Beta Distribution

Aaron Mulvey

9/27/2022

R Markdown

The Beta distribution Beta(a, b) has the following Probability Density Function (PDF):

$$f(x) = cx^{\alpha-1}(1-x)^{\beta-1}$$

$$0 < x < 1,$$

with parameters a > 0 and b > 0, and a constant c chosen to make it a valid PDF. Let X Beta(a, b) with parameters a $= 6$ and b $= 2$.

1. Find c for the given parameters

This is the beta fuction:

$$f(x;\alpha;\beta) = cx^{\alpha-1}(1-x)^{\beta-1}$$

c makes the function valid so the integral in the support must be $= 1$ so c is equal to:

$$c\int_0^1 f(x;\alpha;\beta)dx = 1 \implies c = \frac{1}{\int_0^1 f(x;\alpha;\beta)dx}$$

so that means that:

$$c = \frac{1}{B(\alpha,\beta)}$$

I substitute in my formula and I get the following:

$$f(x;\alpha;\beta) = \frac{1}{B(\alpha,\beta)}x^{\alpha-1}(1-x)^{\beta-1}$$

I saw that I can also do it using the gamma distribution, looks like this:

$$f(x;\alpha;\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 \omega^{\alpha-1}(1-\omega)^{\beta-1}d\omega}$$

$$= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}$$

```r
a <- 6
b <- 2

constant_with_BETA <- function(a, b){
  constant <-  1/beta(a,b)
  return (constant)
}

constant_with_GAMMA <- function(a, b){
  constant <-  gamma(a + b)/(gamma(a) * gamma(b))
  return (constant)
}
constant_with_GAMMA(a,b)
```

```
## [1] 42
```

```r
constant_with_BETA(a,b)
```

```
## [1] 42
```

2. Find P $(0.2 < X < 0.8)$

$$P(\alpha < X < \beta) = P(\alpha < X \leq \beta) = P(\alpha \leq X < \beta) = P(\alpha \leq X \leq \beta)$$

This is because:

$$P(X = \alpha) = 0$$

Because:

$$\int_{\alpha}^{\alpha} f(x)dx = 0$$

Also using the naive definition of probability:

$$P(X = \alpha) = \frac{\alpha}{\Omega}$$

Since we are working with continous random variables the sample set is infinite so we have that:

$$P(X = \alpha) = \frac{\alpha}{\infty} = 0$$

```r
integrate(dbeta, lower=0.2, upper=0.8, a, b)
```

```
## 0.5763456 with absolute error < 6.4e-15
```

3.1 Find E(X)

formula of expectation:

$$\mu = E[X] = \int_{0}^{1} x f(x; \alpha; \beta)dx$$

I expand it.

$$= \int_0^1 x \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} dx$$

And I get the following:

$$= \frac{\alpha}{\alpha+\beta}$$

Take comun divisor alpha:

$$= \frac{1}{1+\frac{\beta}{\alpha}}$$

```
Expected_value_X <- function(a,b){
  return (1/(1 + (b/a)))
}

Expected_value_X(a,b)
```

```
## [1] 0.75
```

3.2 Find Var(X)

$$var(X) = E[(X-\mu)^2] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

```
var_X <- function(a, b){
  return ((a * b) / ((a + b)^2 * (a + b + 1)))
}

var_X(a,b)
```

```
## [1] 0.02083333
```

Let X be a random variable with PDF f , an example PDF is depicted in Figure 1. Here is what you know:

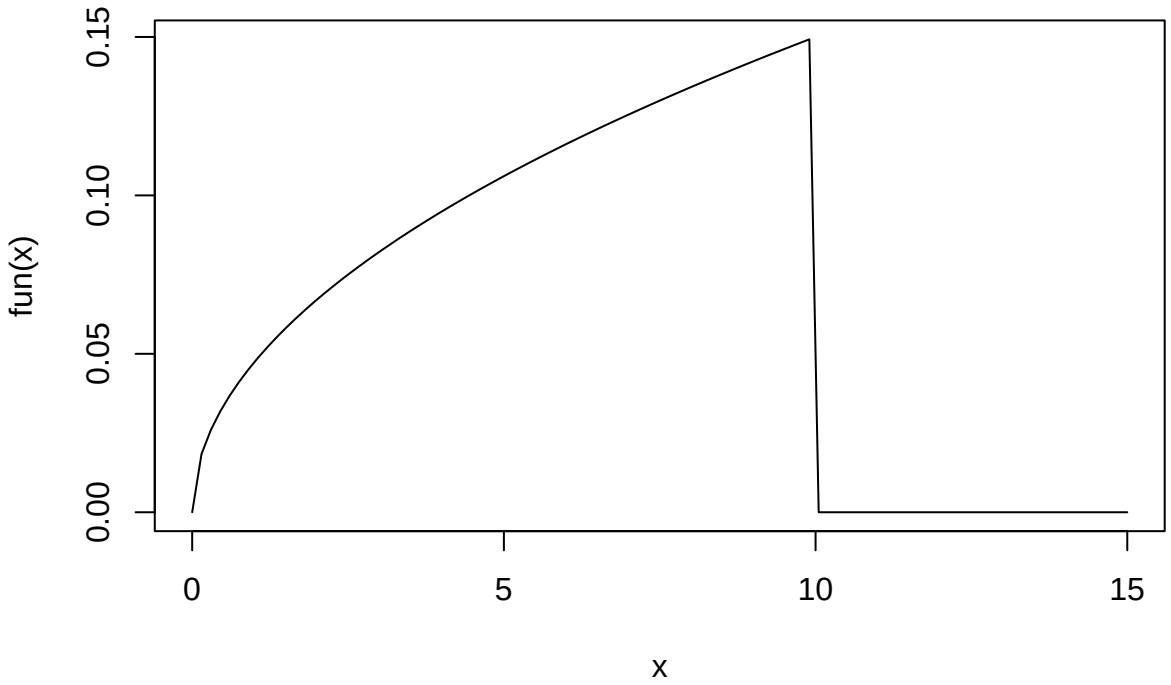
You can evaluate f for specific values x You do not know the Cumulative Distribution Function (CDF) of f You have a random number generator that can draw from the uniform distribution U Unif(a, b), for any a and b You can find the absolute maximum of f The support of f is between 0 and 10

(a) Create your own version of Figure 1 (it doesn't have to be exactly the same function). Implement (in code) a simulation experiment to compute the following probability for your own PDF f, P(3 X < 6).

First, I will create my function:

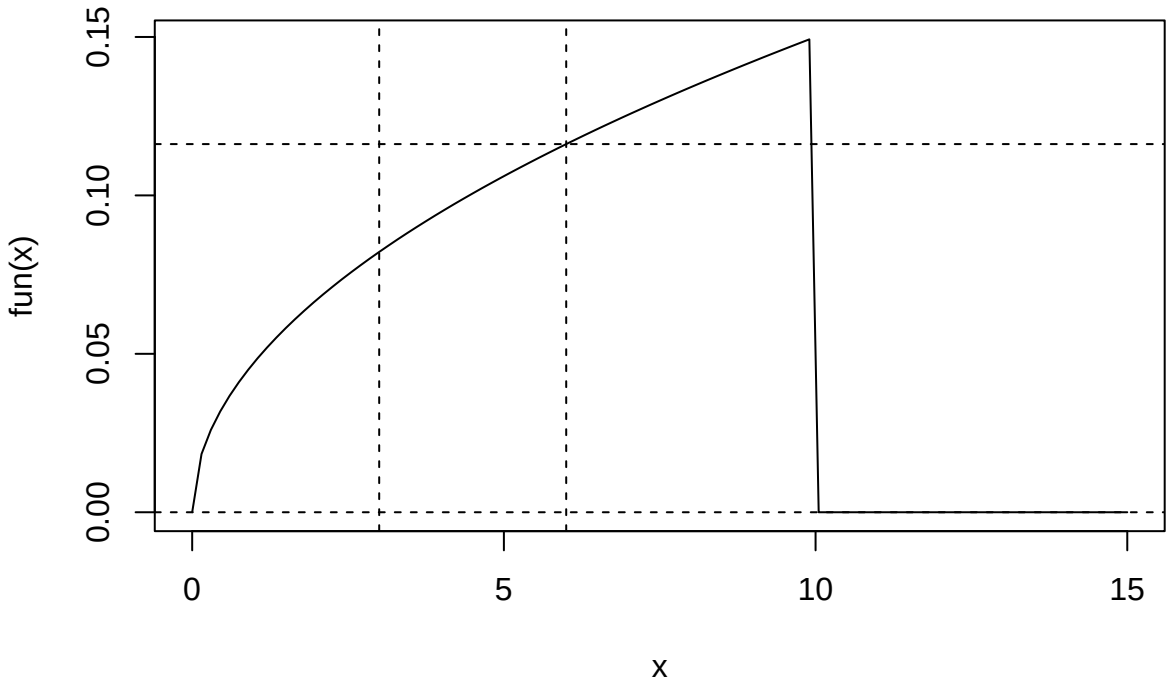
$$f(x) = \frac{\sqrt{x}}{21.08185}$$

```
suport <- 0
SUPORT <- 10
a <- 3
b <- 6

fun <- function(x) ifelse((x > 10),0, (sqrt(x)/21.08185))

curve(fun, 0, 15)
```



Then I will take the maximun point in the interval given, I did this because I can calculate the area of the rectangle knowing the interval (a, b) and the maximun height of my function.

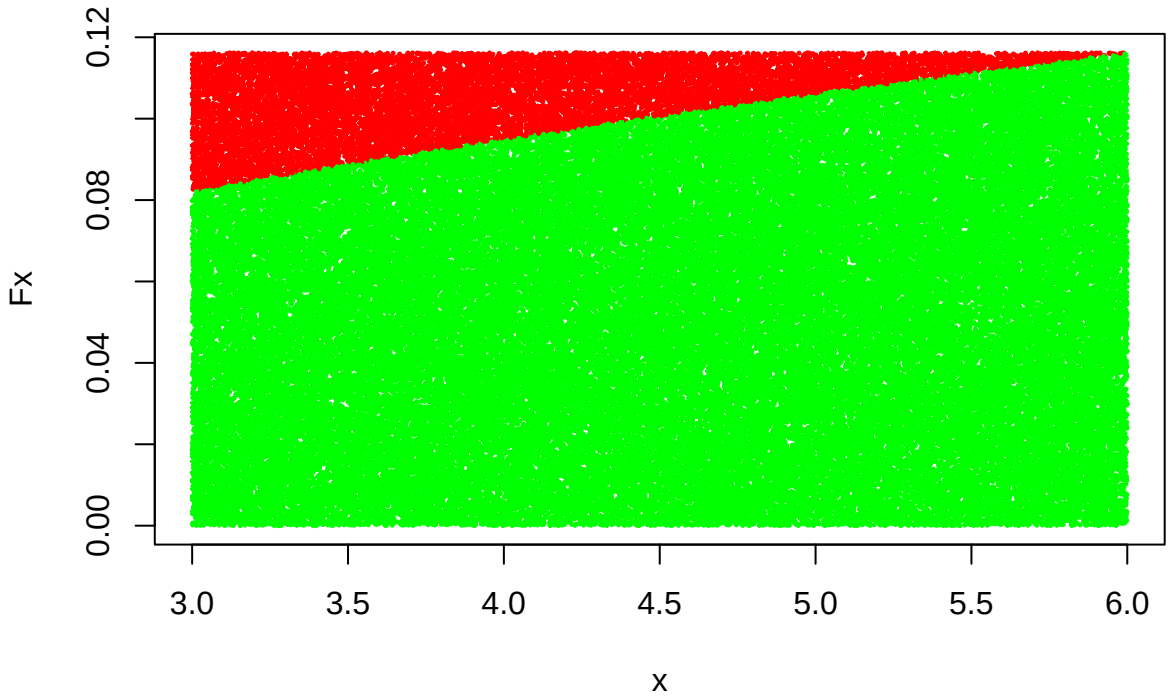
```
FIND_MAXIMUN <- function(fun, a, b, n){
  maximun = fun(a)
  secuencia <- seq(a, b, by = n)
  for (i in secuencia) {
      maximun <- ifelse(fun(i) > maximun, maximun <- fun(i), maximun <- maximun)
  }
  return (maximun)
}

curve(fun, 0, 15)
abline(h = 0, lty = 2)
abline(v = c(3, 6), lty = 2)
abline(h = FIND_MAXIMUN(fun, a , b, 0.01), lty = 2)
```



What I am going to do now is to throw random points in that square and I will store the poits that are below my function whith that I will calculate the ratio and I will multiply the area of the rectangle with that ratio so that I will get the integral in the given point.

```
RANDOM_POINTS <- function(fun, a, b, n, visualice){


  x = c()
  Fx = c()
  color = c()
  true <- 0
  false <- 0

  maximun <- FIND_MAXIMUN(fun,a,b,0.01)

  for (i in 1:n) {

  x[i] = runif(1, a, b)
  Fx[i] = runif(1, 0, maximun)
  color[i] = ifelse(Fx[i] > fun(x[i]), TRUE, FALSE)
  ifelse(color[i] == T, true <- true + 1, false <-  false + 1)

  }


  if(visualice == T){
    plot(x, Fx, col = ifelse(color == TRUE, 'red', 'green'), pch = 19, cex = 0.2)
  }else{
    return (((b-a) * maximun) * (false/n))
  }


}
```

```
RANDOM_POINTS(fun, a, b, 100000, T)
```



Now I will check if the result of the integral is correct, whith the method that R provide us.

```
integrate(fun, lower = a, upper = b)
```

```
## 0.3004412 with absolute error < 3.3e-15
```

```
RANDOM_POINTS(fun, a, b, 10000, F)
```

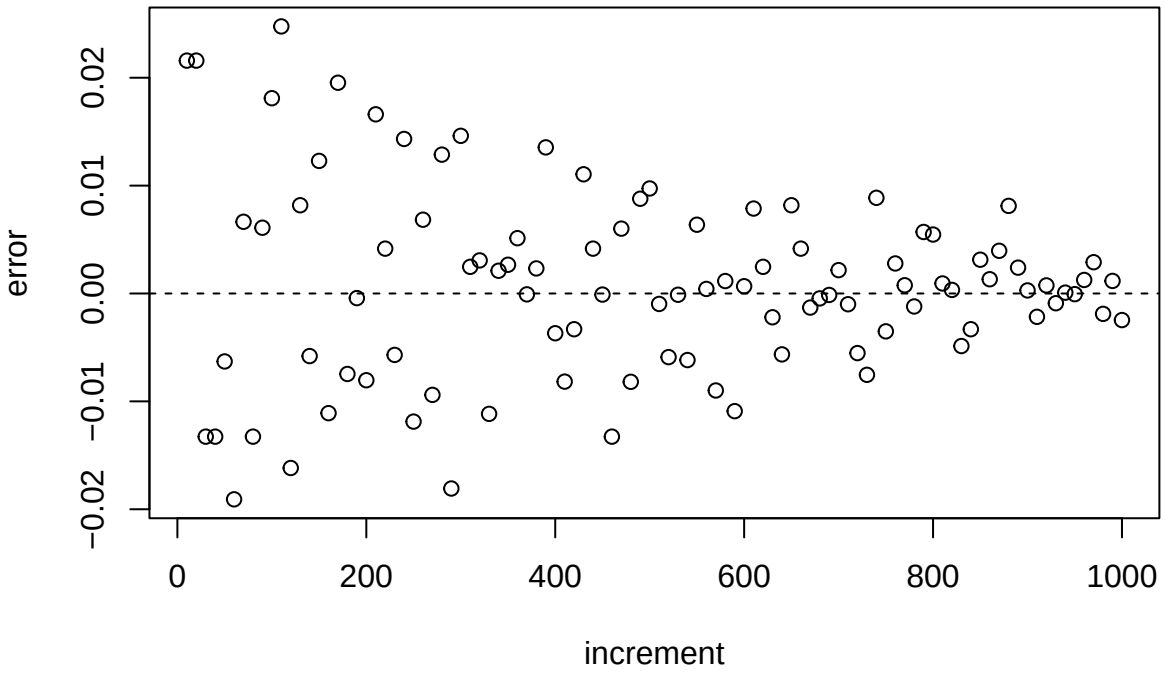
```
## [1] 0.3007101
```

(b) What components of your simulation have impact on approximation accuracy?

I will do an experiment increasing the number of points that I input to my simulation, whit this I will plot in a graph for see how the number of points affects my approximation, for calculating the error I will use the value that the native function of R provide us.

```
n <- 10
N <- 1000
delta <- 10
error <- c()

EXPERIMENT <- function(n, N, delta, integral, fun, a, b){

  increment <- seq(n, N, by = delta)
    for (i in 1:length(increment)) {
      aproximation <- RANDOM_POINTS(fun, a, b, increment[i], F)
      error[i] <- (integral -  aproximation)
```

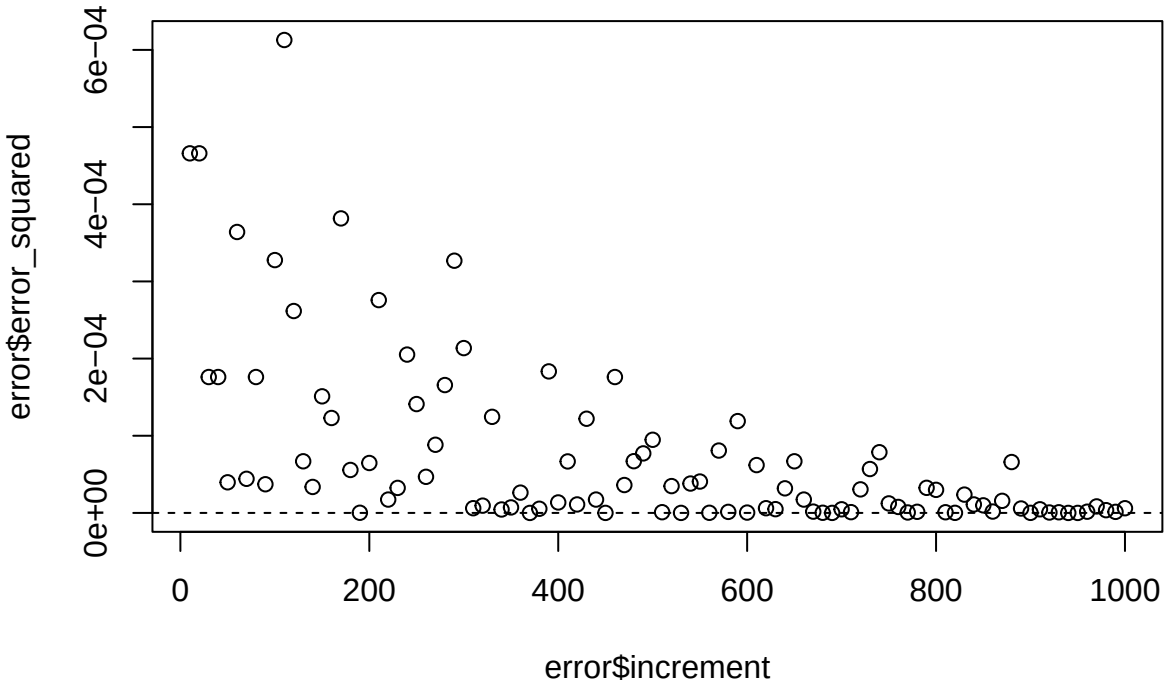
```
    }
  plot(increment, error)
  abline(h = 0, lty = 2)
  return (data.frame(error, increment))
}

integral <- integrate(fun, lower = a, upper = b)

integral <- integral$value

error <- EXPERIMENT(n, N, delta, integral, fun, a , b)
```



As we can see as the number of points increments the error goes decreasing, so the number of points is the component that affects my simulation acuracy.

Same thing happends when I use the squared error:

```
error <- error %>%
  mutate(error_squared = error^2)

plot(x = error$increment, y = error$error_squared)
abline(h = 0, lty = 2)
```



I have not mentioned but the PDF of my function is valid because:

```
integrate(fun, lower = suport, upper = SUPORT)
```

```
## 1 with absolute error < 0.00012
```

And also is bigger than 0