Challenge ???

Introduction

The main aim of this project is to create an $argmin$ of the estimate_π_resample function found below.

```r
# Empirical Methods -------------------------------------------------------

# generate uniformly distributed points
generate_points <- function(n){
  XY <- matrix(runif(2*n), nrow = n, ncol = 2)
  return(XY)}

get_distance <- function(XY){
  dist <- sqrt(XY[,1]^2 + XY[,2]^2) < 1
  return(dist)}

approx_pi <- function(dist){
  return(4*sum(dist)/length(dist))}

# Main Function -----------------------------------------------------------

estimate_pi_empirical <- function(n){
  dist <- get_distance(generate_points(n = n))
  return(approx_pi(dist = dist))}
```

How does the empirical Algo work

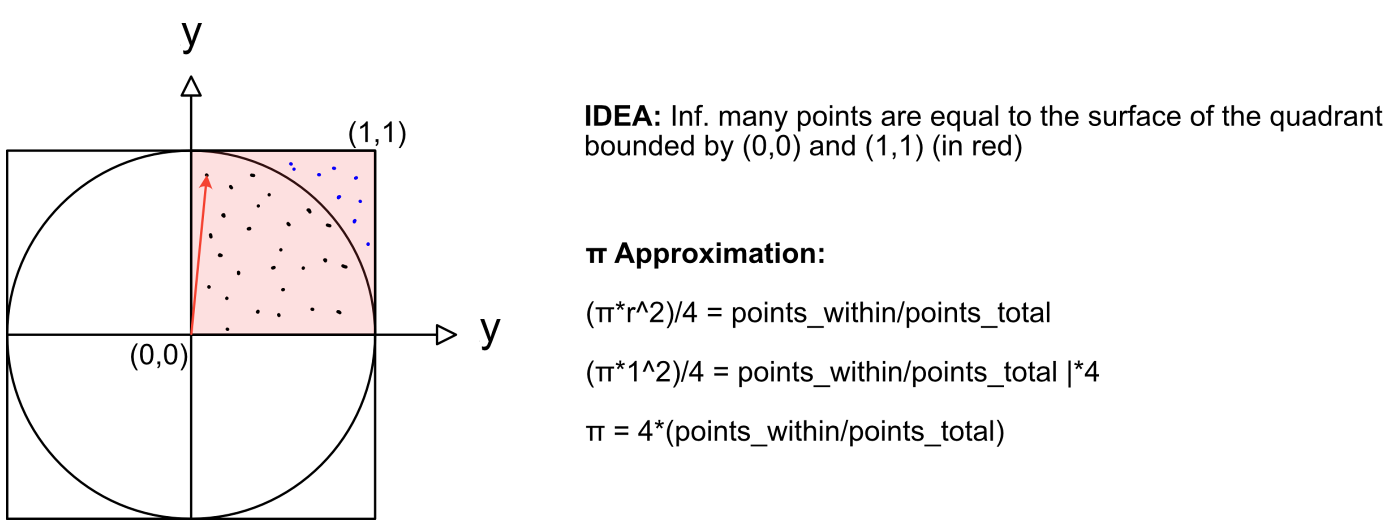

Figure 1: Schematic of the empirical algo.

Steps of the Algo:

1. Generate $2n$ uniformly distributed values within $[0, 1]$ (`runif(2n, 0, 1)`). $2n$ because there are x and y values.

2. Calculate euclidean distance from origin (0,0), i.e. vector norm of value (see arrow in Figure). $\sqrt{x^2 + y^2} < 1$ are all the points that are within the unit circle, since the radius is 1.

3. Approximate π using: $\pi \approx 4 \cdot \frac{\text{points\_within}}{\text{points\_all}}$

The Resampling-Algo

Why is the legitimacy of this algo? The empirical algo gains accuracy by sampling more points, obviously. However, in *R*, matrices/vectors larger than $1e7$ are generated slowly. Therefore, I thought about generating a matrix of points within the unit quadrant as above, but with fewer points, e.g., $1e6$.

To compensate the loss of information, I would create a distribution of the ratios $\frac{\text{points\_within}}{\text{points\_all}}$. To do that, I calculate the ratio on a row-by-row basis, which means I would get many estimates for the ratio.

Next, I used *fitdistrplus* to evaluate best-fitting `shape` and `rate` parameters of the γ-distribution. Once, the params are fixed, I can sample ratios from this γ-distribution to approximate the pi as described above.

The idea was nice, but for the moment the resampling-algo is neither faster nor more accurate (Repo). So, the idea is to optimize its parameters.

Parameters of the Resampling-Algo

Let's call the parameters θ. The function can be found in the Appendix.

1. θ_1 aka n : Size of the elements in the matrix for the ratio calculation. To put it simply, how many points are generated in `runif(n,0,1)`.

2. θ_2 aka *samplingSize*: This is the row number that you want to set for the matrix that has n elements of uniformly drawn values. This number should be $\text{mod}10 == 0$ and not smaller than 100.

3. θ_3 aka *outputLength*: How many values are drawn from theγ-distribution generated in Step 2. The values drawn from this distribution represent the ratio of points within the unit circle.

Scoring Function

The scoring function is super easy, it is just the accuracy of the estimates calculated as the difference to real π, namely, $\mathit{accuracy} = |\hat{\pi} - \pi|$, where $\hat{\pi}$ is the estimated value from the function. The lower the *accuracy* value the better.

Aim

First, I have to admit that I wrote this code one morning and I did not try to make it as optimal as possible. Hence, if there is a way to optimize the Main-Function, go for it!

The main aim is to find the optimal parameters for the Resampling-Algo, which we will call Π. Hence, we need $\vec{\theta}^* = \underset{\vec{\theta}}{\operatorname{argmin}}\, \Pi(\vec{\theta})$, where $\vec{\theta}$ is the vector containing all the parameters θ described above. Consequently,$\vec{\theta}^*$ is the vector containing the optimized parameters.

Some words about constraints, the Resampling-Algo should be either faster than the Empirical-Algo with similar accuracy, or more accurate with similar speed. Thus, I will lay an upper limit upon thy, lets say, $1e6$ for θ_1 and θ_3. The sampling size, θ_2 should be around a factor of 100 smaller than θ_1, since it is the length of the row-vectors taken from n, from which the mean is calculated.

Thus, there is a trade-off between accuracy and number of ratios calculated. If θ_2 is small, many estimates are generated, but they are inaccurate, whereas if θ_2 is big, the number of ratios is small, but the accuracy is low. Here are some plots about how the parameters evolve from an lower to an upper bound, without interaction of other parameters.

Appendix

```
# Resampling Methods ------------------------------------------------------

# calculate ratio of points within
calc_ratio <- function(n, samplingSize, plot){
  distMatrix <- matrix(get_distance(generate_points(n)), nrow = samplingSize)
  ratioVector <- rowSums(distMatrix)/ncol(distMatrix)

  if (plot){
    hist(ratioVector, freq = F)
  }
  return(ratioVector)
}

# fit a gamma distribution to ratio data set and sample from gamma
generate_gamma <- function(ratioVector, outputLength, plot){
  thetaGamma <- fitdistr(ratioVector, "gamma")$estimate

  # histogram
  if (plot){
    hist(rgamma(outputLength,shape = thetaGamma[1],rate = thetaGamma[2]),
         add = T, col = rgb(0.9,0.1,0.1,0.2), freq = F)
  }
  return(rgamma(outputLength,shape = thetaGamma[1],rate = thetaGamma[2]))
}

# approximate pi (same as above).
approx_pi_resample <- function(rgammaVec){
  withinCircle <- mean(rgammaVec)
  outsideCircle <- 1 - withinCircle

  return(4*(withinCircle/(withinCircle+outsideCircle)))
}

# Main Function -----------------------------------------------------------

estimate_pi_resampled <- function(n,
                                  outputLength = 1e6,
                                  samplingSize = 1e5,
                                  plot = F){

  if(!require(fitdistrplus)){
    message("Install 'fitdistrplus' first!")
    return(NULL)
  }

  # generate ratios from n
  # create gamma distribution
  rG <- generate_gamma(calc_ratio(n, samplingSize = samplingSize, plot = plot
   ),
                       plot = plot, outputLength = outputLength)

  # use resampled data to approx pi
  return(approx_pi_resample(rG))
}
```