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] 2 7
```

Examples

Normal(0, 1) team powers

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

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.5561499 0.6653675 0.7019442
## [2,] 0.4438501 0.5000000 0.6134306 0.6527212
## [3,] 0.3346325 0.3865694 0.5000000 0.5422158
## [4,] 0.2980558 0.3472788 0.4577842 0.5000000
```

Normal(10, 0.01) team powers

Note the probabilities are closer to 0.5

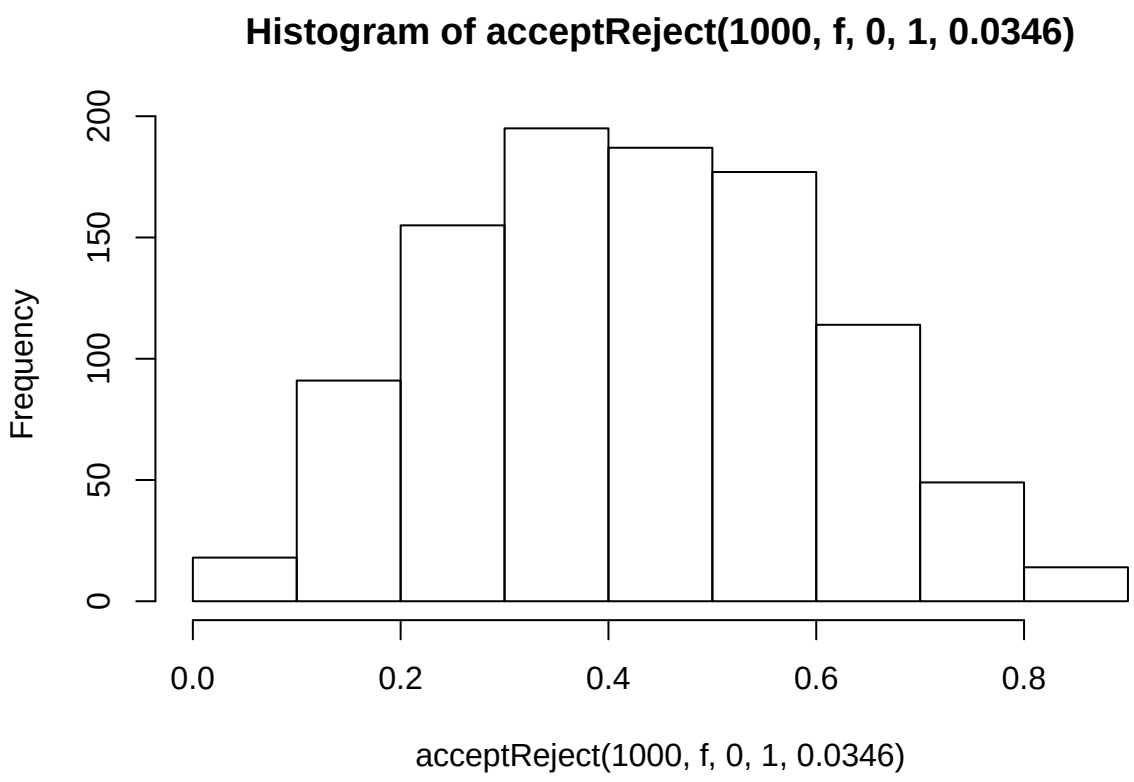
```
genCrossTeamWinningProbabilities(
 genNormalPowers(4, mean=10, sd=0.1)
)
```

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.5008073 0.5019978 0.5077414
## [2,] 0.4991927 0.5000000 0.5011905 0.5069342
## [3,] 0.4980022 0.4988095 0.5000000 0.5057439
## [4,] 0.4922586 0.4930658 0.4942561 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))
```

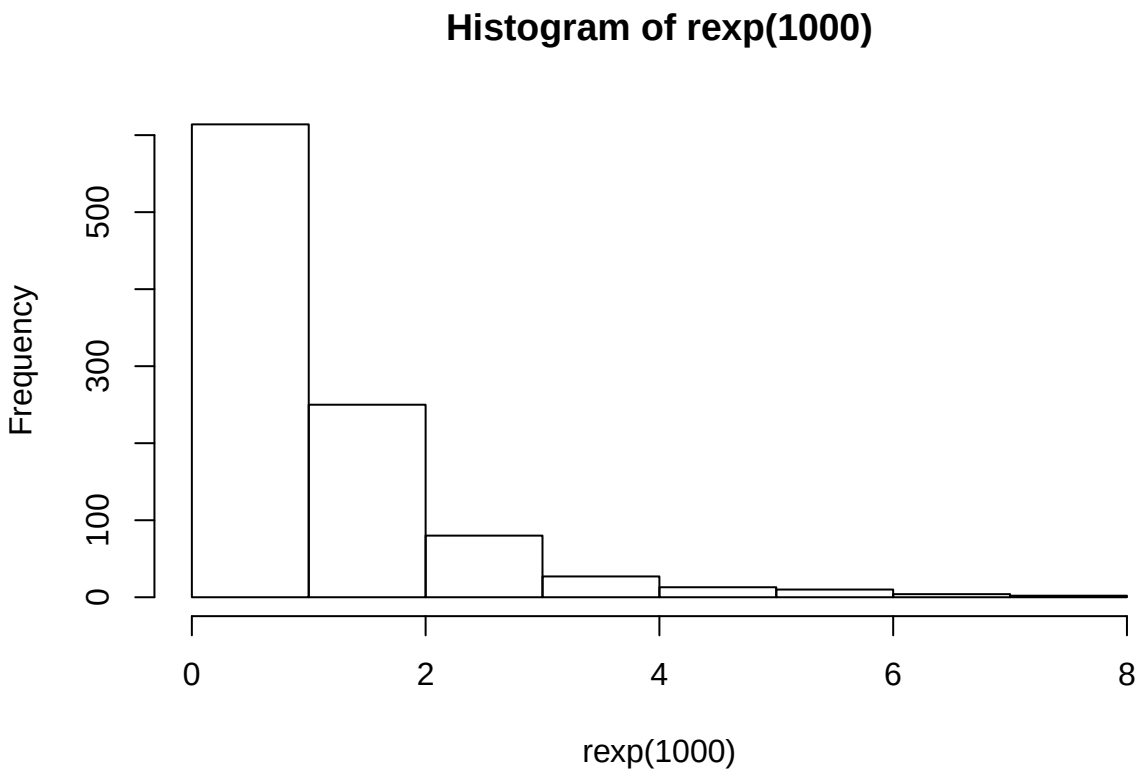



```
# Actual
genCrossTeamWinningProbabilities(
 acceptReject(4, f, 0, 1, 0.0346)
)
```

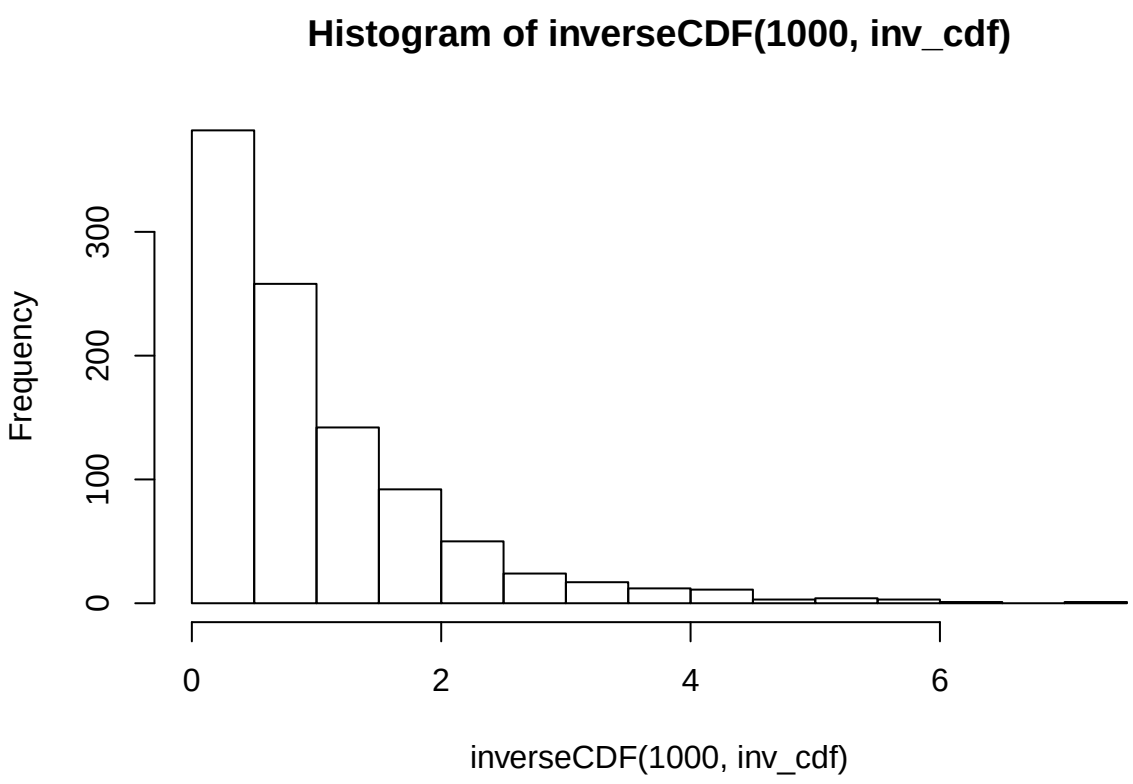
```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5000000 0.5016359 0.6512383 0.8607525
## [2,] 0.4983641 0.5000000 0.6497506 0.8599664
## [3,] 0.3487617 0.3502494 0.5000000 0.7680027
## [4,] 0.1392475 0.1400336 0.2319973 0.5000000
```

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 0.6765656 0.7463379 0.8438205
## [2,] 0.3234344 0.5000000 0.5844680 0.7208940
## [3,] 0.2536621 0.4155320 0.5000000 0.6474292
## [4,] 0.1561795 0.2791060 0.3525708 0.5000000
```