Riddler Classic 3-27-20

Austin Lesh

3/30/2020

The puzzle can be found here: https://fivethirtyeight.com/features/can-you-get-the-gloves-out-of-the-box/

```r
library(ggplot2)
library(fitdistrplus)
```

First, let's create a function `trial` with parameter `n` that simulates one trial with an n-sided die. The die is rolled, then the results are added to a new die, and so on until the die has the same number on every face. It will return the number of times the die is rolled.

```r
trial <- function(n){
  die <- 1:n
  roll <- sample(die, size = n, replace = TRUE)
  count <- 1

  # Take advantage of the fact that the variance of a constant is zero.
  # There are other ways to condition the loop, such as while(all(x == x[1])),
  # but this one seems like the most fun
  while (var(roll) != 0){
    die <- roll
    roll <- sample(die, size = n, replace = TRUE)
    count <- count + 1
  }
  return(count)
}
```

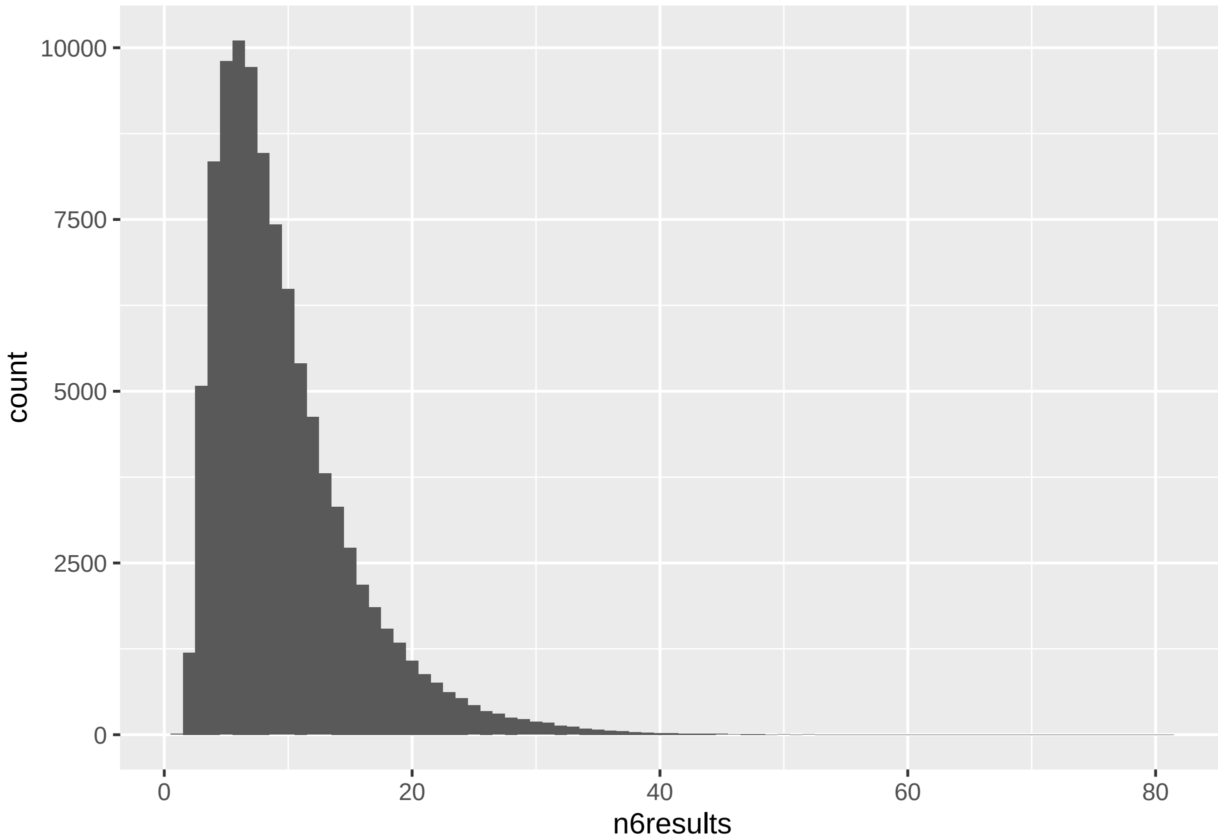
Let's dive a little deeper into a 6-sided die, since it's the most commonly used. We can run an experiment with 100,000 replicates to get a pretty good idea of how the average number of rolls converges.

```r
n6results <- replicate(100000, trial(6))
mean(n6results)
```

```
## [1] 9.642
```

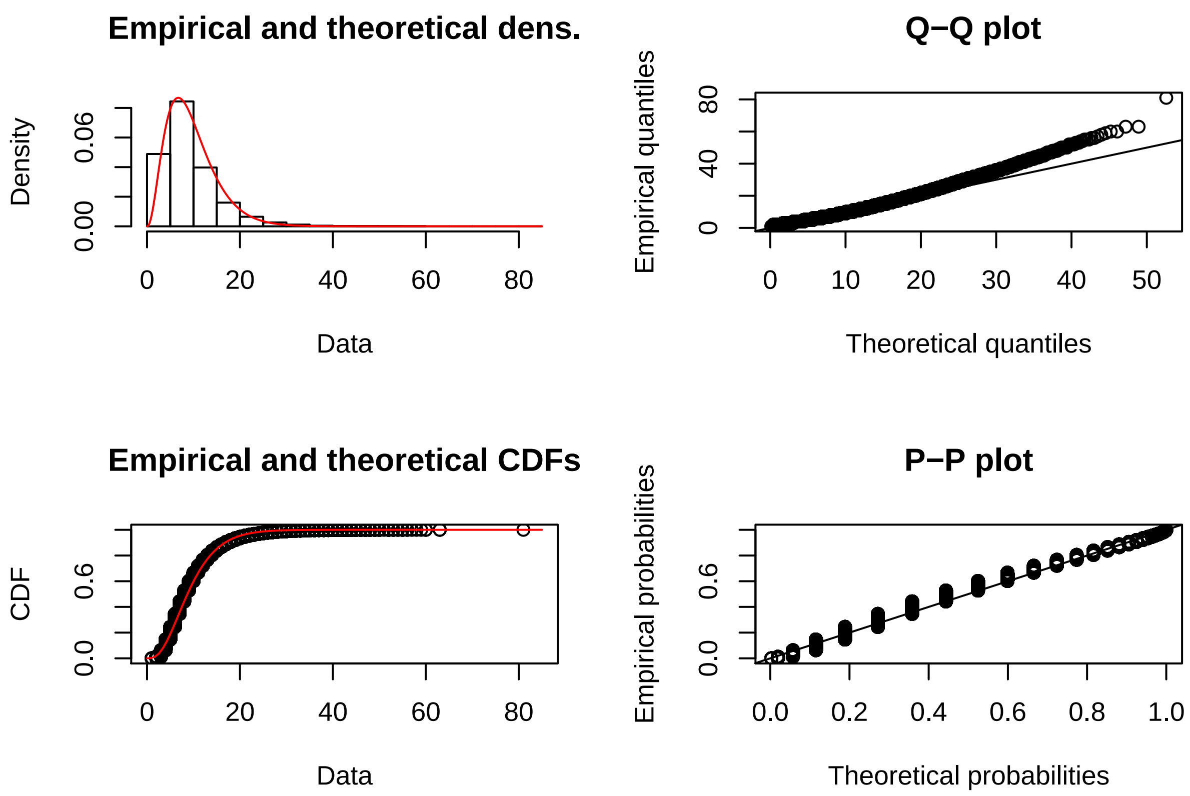
We see that the average number of rolls is 9.64. What does the distribution of `n6results` look like? We can inspect it via histogram:

```r
ggplot() + aes(x = n6results) + geom_histogram(binwidth = 1)
```



Interesting! The distribution of rolls kind of looks like a gamma distribution. Let's try to fit it:

```
gamma_fit <- fitdist(n6results, distr = "gamma", method = "mle")
plot(gamma_fit)
```



Not perfect, but not terrible either.

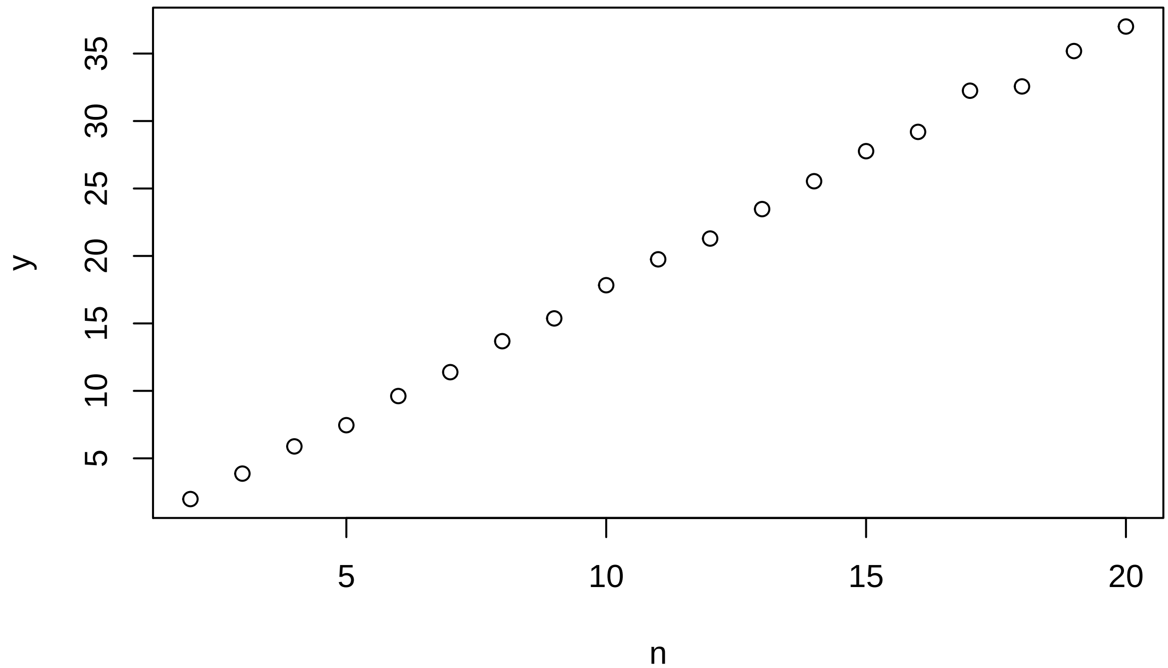
Let's look at different values of `n` and see if there's a function that maps `n` to the number of rolls needed to achieve equilibrium. I'll go up to n = 20, to satisfy my inner Dungeons and Dragons nerd. Also note that the case where n = 1 is trivial. And, we'll create a custom replicating function to perform multiple trials with `sapply` easily.

```r
n <- 2:20

rep <- function(n, x) {
  return(mean(replicate(x, trial(n))))
}

y <- sapply(n, rep, x = 1000)

# To see the results together:
df <- data.frame(n = n, y = y)
df
```

```
##     n      y
## 1   2  1.980
## 2   3  3.868
## 3   4  5.885
## 4   5  7.456
## 5   6  9.616
## 6   7 11.386
## 7   8 13.681
## 8   9 15.371
```

```
## 9  10 17.831
## 10 11 19.750
## 11 12 21.290
## 12 13 23.473
## 13 14 25.539
## 14 15 27.766
## 15 16 29.196
## 16 17 32.248
## 17 18 32.565
## 18 19 35.187
## 19 20 37.005
```

```r
plot(y ~ n)
```



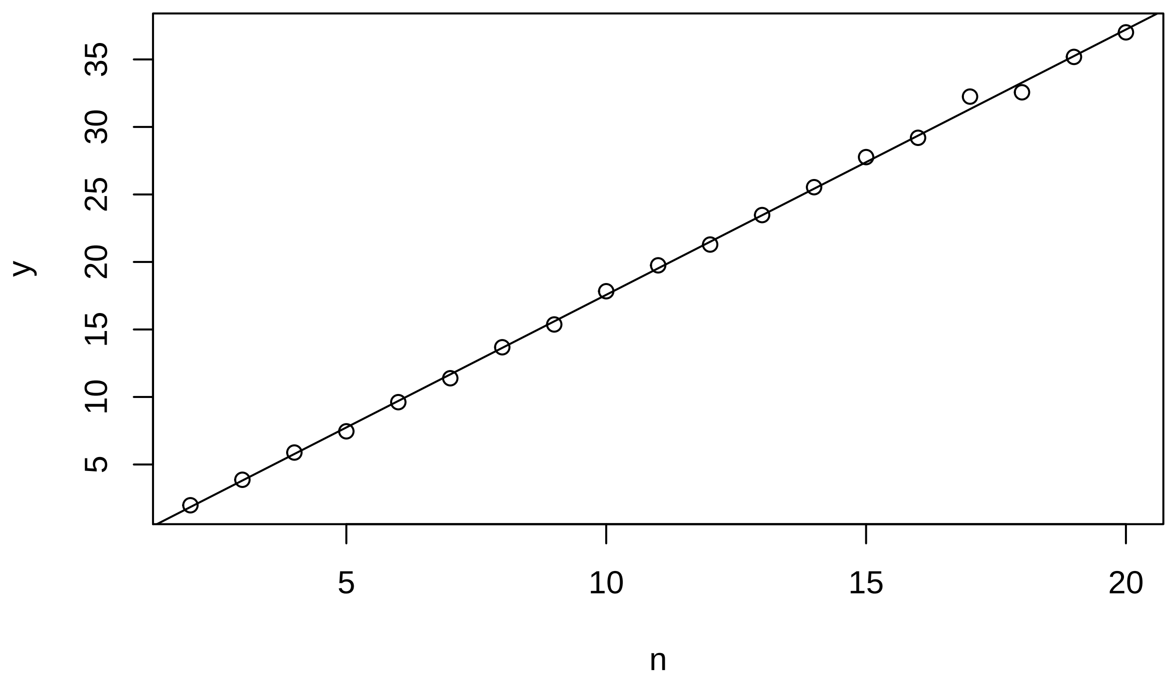
Maybe this is revealing my naivety, but I was not expecting this to be linear! We can now use simple linear regression to easily find the final form of the function.

```r
mod1 <- lm(y ~ n)
summary(mod1)
```

```
##
## Call:
## lm(formula = y ~ n)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.71993 -0.20780  0.01215  0.12274  0.92788
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.08178    0.17769  -11.72 1.45e-09 ***
## n            1.96482    0.01446  135.88  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3452 on 17 degrees of freedom
## Multiple R-squared:  0.9991, Adjusted R-squared:  0.999
## F-statistic: 1.846e+04 on 1 and 17 DF,  p-value: < 2.2e-16
```

```r
plot(y ~ n)
abline(mod1)
```



Pretty good! Doing a little bit of rounding, and letting A = the average number of rolls, we get A = 1.96n - 2.08 for n $\in \{2, 3, ..., 20\}$.