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] 6 3
```

Examples

Normal(0, 1) team powers

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

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.6279637 0.6904182 0.7686506
## [2,] 0.3720363 0.5000000 0.5691991 0.6631172
## [3,] 0.3095818 0.4308009 0.5000000 0.5983591
## [4,] 0.2313494 0.3368828 0.4016409 0.5000000
```

Normal(10, 0.01) team powers

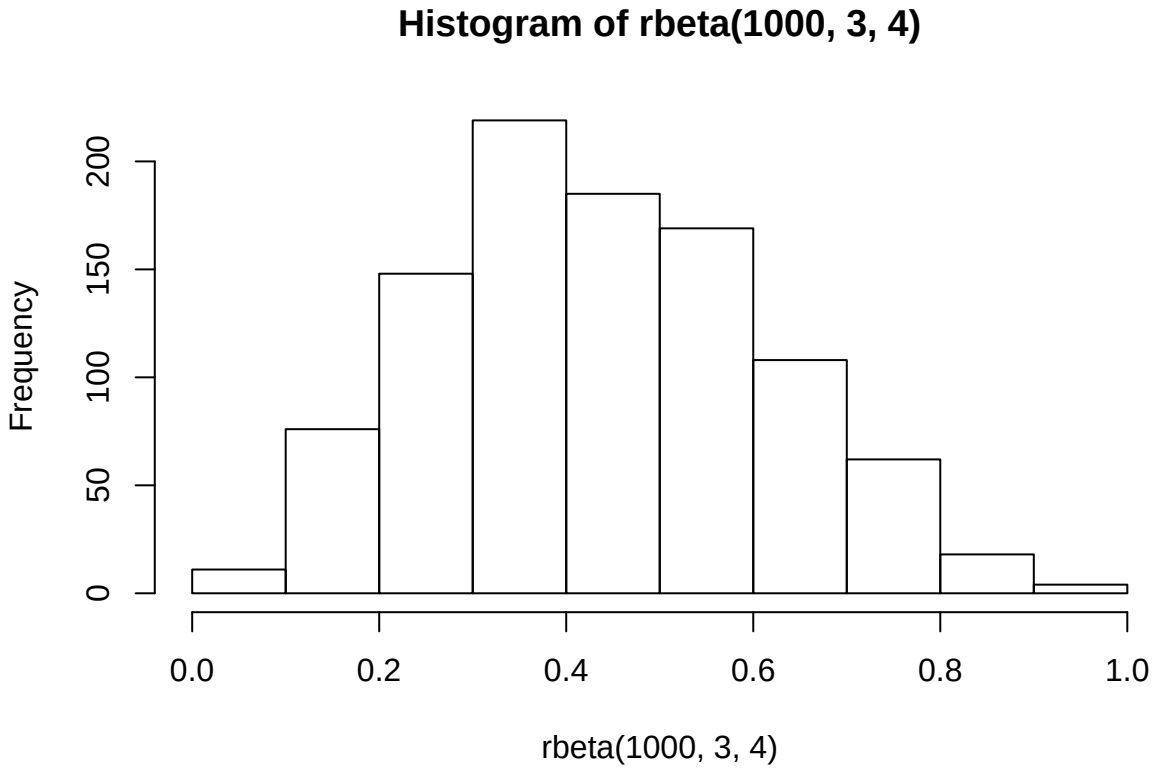
Note the probabilities are closer to 0.5

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

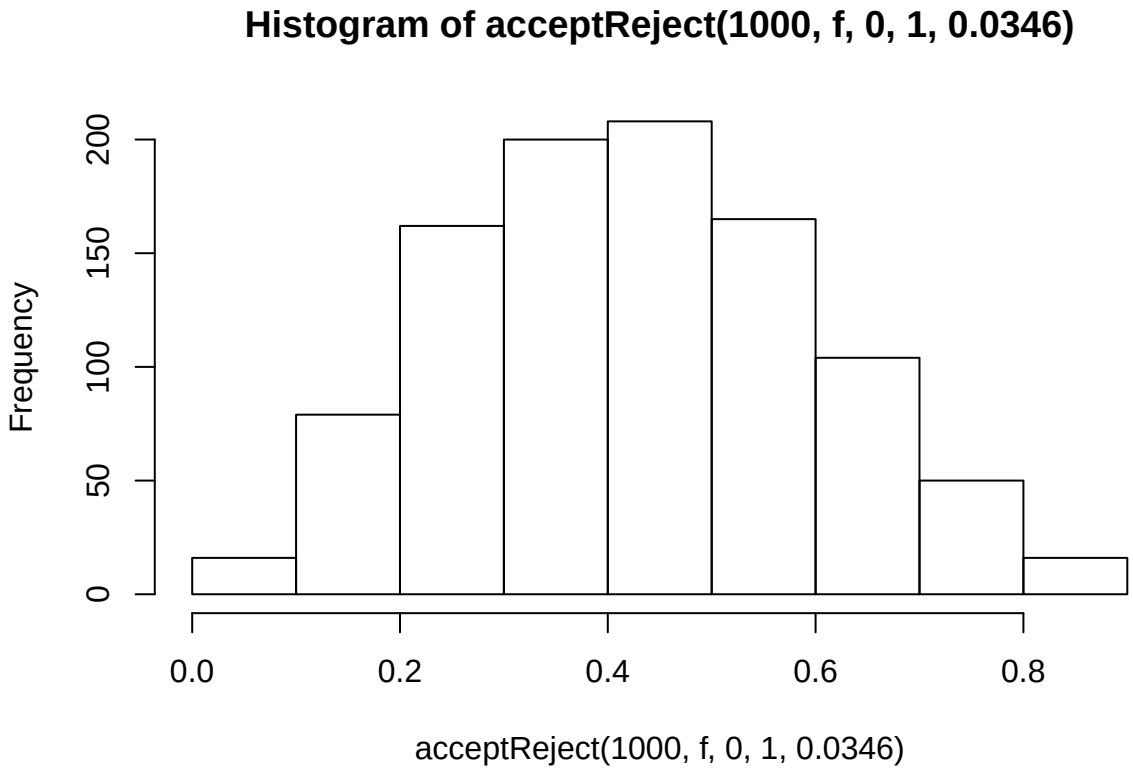
```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.5001309 0.5056079 0.5074967
## [2,] 0.4998691 0.5000000 0.5054770 0.5073658
## [3,] 0.4943921 0.4945230 0.5000000 0.5018891
## [4,] 0.4925033 0.4926342 0.4981109 0.5000000
```

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))
```

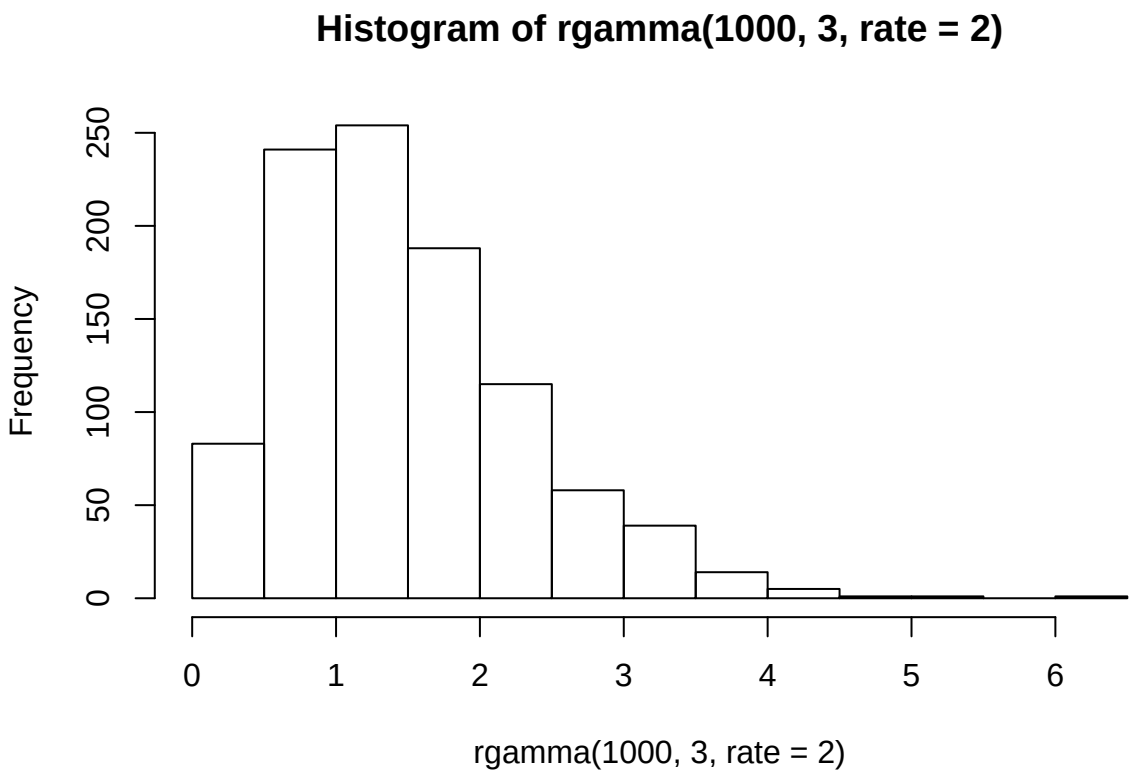



```
# Actual
genCrossTeamWinningProbabilities(
 acceptReject(4, f, 0, 1, 0.0346)
)
```

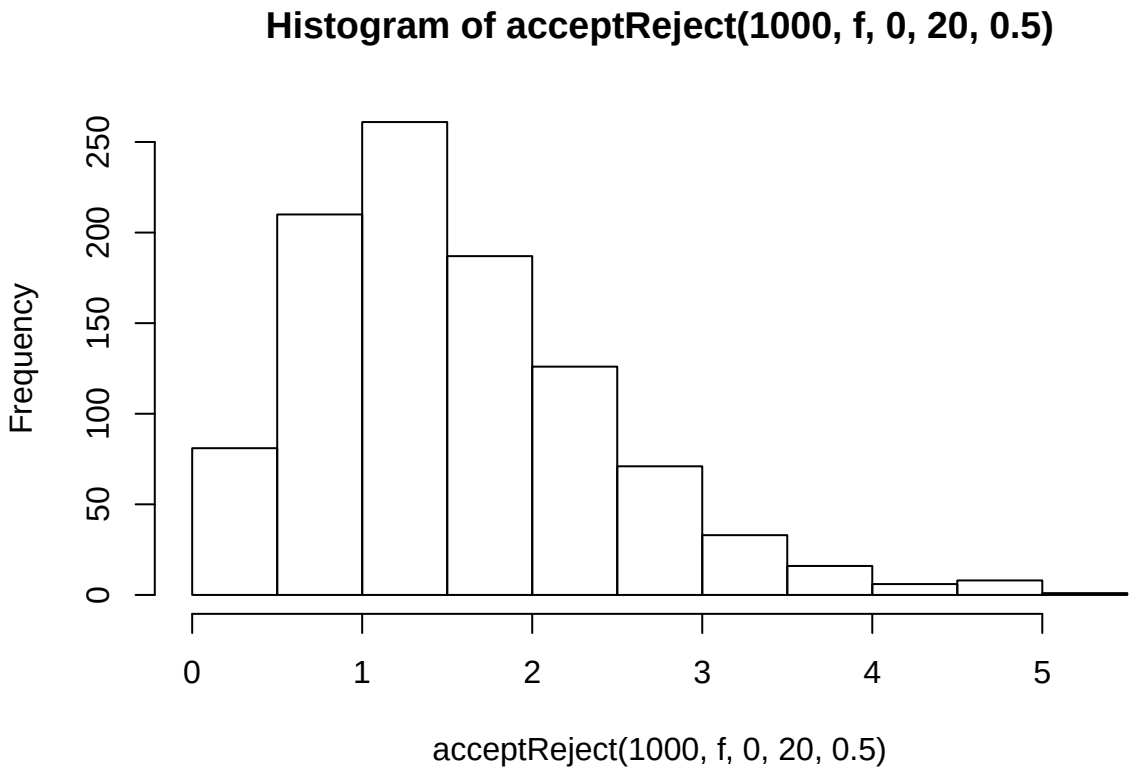
```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.6141587 0.6382600 0.7792108
## [2,] 0.3858413 0.5000000 0.5257254 0.6891709
## [3,] 0.3617400 0.4742746 0.5000000 0.6666903
## [4,] 0.2207892 0.3108291 0.3333097 0.5000000
```

Gamma(3, 2) team powers using Accept-Reject

```
# Test `acceptReject`
f <- function(x) {
    return(2^3 / gamma(3) * x^2 * exp(-2*x))
}
hist(rgamma(1000, 3, rate=2))
```




```
hist(acceptReject(1000, f, 0, 20, 0.5))
```



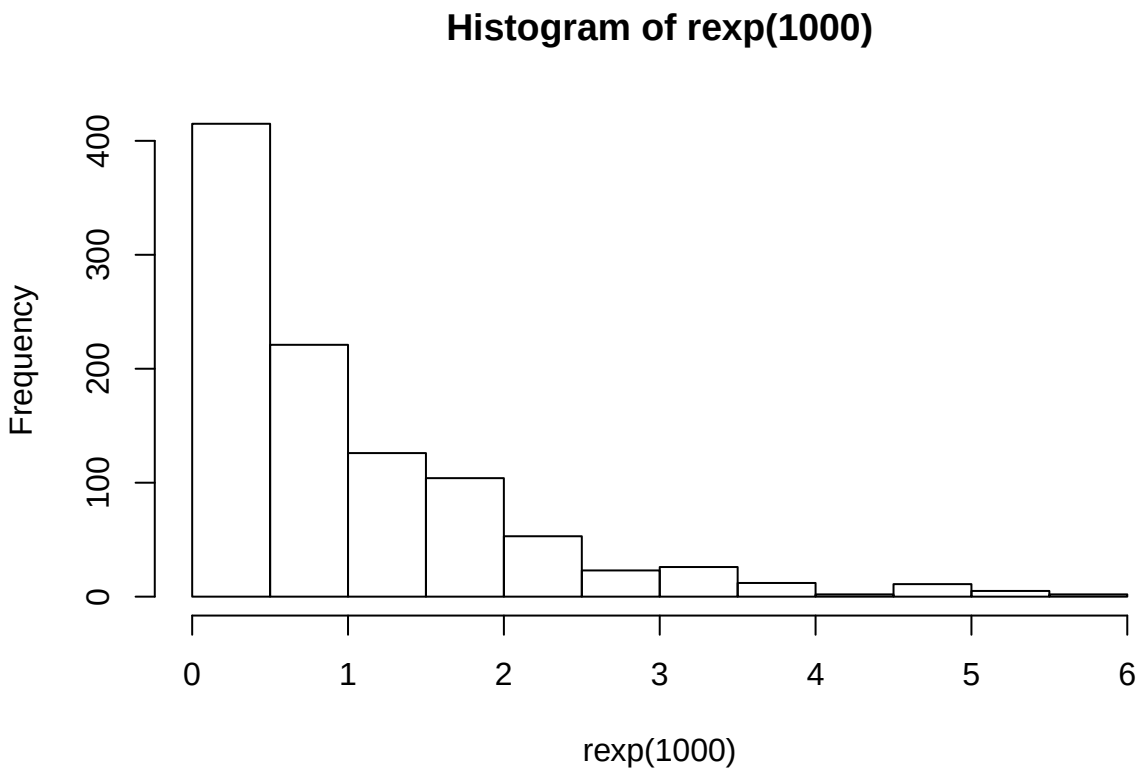

```
# Actual
genCrossTeamWinningProbabilities(
 acceptReject(4, f, 0, 20, 0.5)
```

```
)
```

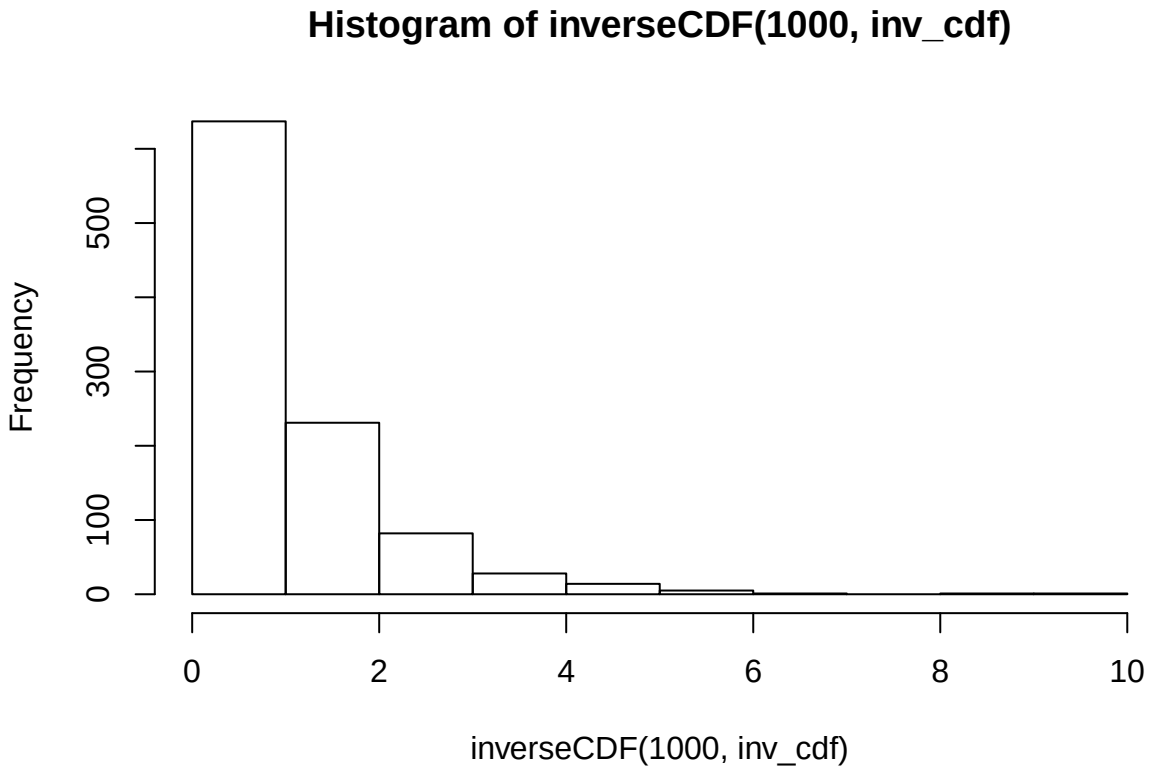
```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.5371378 0.6408147 0.6545621
## [2,] 0.4628622 0.5000000 0.6058920 0.6201839
## [3,] 0.3591853 0.3941080 0.5000000 0.5150584
## [4,] 0.3454379 0.3798161 0.4849416 0.5000000
```

Exp(1) team powers using Inverse CDF

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




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

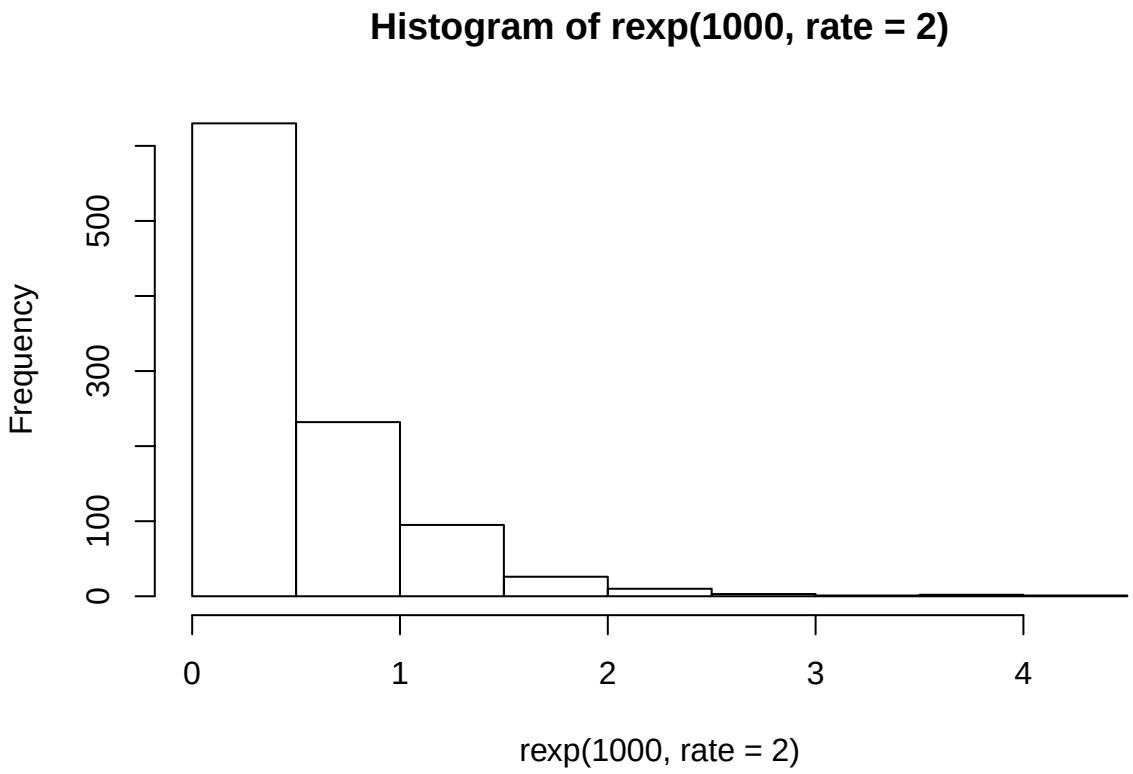



```
# Actual
genCrossTeamWinningProbabilities(
 inverseCDF(
    4,
    inv_cdf
  )
)
```

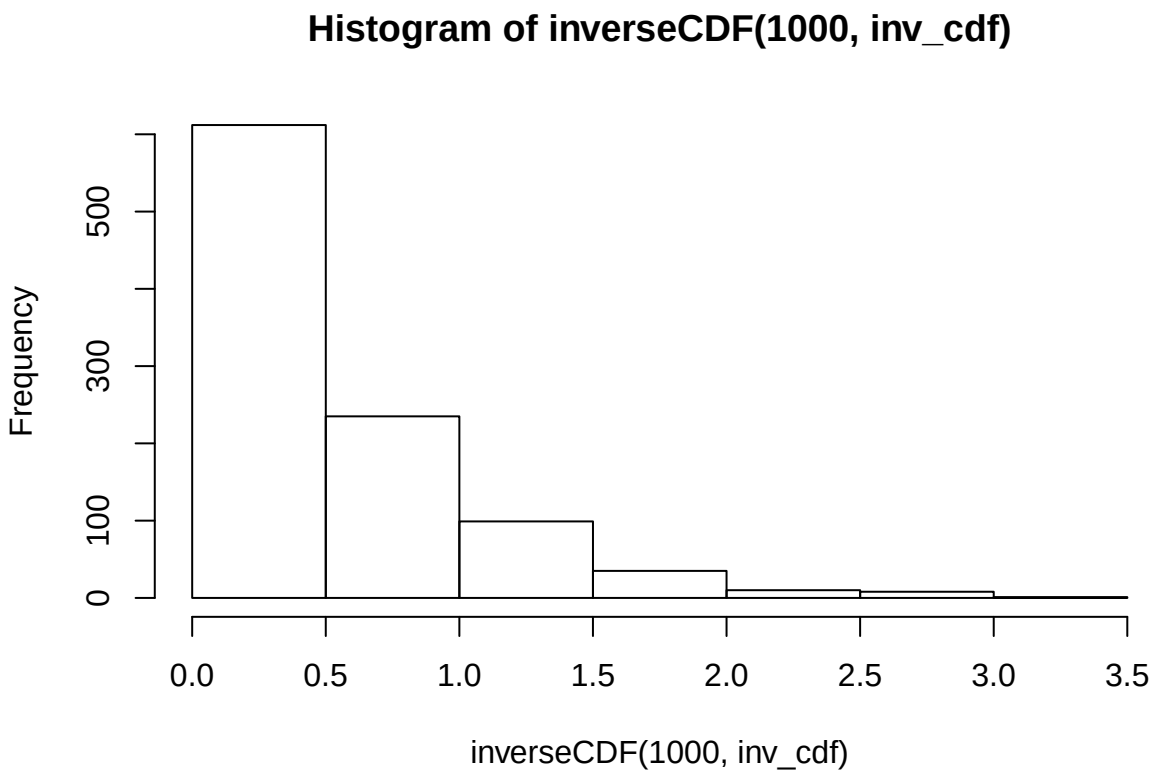
```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.6849519 0.7334374 0.8071586
## [2,] 0.3150481 0.5000000 0.5586068 0.6581428
## [3,] 0.2665626 0.4413932 0.5000000 0.6033681
## [4,] 0.1928414 0.3418572 0.3966319 0.5000000
```

Exp(2) team powers using Inverse CDF

```
# Test `inverseCDF`
inv_cdf <- function(x) {return(-log(x) / 2)}
hist(rexp(1000, rate=2))
```




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

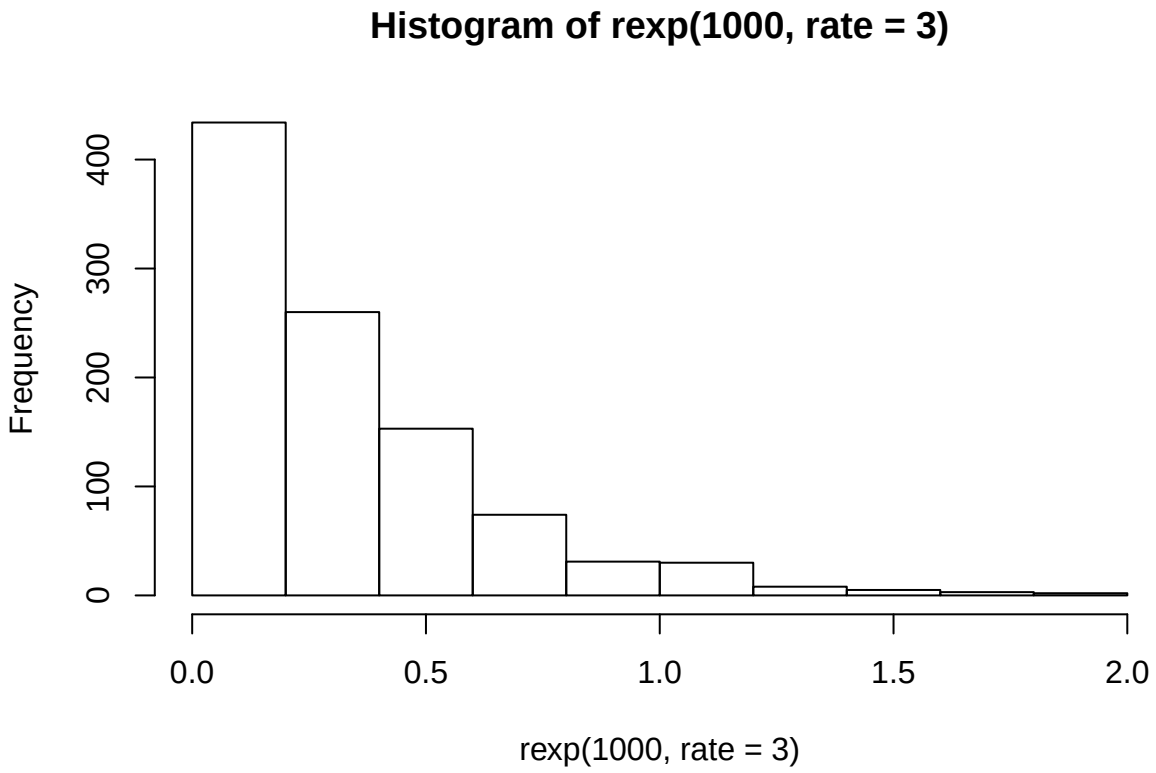


```
# Actual
genCrossTeamWinningProbabilities(
 inverseCDF(
    4,
    inv_cdf
  )
)
```

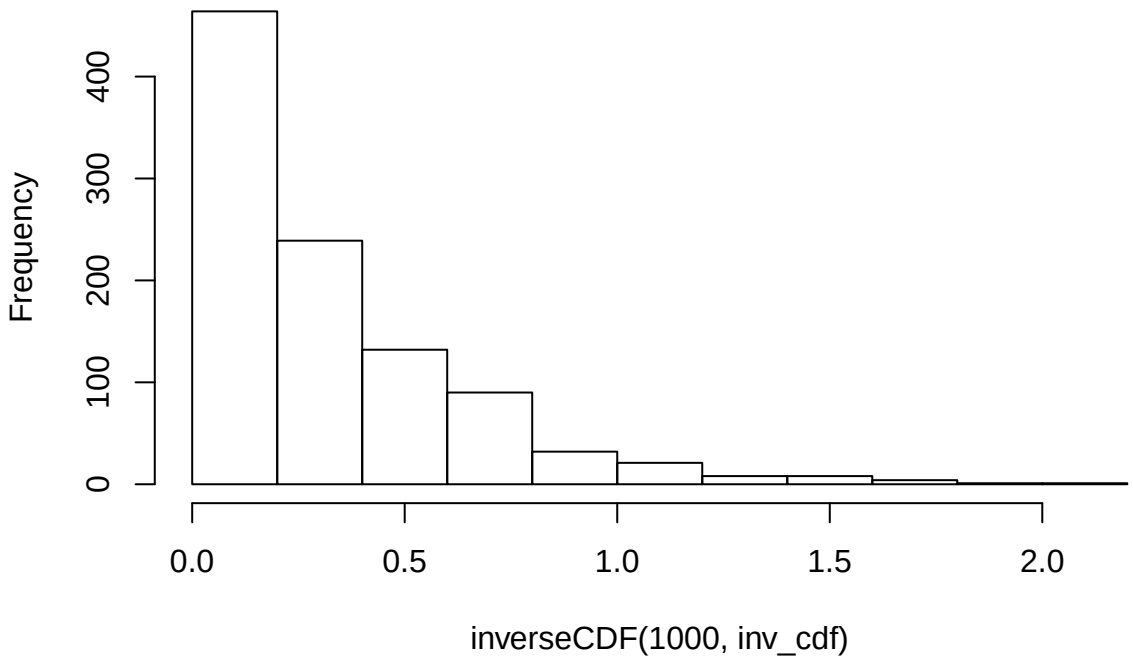
```
##            [,1]       [,2]      [,3]      [,4]
## [1,] 0.50000000 0.67724491 0.7636099 0.9749258
## [2,] 0.32275509 0.50000000 0.6062161 0.9487962
## [3,] 0.23639011 0.39378386 0.5000000 0.9232925
## [4,] 0.02507424 0.05120377 0.0767075 0.5000000
```

Exp(3) team powers using Inverse CDF

```
# Test `inverseCDF`
inv_cdf <- function(x) {return(-log(x) / 3)}
hist(rexp(1000, rate=3))
```




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




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

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.6651496 0.6811446 0.8629892
## [2,] 0.3348504 0.5000000 0.5181692 0.7602434
## [3,] 0.3188554 0.4818308 0.5000000 0.7467410
## [4,] 0.1370108 0.2397566 0.2532590 0.5000000
```