Data Generation

Yuchen Li (li215), section 1UG

April 14, 2019

Contents

Team Powers

Standard Probability Distributions

Normal

```
genNormalPowers <- function(n, mean=0, sd=1) {
  # INPUT:
  # n is the number of teams

  # OUTPUT:
  # returns a vector of team powers, sorted in decreasing order
  powers <- rnorm(n, mean, sd)
  return(sort(abs(powers), decreasing=TRUE))
}
```

Accept-Reject

```
# Reference: adapted from Yuchen Li (li215), HW2, Exercise 4

acceptReject <- function(nsim, f, min, max, M) {
  # INPUT:
  # nsim is the number of simulations
  # f is the target distribution
  # min is the min value in the domain of f
  # max is the max value in the domain of f
```

```
  # max
  # M >= sup{f(x)}

  # OUTPUT:
  # returns a vector of random variates sampled from f, using the
  #   Accept-Reject method with Unif(min, max) as the reference distribution
  k1 = 0              # counter for accepted samples
  j1 = 0              # number of iterations required to get desired sample size
  y1 = numeric(nsim)    # storing the sample
  while(k1 < nsim){
      u = runif(1)
      x = runif(1, min, max)  # random variate from reference distribution
      g1 = 1
      if (u < f(x) / M / g1) {
        # condition of accepting x in our sample
        k1 = k1 + 1
        y1[k1] = x
      }
      j1 = j1 + 1
  }
  return(sort(y1, decreasing=TRUE))
}
```

Inverse CDF

```
inverseCDF <- function(n, inv_cdf) {
  # INPUT:
  # n is the number of simulations
  # inv_cdf is the inverse CDF function for f

  # OUTPUT:
  # returns a vector of random variates sampled from PDF f,
  #   using the Inverse CDF method
  u = runif(n)
  y = numeric(n)
  for (i in 1:n) {
    y[i] = inv_cdf(u[i])
  }
  return(sort(y, decreasing=TRUE))
}
```

Cross-Team Winning Probabilities

```
genCrossTeamWinningProbabilities <- function(powers) {
  # INPUT:
  # powers is the teams powers

  # OUTPUT:
  # returns an n x n matrix M where M_{ij} is the probability of team-i beating team-j
  n = length(powers)
```

```
  probs = matrix(nrow=n, ncol=n)
  for (i in 1:n) {
    for (j in 1:n) {
      probs[i,j] = powers[i] / (powers[i] + powers[j])
    }
  }
  return(probs)
}
```

Seeding

What are the other good methods than random selection? (In the data generation part, we do not have actual competition data yet.)

```
# Example
sample(1:8, size=2)
```

```
## [1] 8 7
```

Examples

Normal(0, 1) team powers

```
genCrossTeamWinningProbabilities(
 genNormalPowers(4)
)
```

```
##             [,1]        [,2]       [,3]      [,4]
## [1,] 0.500000000 0.682728874 0.83305251 0.9979633
## [2,] 0.317271126 0.500000000 0.69869183 0.9956276
## [3,] 0.166947492 0.301308169 0.50000000 0.9899190
## [4,] 0.002036687 0.004372444 0.01008096 0.5000000
```

Normal(4, 10) team powers

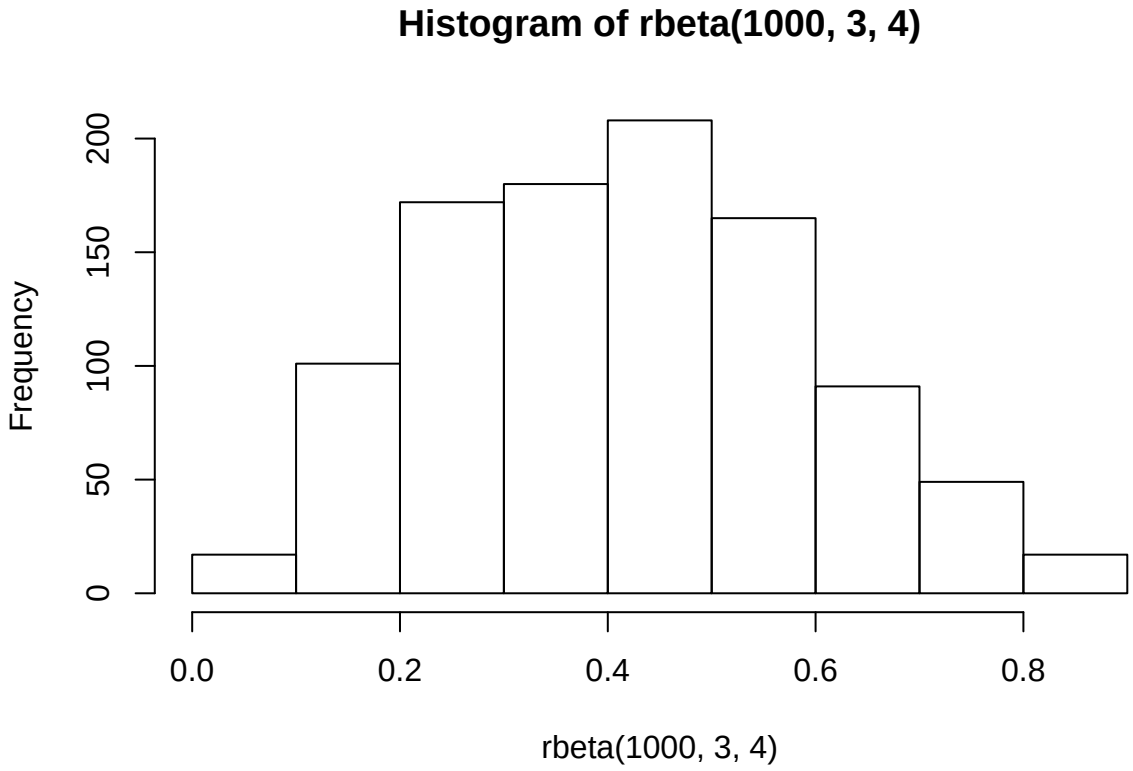
Note the probabilities are closer to 0.5

```
genCrossTeamWinningProbabilities(
 genNormalPowers(4, mean=10, sd=0.1)
)
```

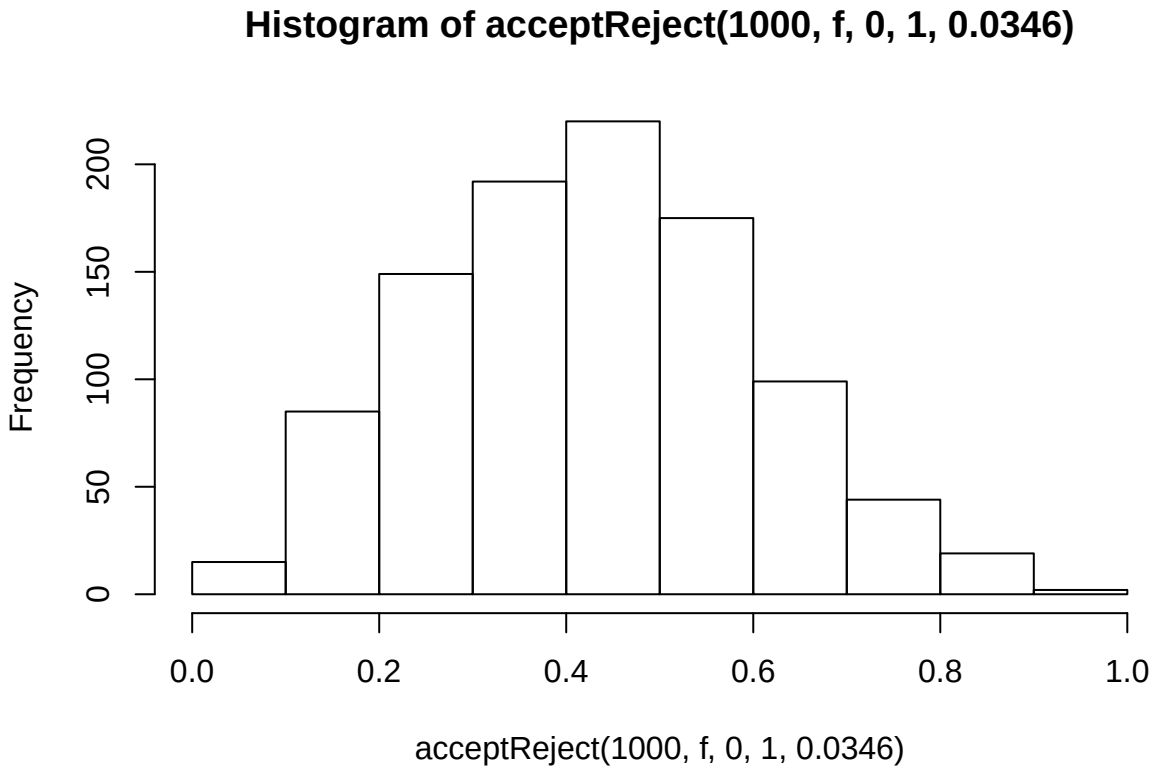
```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.5008158 0.5050294 0.5052694
## [2,] 0.4991842 0.5000000 0.5042136 0.5044536
## [3,] 0.4949706 0.4957864 0.5000000 0.5002401
## [4,] 0.4947306 0.4955464 0.4997599 0.5000000
```

Beta(3, 4) team powers using Accept-Reject

```
# Test `acceptReject`
f <- function(x) {return(x^2 * (1-x)^3)}
hist(rbeta(1000, 3, 4))
```




```
hist(acceptReject(1000, f, 0, 1, 0.0346))
```

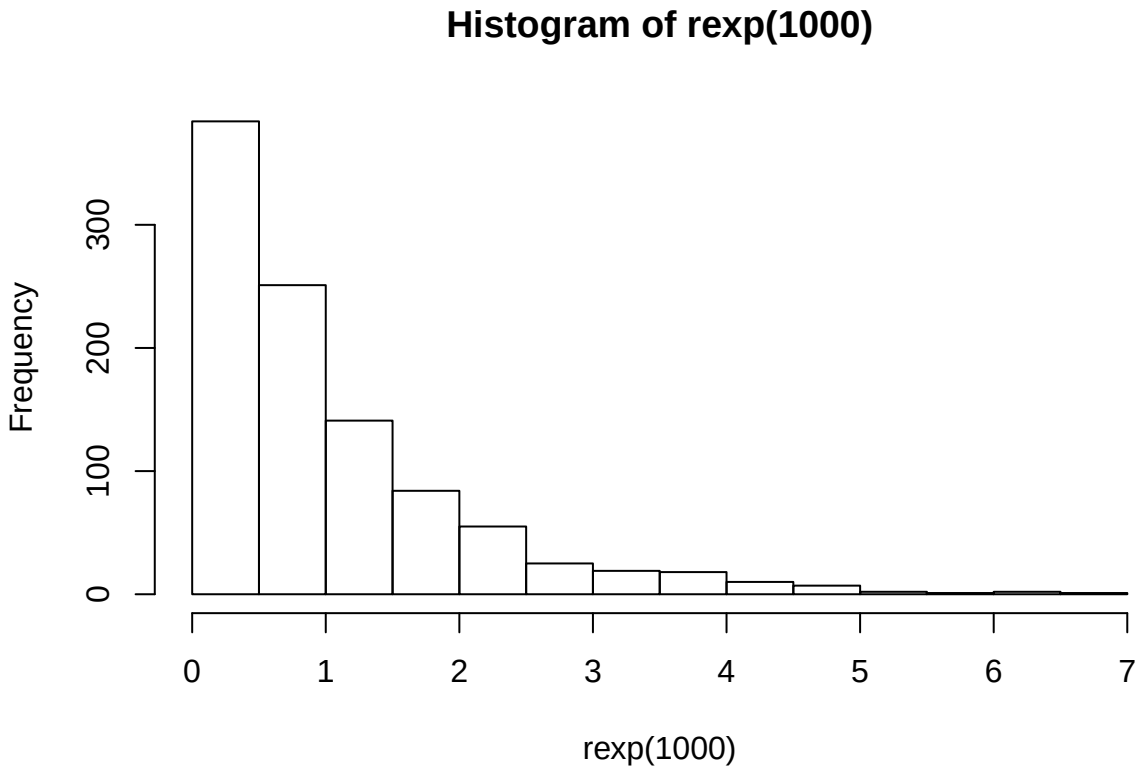



```
# Actual
genCrossTeamWinningProbabilities(
 acceptReject(4, f, 0, 1, 0.0346)
)
```

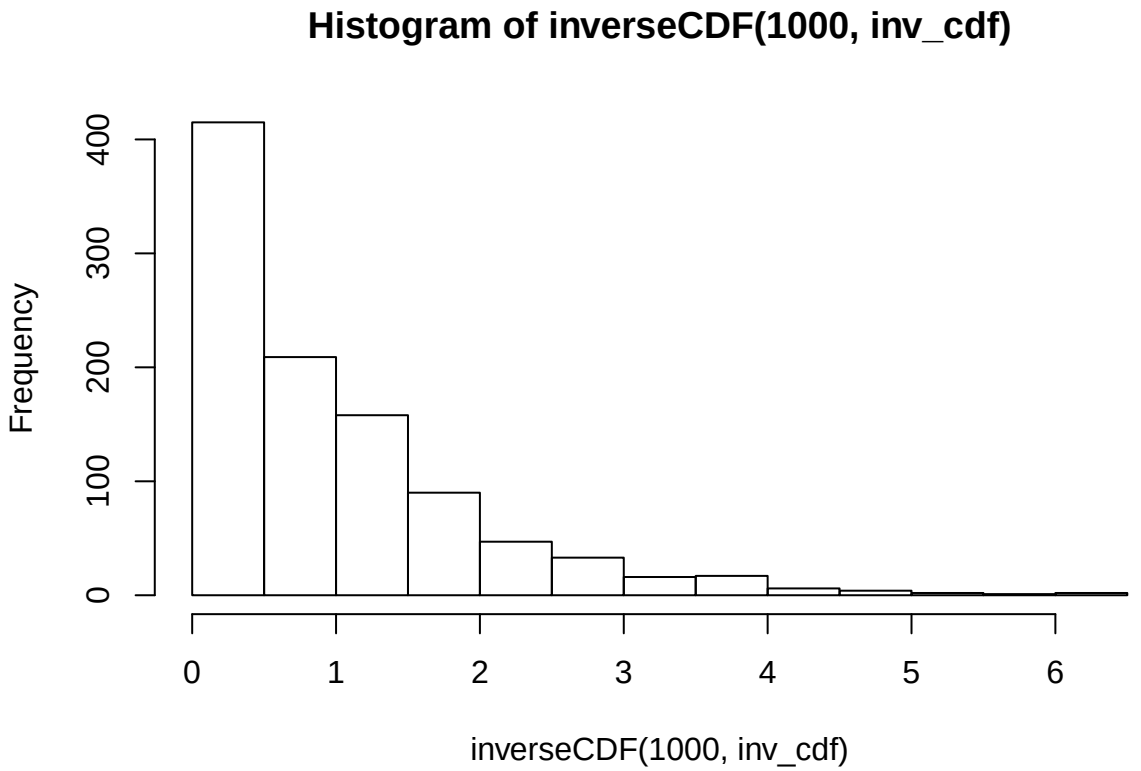
```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.5858303 0.6508705 0.7083228
## [2,] 0.4141697 0.5000000 0.5685931 0.6319282
## [3,] 0.3491295 0.4314069 0.5000000 0.5657137
## [4,] 0.2916772 0.3680718 0.4342863 0.5000000
```

Exp(1) team powers using Inverse CDF

```
# Test `inverseCDF`
inv_cdf <- function(x) {return(-log(x))}
hist(rexp(1000))
```




```
hist(inverseCDF(
    1000,
    inv_cdf
  )
)
```



```r
# Actual
genCrossTeamWinningProbabilities(
 inverseCDF(
    4,
    inv_cdf
  )
)
```

```
##            [,1]       [,2]      [,3]      [,4]
## [1,] 0.50000000 0.80916437 0.9443095 0.9842207
## [2,] 0.19083563 0.50000000 0.7999618 0.9363484
## [3,] 0.05569047 0.20003822 0.5000000 0.7862567
## [4,] 0.01577928 0.06365162 0.2137433 0.5000000
```