvec = 10000` and vary `n` to be 10, 100, 1000. Plot the density of angles for all three values of `n` on one plot using color to signify `n`. Make sure you have a color legend. This is not easy.

```
#I cannot run it with Nvec = 10000, rStudio uses over 16gb of ram :( Need to download more ram XD
Nvec = 1000
n = c(10, 100, 1000)
angleDensities = vector("list", 3)

for(i in 1:length(n)){

  X = matrix(rnorm(n[i] * Nvec), nrow = Nvec)

  angleDensities[[i]] = all_angles_cpp(X)
}


ggplot()+
  geom_density(aes(x=angleDensities[[1]], fill = 'red'), alpha = 0.5)+
  geom_density(aes(x=angleDensities[[2]], fill = 'blue'), alpha = 0.5)+
  geom_density(aes(x=angleDensities[[3]], fill = 'green'), alpha = 0.5)+
  scale_fill_discrete(labels = c("n=10", "n=100", "n=1000"))+
  xlab("Angle")+
  ylab("Density")
```

```
## Warning: Removed 500500 rows containing non-finite values (stat_density).
## Removed 500500 rows containing non-finite values (stat_density).
```

```
## Removed 500500 rows containing non-finite values (stat_density).
```



Write an R function `nth_fibonnaci` that finds the nth Fibonacci number via recursion but allows you to specify the starting number. For instance, if the sequence started at 1, you get the familiar 1, 1, 2, 3, 5, etc. But if it started at 0.01, you would get 0.01, 0.01, 0.02, 0.03, 0.05, etc.

```
nth_fibonnaci = function(fibLimit){

  if(fibLimit <= 1){
    return(fibLimit)
  }else{

    return(nth_fibonnaci(fibLimit-1) + nth_fibonnaci(fibLimit-2))

  }

}
```

Write an Rcpp function `n4th_fibonnaci_cpp` that does the same thing. Use an IDE if ou want, but write it below in-line.

```
cppFunction('
   int nth_fibonnaci_cpp(int fibLimit){
    if(fibLimit <=1){
      return(fibLimit);
```

```
    }else{

      return (nth_fibonnaci_cpp(fibLimit-1) + nth_fibonnaci_cpp(fibLimit-2));
    }
   }

')
```

Time the difference in these functions for n = 100, 200, ..., 1500 while starting the sequence at the smallest possible floating point value in R. Store the results in a matrix.

```
n = c(1,5,10,15,20,25) #n values > 25 both functions start to struggle

fib_timing = data.frame(
  n = n,
  fib_R = numeric(length = length(n)),
  fib_cpp = numeric(length = length(n))
)

for (i in 1 : length(n)){


  fib_timing$fib_R[i] = mean(microbenchmark(r_fib = nth_fibonnaci(n[i]), unit = "s")$time)

  fib_timing$fib_cpp[i] = mean(microbenchmark(cpp_fib = nth_fibonnaci_cpp(n[i]), unit = "s")$time)

}

fib_timing
```

```
##    n       fib_R   fib_cpp
## 1  1    35547.73  14042.87
## 2  5     5191.19   1920.90
## 3 10    61705.03   2504.12
## 4 15   794515.29   3204.04
## 5 20  8388189.42  23280.90
## 6 25 89912683.28 187787.47
```

```
#Below is just a fun exercise to really test the two languages against each other without recursion
#Using dynamic programming allows us to go to any nth number in the Fibonacci sequence so we are simply
#The only "issue" here now is overflow for when the numbers get too large for cpp, but cpp is nearly in

nth_fibonnaci_dynamic = function(fibLimit){
  f = c()
  f[1] = 0
  f[2] = 1

  for(i in 3:fibLimit){

    f[i] = f[i-1] + f[i-2]
  }
```

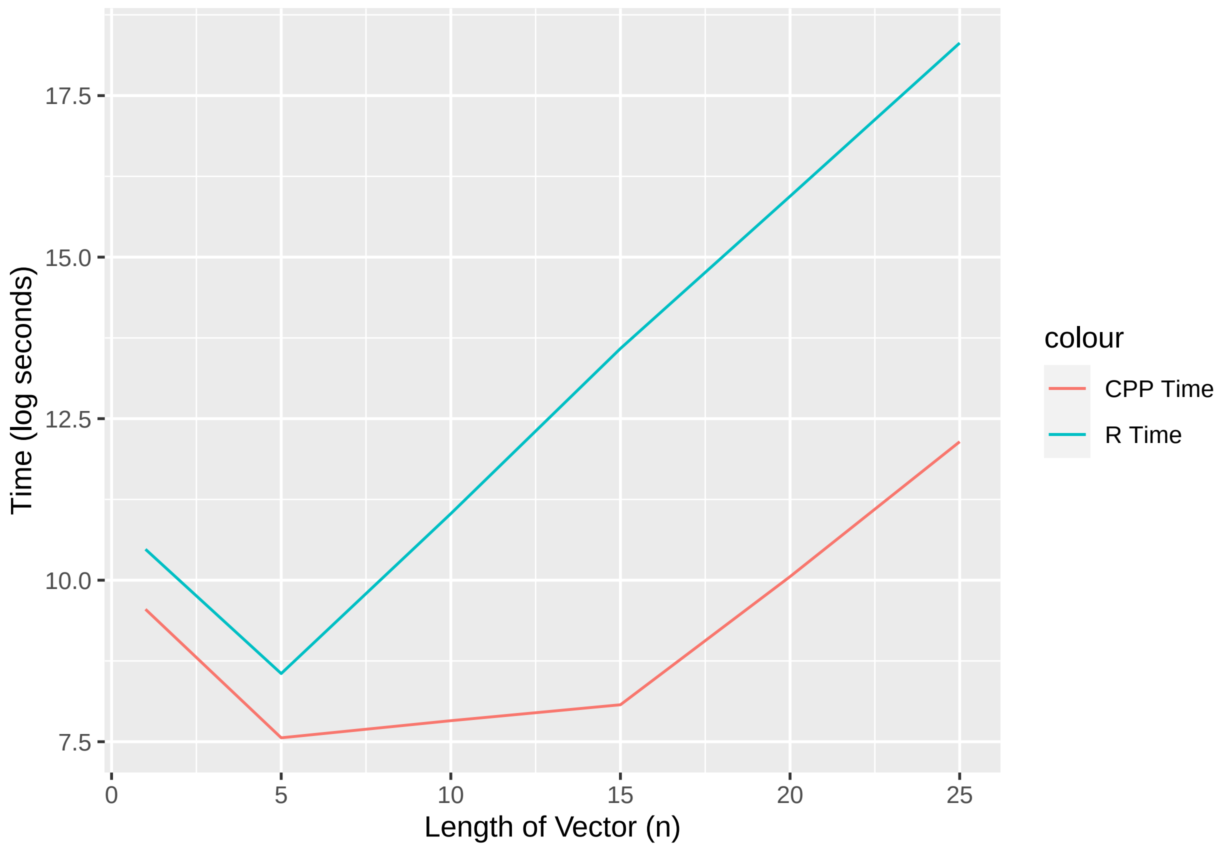
```
  return(f[fibLimit])
}



cppFunction('
   long nth_fibonnaci_cpp_dynamic(int fibLimit){
      int f[fibLimit + 2];

      f[0] = 0;
      f[1] = 1;

      for(int i = 2; i <= fibLimit; i++){
        f[i] = f[i - 1] + f[i - 2];
      }
      return f[fibLimit];
    }
')

n_dyn = c(5,10,15,20,25,40)
fib_timing_dynamic = data.frame(
  n_dyn = n_dyn,
  fib_R_dyn = numeric(length = length(n_dyn)),
  fib_cpp_dyn = numeric(length = length(n_dyn))
)

for (i in 1 : length(n_dyn)){


  fib_timing_dynamic$fib_R_dyn[i] = mean(microbenchmark(r_fib_dyn = nth_fibonnaci_dynamic(n_dyn[i]), uni

  fib_timing_dynamic$fib_R_dyn[i] = mean(microbenchmark(cpp_fib_dyn = nth_fibonnaci_cpp_dynamic(n_dyn[i

}

fib_timing_dynamic
```

```
##   n_dyn fib_R_dyn fib_cpp_dyn
## 1     5  12744.72           0
## 2    10   2144.18           0
## 3    15   2070.02           0
## 4    20   3026.15           0
## 5    25   1756.93           0
## 6    40   2150.79           0
```

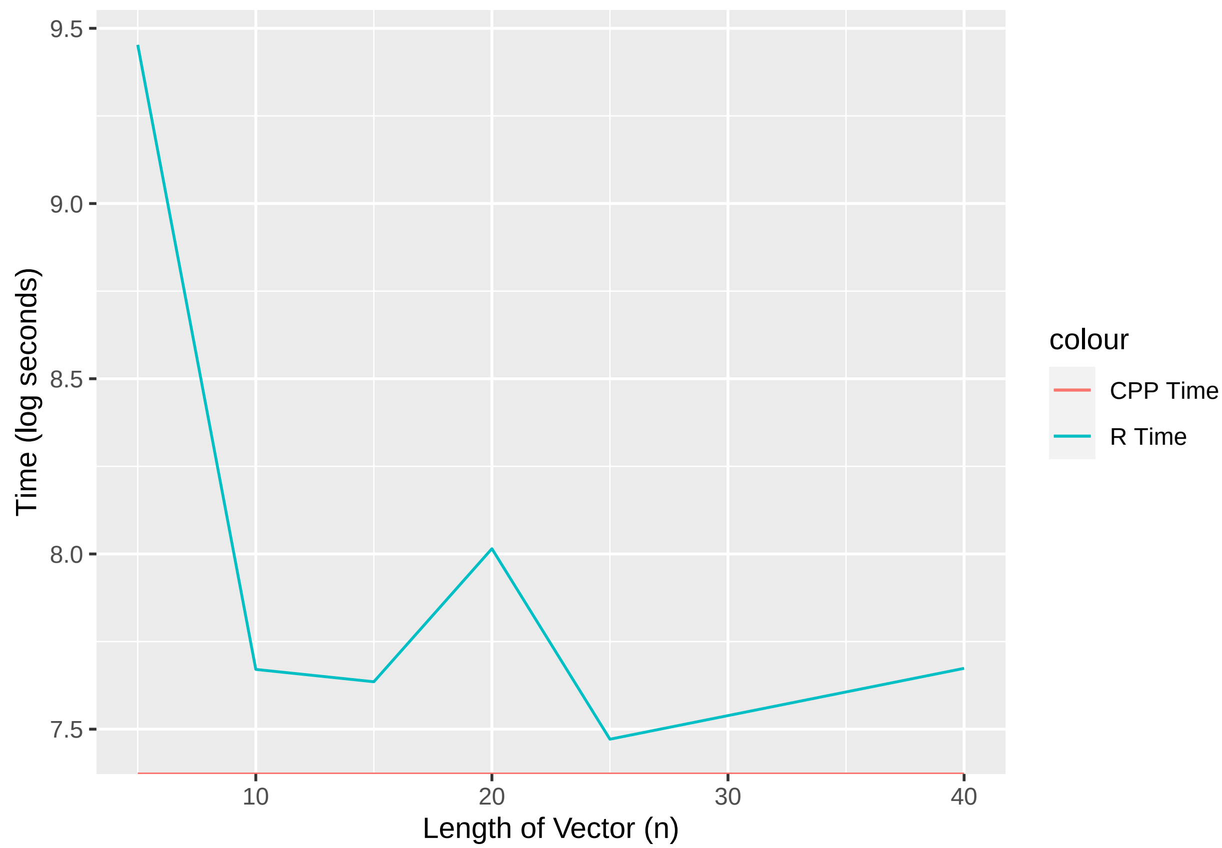
Plot the divergence of performance (in log seconds) over n using a line geometry. Use two different colors for the R and CPP functions. Make sure there's a color legend on your plot.

```
#With recursion

ggplot(fib_timing) +
  geom_line(aes(x = n, y = log(fib_R),col = "R Time")) +
  geom_line(aes(x = n, y = log(fib_cpp), col = "CPP Time")) +
```

```
  xlab("Length of Vector (n)")+
  ylab("Time (log seconds)")
```




```
#With Dynamic Programming, CPP is so close to 0 seconds it doesn't scale on axis!
ggplot(fib_timing_dynamic) +
  geom_line(aes(x = n_dyn, y = log(fib_R_dyn),col = "R Time")) +
  geom_line(aes(x = n_dyn, y = log(fib_cpp_dyn), col = "CPP Time")) +
  xlab("Length of Vector (n)")+
  ylab("Time (log seconds)")
```



Tress, bagged trees and random forests

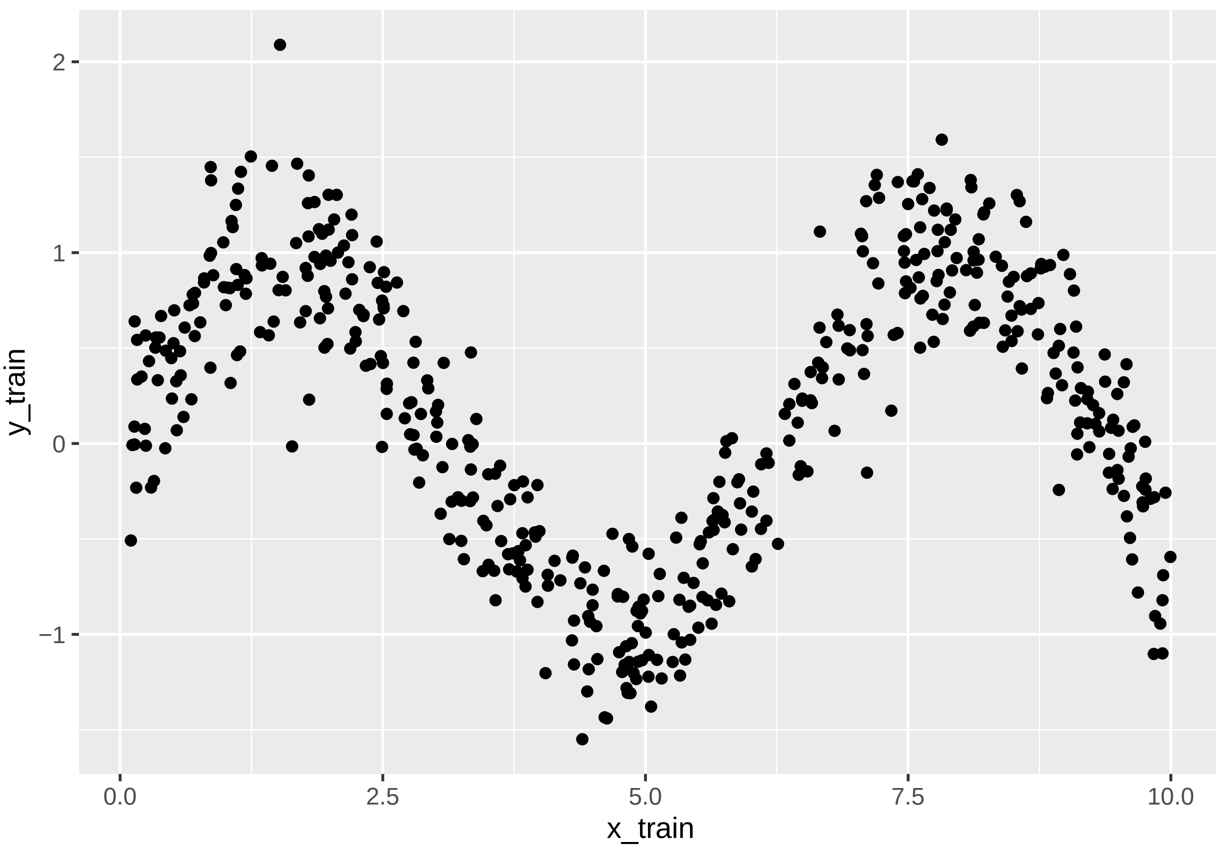
You can use the `YARF` package if it works, otherwise, use the `randomForest` package (the standard).

Let's take a look at a simulated sine curve. Below is the code for the data generating process:

```
rm(list = ls())
n = 500
sigma = 0.3
x_min = 0
x_max = 10
f_x = function(x){sin(x)}
y_x = function(x, sigma){f_x(x) + rnorm(n, 0, sigma)}
x_train = runif(n, x_min, x_max)
y_train = y_x(x_train, sigma)
```

Plot an example dataset of size 500:

```
pacman::p_load(ggplot2)

ggplot(data.frame(cbind(x_train, y_train)))+
  geom_point(aes(x=x_train, y=y_train))
```



Create a test set of size 500 as well

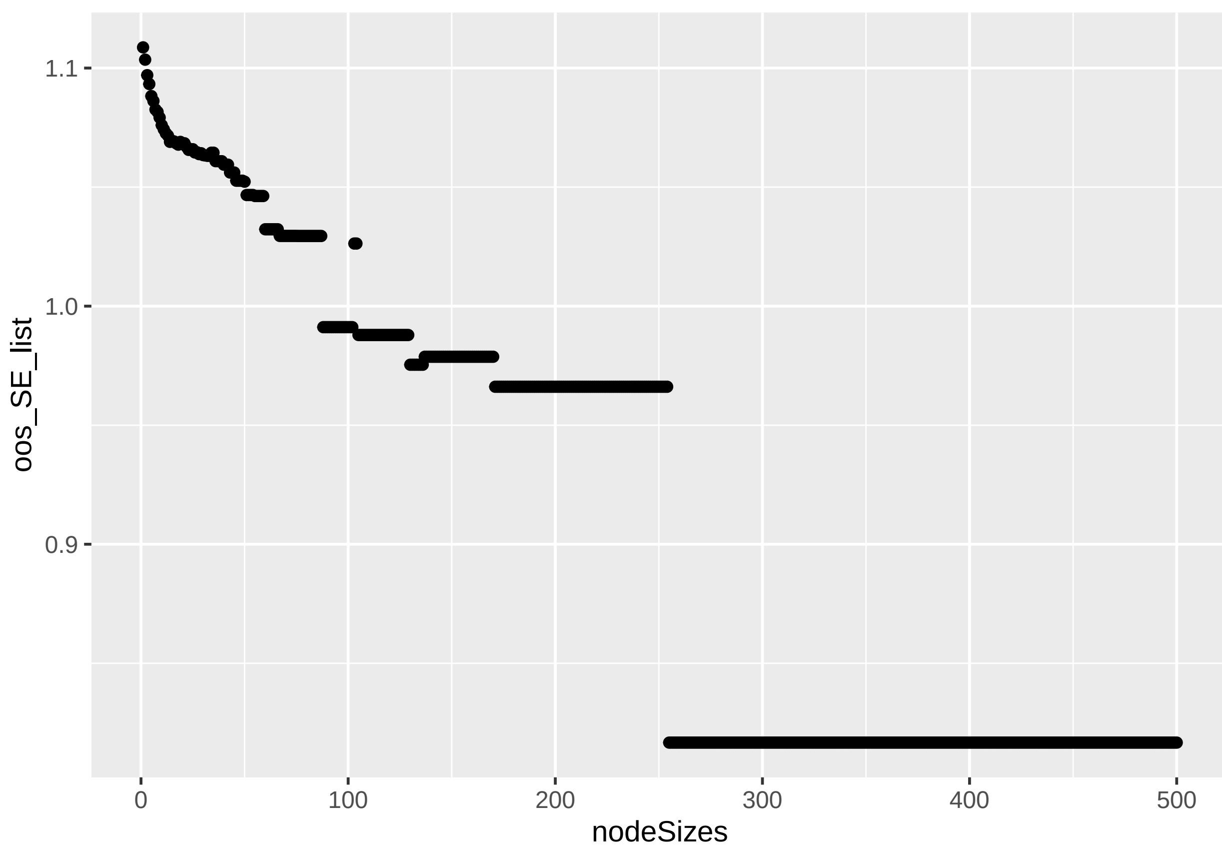
```
x_test = runif(n, x_min, x_max)
y_test = y_x(x_test, sigma)
```

Locate the optimal node size hyperparameter for the regression tree model. I believe you can use `randomForest` here by setting `ntree = 1`, `replace = FALSE`, `sampsize = n` (`mtry` is already set to be 1 because there is only one feature) and then you can set `nodesize`. Plot nodesize by out of sample s_e. Plot.

```
library('randomForest')
```

```
## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

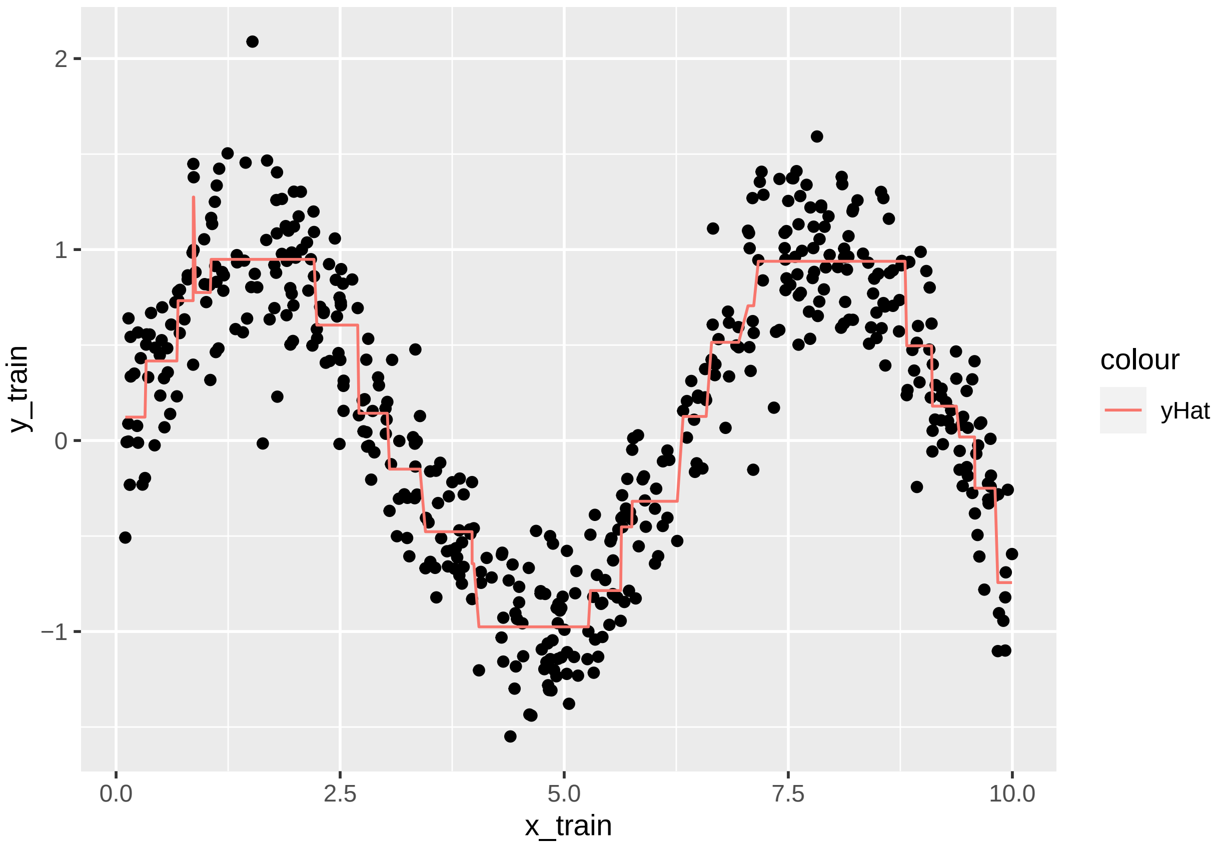
```r
nodeSizes = seq(1,500)
oos_SE_list = rep(NA,length(nodeSizes))

for(currNodeSize in nodeSizes){
  #Create a random forest model with nodesize, y_train ~ x_train
  currRandForest = randomForest(y_train ~ x_train, nodesize = currNodeSize, ntree = 1, replace = FALSE,

  #Compute error of y_hat with y_test
  y_hat = predict(currRandForest, x_test)

  res_vals = y_test - y_hat
  oos_SE = sd(res_vals)

  #Add to out of sample se list
  oos_SE_list[currNodeSize] = oos_SE
}

ggplot(data.frame(cbind(nodeSizes, oos_SE_list)))+
  geom_point(aes(x=nodeSizes, y=oos_SE_list))
```



Plot the regression tree model g(x) with the optimal node size.

```r
optRandForest = randomForest(y_train ~ x_train, nodesize = 25, ntree = 1, replace = FALSE, sampsize=n)


ggplot(data.frame(cbind(x_train,y_train)))+
```

```
  geom_point(aes(x=x_train, y=y_train))+
  geom_line(aes(x=x_train, y=predict(optRandForest, x_test), color = 'yHat'))
```



Provide the bias-variance decomposition of this DGP fit with this model. It is a lot of code, but it is in the practice lectures. If your three numbers don't add up within two significant digits, increase your resolution.

```
#TO-DO
```

```
rm(list = ls())
```

Take a sample of n = 2000 observations from the diamonds data.

```
n=2000
k = 1/5
diamonds = diamonds[sample(1:nrow(diamonds), n),]
trainSplit = diamonds[1:(nrow(diamonds)*(1-k)),]
testSplit = diamonds[(nrow(diamonds)*(1-k)+1):nrow(diamonds),]

x_train = trainSplit[,2:ncol(trainSplit)]
y_train = trainSplit$carat

x_test = testSplit[,2:ncol(testSplit)]
y_test = testSplit$carat

x_train
```

```
## # A tibble: 1,600 x 9
##    cut       color clarity depth table price     x     y     z
##    <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 Ideal     H     SI1      62.8    54   357  4.05  4.07  2.55
##  2 Very Good F     VVS1     59      61   547  4.17  4.2   2.47
##  3 Premium   I     VS2      58.9    61 18447  8.36  8.35  4.92
##  4 Very Good H     I1       63.1    60  2850  6.75  6.67  4.23
##  5 Premium   F     VS2      62.2    58 12985  7.38  7.29  4.56
##  6 Ideal     E     SI1      61.6    55  1649  5.31  5.33  3.28
##  7 Premium   F     VS1      61.6    56 15801  7.45  7.38  4.57
##  8 Premium   H     SI1      59.5    58  6025  7.02  6.96  4.16
##  9 Good      E     VS2      63.4    56  5954  6.35  6.37  4.03
## 10 Premium   D     SI2      62.4    58   574  4.31  4.28  2.68
## # ... with 1,590 more rows
```

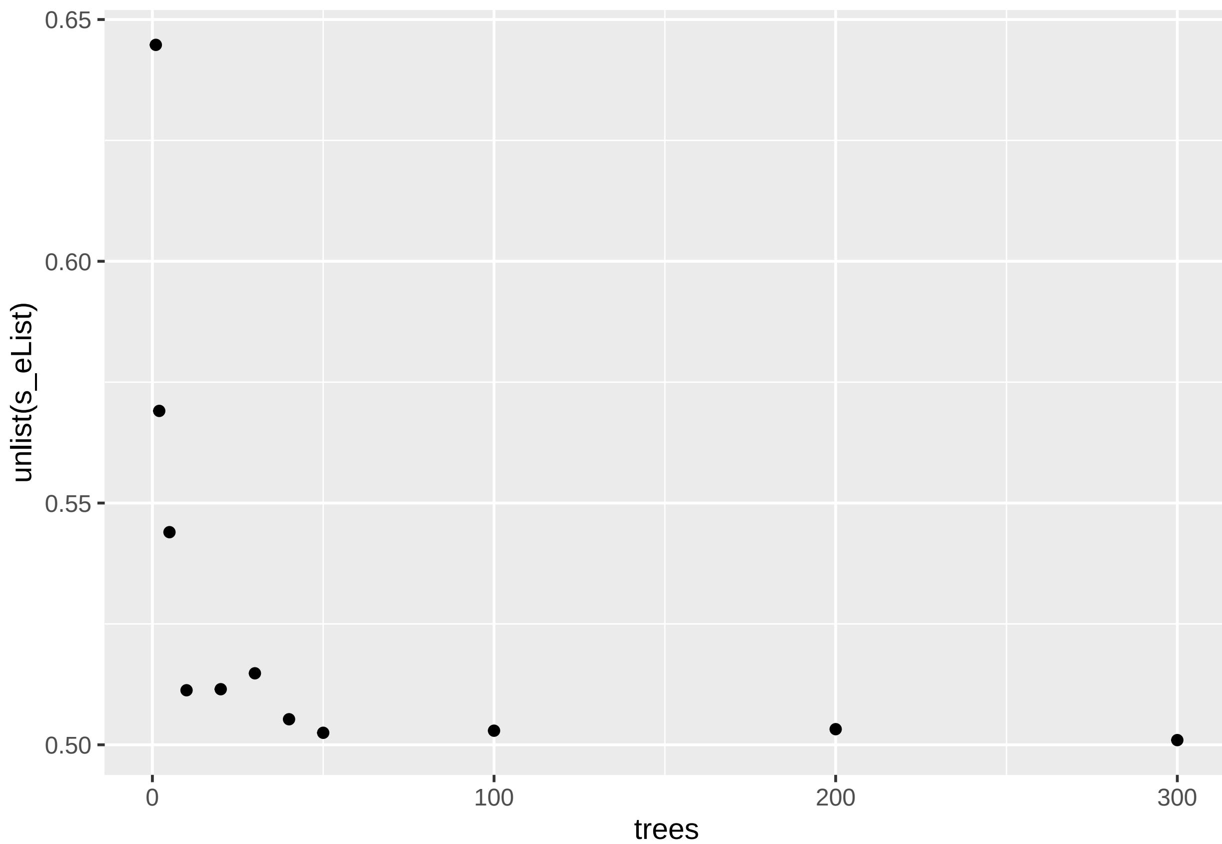
Find the bootstrap s_e for a RF model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees. If you are using the `randomForest` package, you can calculate oob residuals via `e_oob = y_train - rf_mod$predicted`. Plot.

```
#Trying with YARF rather than randomForest, takes too long for above 300 trees on my computer
trees = c(1,2,5,10,20,30,40,50,100,200,300)
s_eList = list()

for(currIndex in 1:length(trees)){


    #Bootstrap forests
    bootstrap_indices = sample(1 : (n*(1-k)), replace = TRUE)

    curr_Forest = YARF(data.frame(x=x_train[bootstrap_indices,]), y_train, calculate_oob_error = FALSE

    #Predict
    y_hats = predict(curr_Forest, data.frame(x = x_test))

    e_oob = y_train - y_hats

    s_eList[currIndex] = sd(e_oob)
}
```

```
## YARF initializing with a fixed 1 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 2 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 5 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 10 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
```

```
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 20 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 30 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 40 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 50 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 100 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 200 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 300 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
```

```
ggplot(data.frame(cbind(trees,unlist(s_eList))))+
  geom_point(aes(x=trees, y=unlist(s_eList)))
```



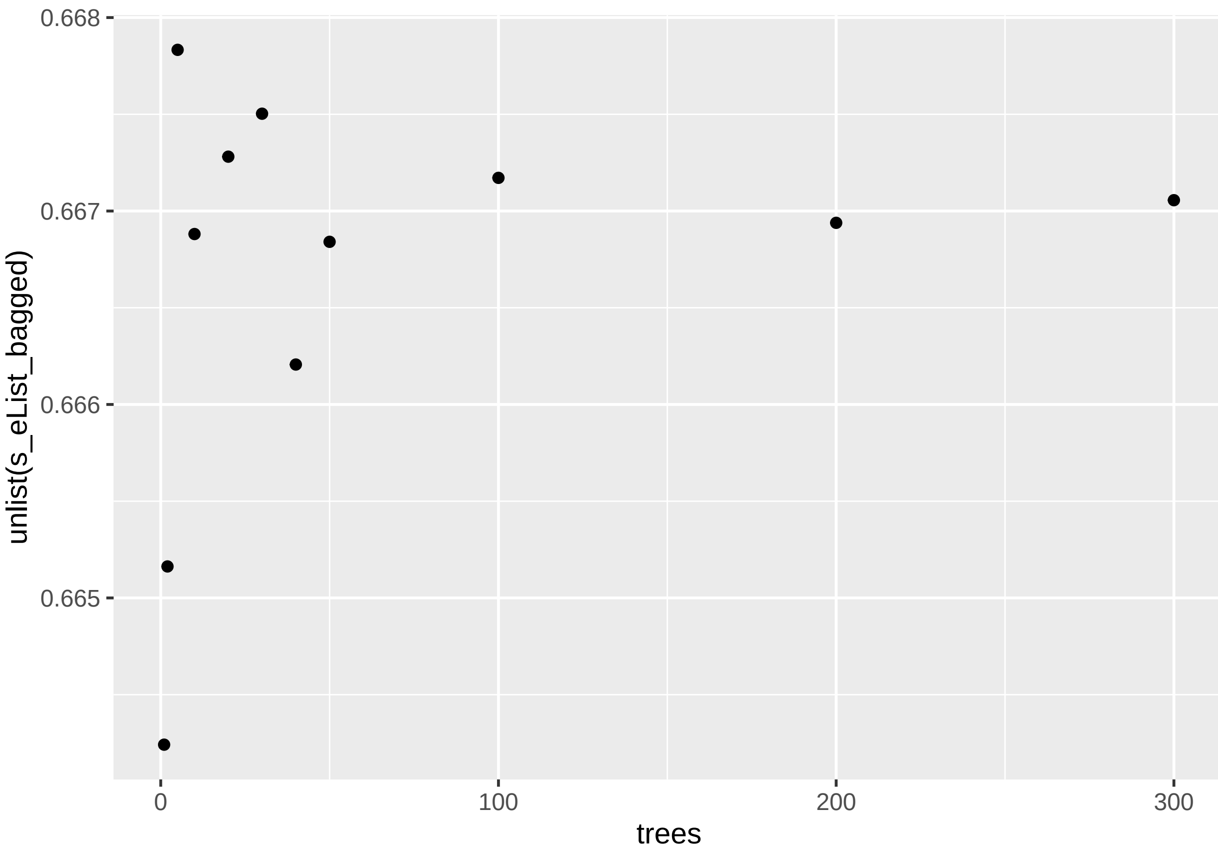
Using the diamonds data, find the oob s_e for a bagged-tree model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees. If you are using the `randomForest` package, you can create the bagged tree model via setting an argument within the RF constructor function. Plot.

```
trees = c(1,2,5,10,20,30,40,50,100,200,300)
s_eList_bagged = list()

for(currIndex in 1:length(trees)){


    curr_Forest = YARFBAG(data.frame(x=x_train), y_train, calculate_oob_error = FALSE, num_trees = trees

    #Predict
    y_hats = predict(curr_Forest, data.frame(x = x_test))

    e_oob = y_train - y_hats

    s_eList_bagged[currIndex] = sd(e_oob)
}
```

```
## YARF initializing with a fixed 1 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 2 trees...
## YARF factors created...
```

```
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 5 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 10 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 20 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 30 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 40 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 50 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 100 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 200 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
## YARF initializing with a fixed 300 trees...
## YARF factors created...
## YARF after data preprocessed... 9 total features...
## Beginning YARF regression model construction...done.
```

```
ggplot(data.frame(cbind(trees,unlist(s_eList_bagged))))+
  geom_point(aes(x=trees, y=unlist(s_eList_bagged)))
```



What is the percentage gain / loss in performance of the RF model vs bagged trees model?

```
#Bagging decreases the variance by quite a bit which is evident by the y-axis values
((unlist(s_eList_bagged)-unlist(s_eList))/(unlist(s_eList)))*100
```

```
##  [1]  3.021421 16.888807 22.769667 30.433889 30.456282 29.665594 31.847678
##  [8] 32.709804 32.660615 32.531075 33.153065
```

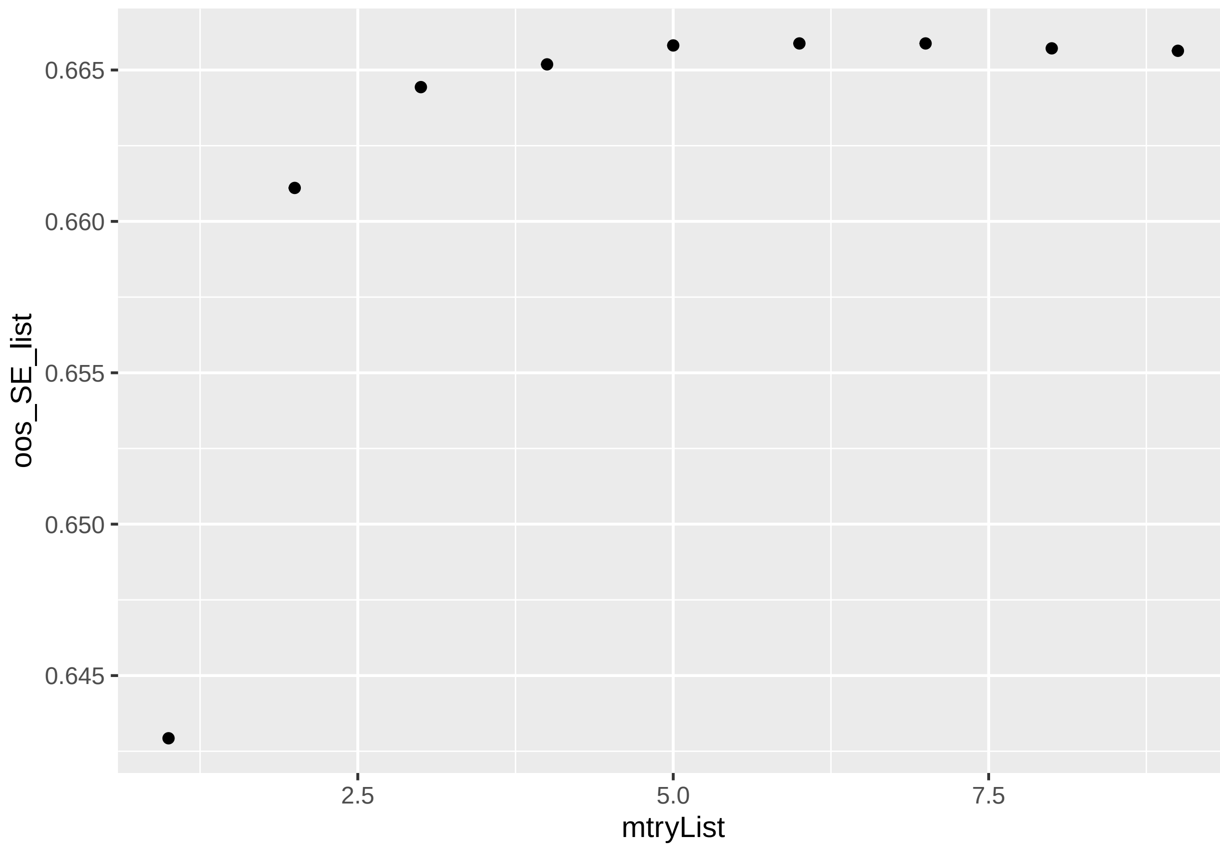
```
#This is showing that there is a considerable jump in error of the bagged trees
#We would consider this not to be good, but the variance has decreased which is better in our case
#This represents a better "average" tree rather than simply getting lucky with 1 good tree without bagg
```

Plot oob s_e by number of trees for both RF and bagged trees using a long data frame.

Build RF models for 500 trees using different `mtry` values: 1, 2, ... the maximum. That maximum will be the number of features assuming that we do not binarize categorical features if you are using `randomForest` or the number of features assuming binarization of the categorical features if you are using `YARF`. Calculate oob s_e for all mtry values.

```
maxFeat = 9
mtryList = seq(1,9)
oos_SE_list = rep(NA,length(mtryList))

for(currFeat in mtryList){

  #Create a random forest model with nodesize, y_train ~ x_train
```

```
  currRandForest = randomForest(x_train, y_train, num_trees = 500, mtry = currFeat,  replace = FALSE)

  #Compute error of y_hat with y_test
  y_hat = predict(currRandForest, x_test)

  e_oob = y_train - y_hat

  #Add to out of sample se list
  oos_SE_list[currFeat] = sd(e_oob)
}

ggplot(data.frame(cbind(mtryList, oos_SE_list)))+
  geom_point(aes(x=mtryList, y=oos_SE_list))
```



Plot oob s_e by mtry.

```
#Plot is above
```

```
rm(list = ls())
```

Take a sample of n = 2000 observations from the adult data.

```
pacman::p_load_gh("coatless/ucidata")
library(tidyr)
adult = adult%>%drop_na()
```

```
n_samp = 2000
adultSample = adult[sample(1:nrow(adult), n_samp),]

k = 1/5
trainSplit = adultSample[1:(nrow(adultSample)*(1-k)),]
testSplit = adultSample[(nrow(adultSample)*(1-k)+1):nrow(adultSample),]

x_train = trainSplit[,2:ncol(trainSplit)]
y_train = trainSplit$income

x_test = testSplit[,2:ncol(testSplit)]
y_test = testSplit$income
```

Using the adult data, find the bootstrap misclassification error for an RF model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees.

```
trees = c(1,2,5,10,20,30,40,50,100,200,300)
m_error_rf = list()
m_error_bagged = list()
for(currTrees in 1:length(trees)){

  mod_bag = YARFBAG(x_train, y_train, num_trees = trees[currTrees])

  currForest = YARF(x_train, y_train, num_trees = trees[currTrees], bootstrap_indices = mod_bag$bootstra

  m_error_rf[currTrees] = currForest$misclassification_error
  m_error_bagged[currTrees] = mod_bag$misclassification_error

}
```

```
## YARF initializing with a fixed 1 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 1 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 2 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 2 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 5 trees...
## YARF factors created...
```

```
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 5 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 10 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 10 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 20 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 20 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 30 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 30 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 40 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 40 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 50 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 50 trees...
```

```
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 100 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 100 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 200 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 200 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 300 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
## YARF initializing with a fixed 300 trees...
## YARF factors created...
## YARF after data preprocessed... 94 total features...
## Beginning YARF classification model construction...done.
## Calculating OOB error...done.
```

```
m_error_rf
```

```
## [[1]]
## [1] 0.03407155
##
## [[2]]
## [1] 0.009625668
##
## [[3]]
## [1] 0.01935038
##
## [[4]]
## [1] 0.006321113
##
## [[5]]
## [1] 0
##
## [[6]]
## [1] 0
```

```
##
## [[7]]
## [1] 0
##
## [[8]]
## [1] 0
##
## [[9]]
## [1] 0
##
## [[10]]
## [1] 0
##
## [[11]]
## [1] 0
```

```
m_error_bagged
```

```
## [[1]]
## [1] 0
##
## [[2]]
## [1] 0
##
## [[3]]
## [1] 0
##
## [[4]]
## [1] 0
##
## [[5]]
## [1] 0
##
## [[6]]
## [1] 0
##
## [[7]]
## [1] 0
##
## [[8]]
## [1] 0
##
## [[9]]
## [1] 0
##
## [[10]]
## [1] 0
##
## [[11]]
## [1] 0
```

Using the adult data, find the bootstrap misclassification error for a bagged-tree model using 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000 trees. Plot.

```r
#In the above cell, bagged tree model is included with its own error
```

What is the percentage gain / loss in performance of the RF model vs bagged trees model?

```r
#TO-DO
```

Plot bootstrap misclassification error by number of trees for both RF and bagged trees using a long data frame.

```r
#TO-DO
```

Build RF models for 500 trees using different `mtry` values: 1, 2, ... the maximum (see above as maximum is defined by the specific RF algorithm implementation). Plot.

```r
#TO-DO
```

Plot bootstrap misclassification error by `mtry`.

```r
#TO-DO
```

```r
rm(list = ls())
```