Práctica 8: Ley de los grandes números

Martínez Ostoa Néstor (LCD32)

8 de noviembre del 2020

1. Tiempo de convergencia

1.1 Gráficas Bernoulli

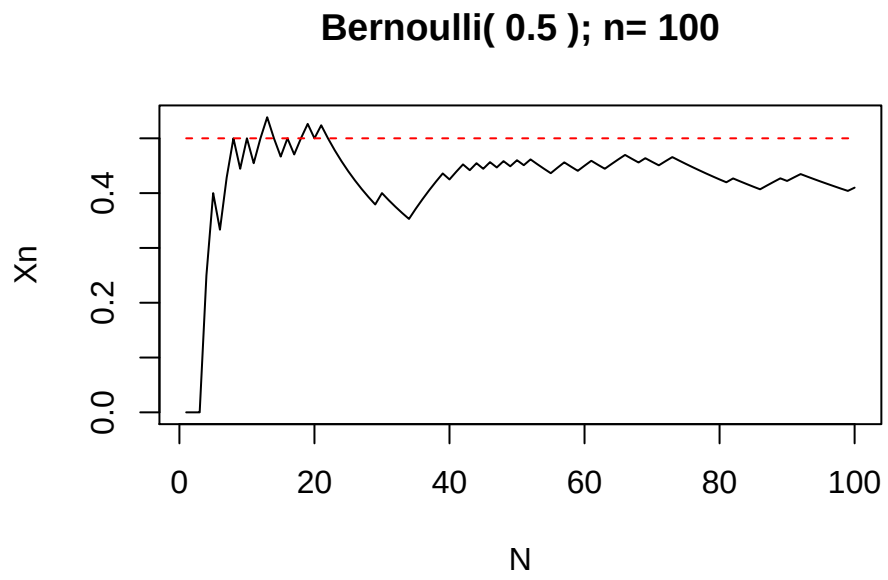
Para esta variable aleatoria tenemos que: $\mu = p$

```
suppressMessages(library(Rlab))
suppressMessages(library(Pareto))

grafica_bernoulli <- function(N, p) {
  x <- 1:N
  y <- cumsum(rbern(N, p)) / x
  plot(x, y, type='l', xlab='N', ylab='Xn', main = paste('Bernoulli(', p, '); n=', N))
  par(new = T)
  lines(x, rep(p, N), col='red', type='l', lty=2)
}
```

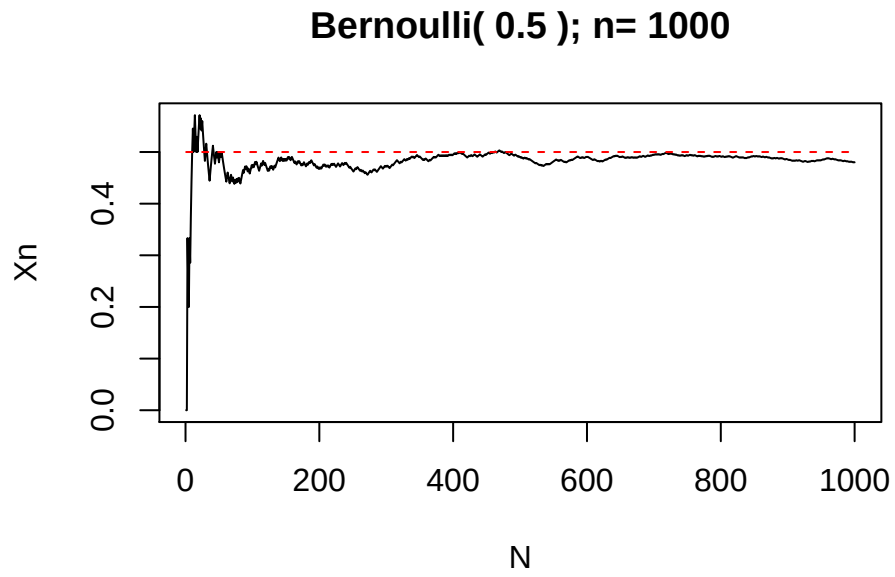
$N = 100$

```
grafica_bernoulli(100, 0.5)
```



$N = 1000$

```
grafica_bernoulli(1000, 0.5)
```



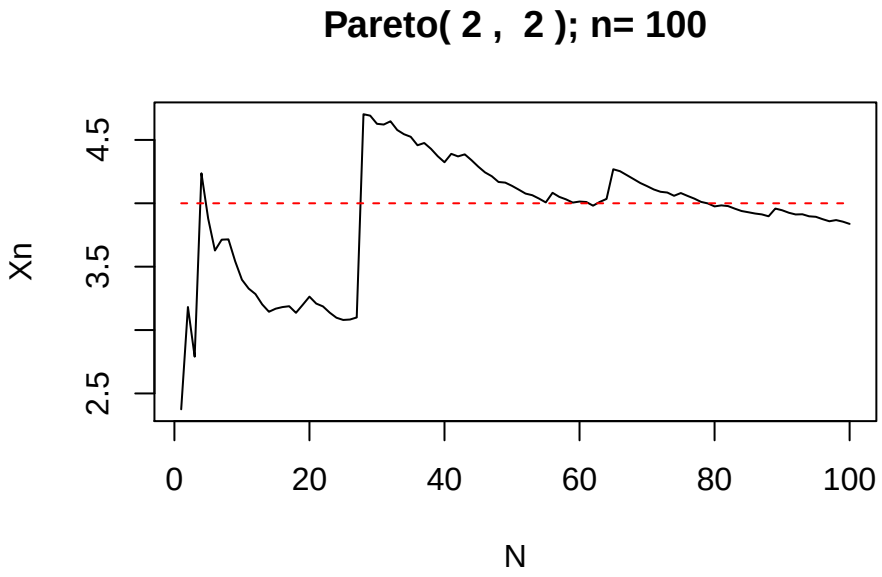
1.2 Gráficas Pareto

Para esta variable aleatoria tenemos que: $\mu = \frac{ab}{a-1}$

```r
grafica_pareto <- function(N, a, b) {
  x <- 1:N
  y <- cumsum(rPareto(N, a, b)) / x
  mu <- (a*b) / (a - 1)
  plot(x, y, type='l', xlab='N', ylab='Xn', main = paste('Pareto(', a, ', ', b, '); n=', N))
  par(new = T)
  lines(x, rep(mu, N), col='red', type='l', lty=2)
}
```

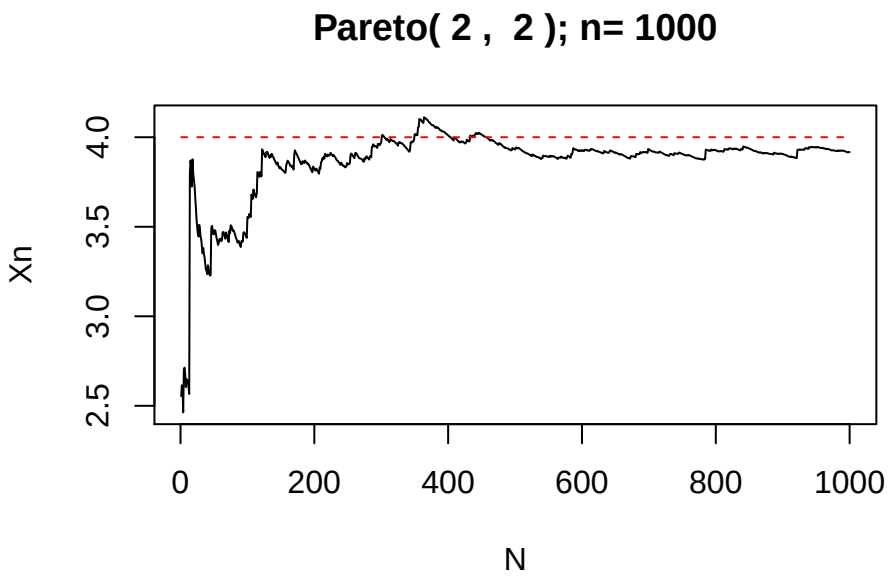
$N = 100$

```r
grafica_pareto(100, 2, 2)
```



$N = 1000$

```r
grafica_pareto(1000, 2, 2)
```



1.3 Errores absolutos

```
promedio <-function(n, f, ...){
  args <-as.list(c(...));
  variables_aleatorias <-do.call(f,c(list(n), args));
  suma <-sum(variables_aleatorias)
  return(suma/n);
}

calcula_n <-function(media, error, n_inicial=1, f, ...){
  n <- n_inicial;
  calculado <- promedio(n, f, ...)
  error_actual <-abs(calculado-media);
  while(error_actual>error){
    n <- n+1;
    calculado <- promedio(n, f, ...)
    error_actual <-abs(calculado-media);
  }
  print(paste('media_calc=', calculado, 'media=', media,
              ' error_actual=', error_actual))
  print(paste('n = ', n))
}
```

1.3.1 Bernoulli

Para esta función de distribución decidí variar las n pues en la gráfcia podemos observar que converge al valor esperado en las primeras iteraciones, después se ven unas fluctuaciones y con n un poco mayores $n > 100$ el valor se estabiliza más.

```r
p <- 0.5
mu <- p
e <- 10 ^ (-5)
calcula_n(mu, e, n_inicial=1, rbern, p)
```

```
## [1] "media_calc= 0.5 media= 0.5  error_actual= 0"
## [1] "n =  2"
```

```r
calcula_n(mu, e, n_inicial=60, rbern, p)
```

```
## [1] "media_calc= 0.5 media= 0.5  error_actual= 0"
## [1] "n =  62"
```

```r
calcula_n(mu, e, n_inicial=100, rbern, p)
```

```
## [1] "media_calc= 0.5 media= 0.5  error_actual= 0"
## [1] "n =  102"
```

1.3.2 Pareto

```r
error <- 10 ^ (-3)
a <- 1.98
b <- 6
media <- b * a / (b - 1)
calcula_n(media, error, n_inicial=1, rPareto, a, b)
```

```
## [1] "media_calc= 2.37685793009009 media= 2.376  error_actual= 0.0008579300900875"
## [1] "n =  172"
```

Convergencia entre Bernoulli vs. Pareto

Pienso que el tiempo de convergencia de la distribución Pareto es más lento que la de Bernoulli debido a que la distribución Pareto es una función cola, lo que implica que sus valores iniciales estarán concentrados en una región del plano Real con valores muy desproporcionados de los valores restantes. Es decir, hay pocos valores que difieren mucho del resto de los valores.

Estimación

Ley de los grandes números en dados

Función que simula tirar 7 dados, cada uno con 6 caras y obtiene como resultado la suma de dichas caras.

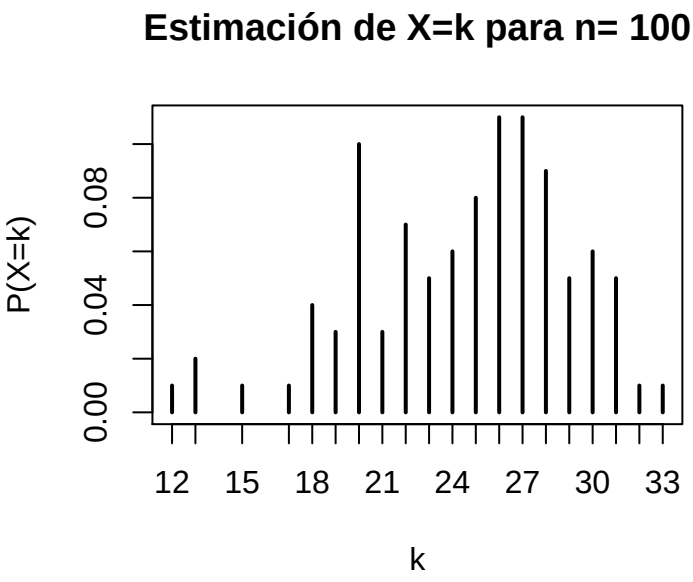
```
simulacion_suma_caras_dados <- function(num_dados = 7, num_caras = 6){
  simulacion_cada_dado <- floor(runif(num_dados) * num_caras + 1)
  suma <- sum(simulacion_cada_dado);
  return(suma);
}
```

Función para graficar la distribución de probabilidad en un intervalo $7 \leq k \leq 42$ donde k es el valor con una probilidad asociada y es el valor que toma la variable aleatoria X.

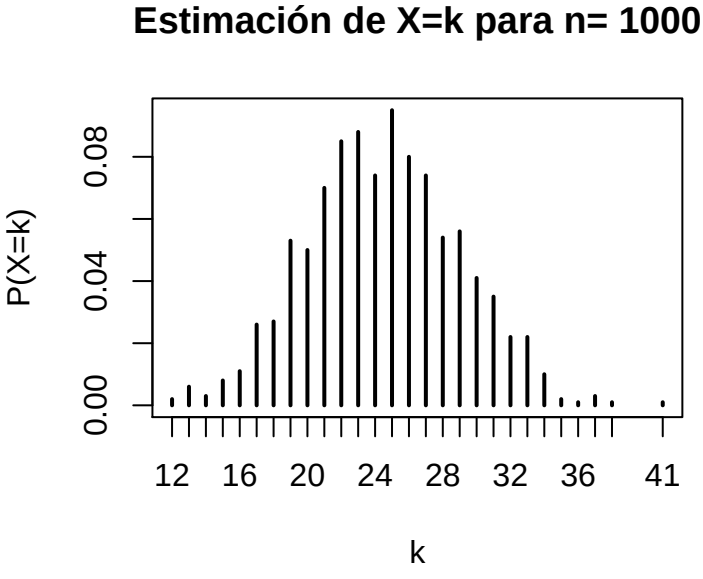
```
verificar_cuantos_hay <- function(n, num_dados = 7, num_caras = 6, graficar = F){
  v <- c()
  dimension <- num_dados * num_caras
  t <- rep(0, dimension)
  for(i in 1:n){
    suma <- simulacion_suma_caras_dados(num_dados, num_caras);
    t[suma] = t[suma] + 1
    v <- c(v, suma);
  }
  cuenta <- table(v) / n;
  if(graficar){
    plot(cuenta, xlab='k', ylab='P(X=k)', main=paste('Estimación de X=k para n=', n));
  }
  total <- t[num_dados : dimension] / n
  return(list("cuenta" = cuenta, "total" = total));
}
```

Y al experimentar con distintos valores de n obtenemos las siguientes gráficas:

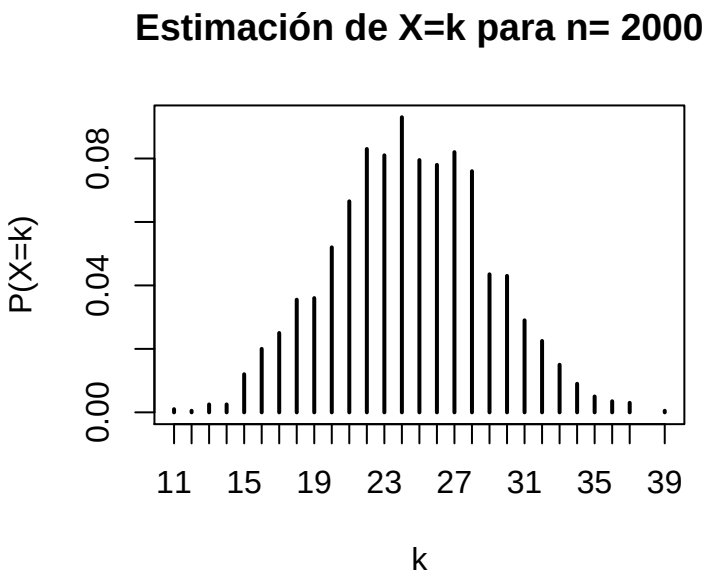
```
resultado <- verificar_cuantos_hay(100, graficar=T)
```




```
resultado <- verificar_cuantos_hay(1000, graficar=T)
```




```
resultado <- verificar_cuantos_hay(2000, graficar=T)
```



Confianza en la aproximación

¿Cómo podemos asegurar que nuestra aproximación es buena si no tenemos un valor esperado al cual llegar? Lo que podemos hacer es comparar los valores para dos instancias diferentes con n diferente. Es decir, calcular un valor para n y después un valor para un n + incremento y comparar ambos valores, si dicha comparación no es muy grande, podemos afirmar que es buena aproximación. El método de comparación está dado por las siguiente ecuación:

$$\text{error}_{calculado} = max\{|n_k - n_{k+\text{incr}}|\}$$

donde **incr** es un incremento arbitrario.

A continuación se muestra la función **error** que reciba como parámetros una e que es el error de acpetación y una n_{inicial} junto con un incremento y lo que hará esta función será calcular un error $\text{e}_{\text{calculado}}$ hasta que sea mayor que el error de aceptación e.

```
error <- function(e, n_inicial = 1000, incremento = 100, verbose = F) {
  n1 <- verificar_cuantos_hay(n_inicial)$total
  n2 <- verificar_cuantos_hay(n_inicial + incremento)$total
  e_calc <- max(abs(n1-n2))
  while (e_calc > e) {
    if (verbose) {
      print(paste('Error calculado=', e_calc, ' n_inicial=', n_inicial,
                  'n_inicial + incr=', n_inicial + incremento))
    }
    n1 <- verificar_cuantos_hay(n_inicial)$total
    n2 <- verificar_cuantos_hay(n_inicial + incremento)$total
    e_calc <- max(abs(n1-n2))
    n_inicial = n_inicial + incremento
  }
  return (e_calc);
}
```

```r
error(e = 0.03, n_inicial = 100, incremento = 30, verbose = T)
```

```
## [1] "Error calculado= 0.0584615384615385  n_inicial= 100 n_inicial + incr= 130"
## [1] "Error calculado= 0.0684615384615385  n_inicial= 130 n_inicial + incr= 160"
## [1] "Error calculado= 0.0615384615384615  n_inicial= 160 n_inicial + incr= 190"
## [1] "Error calculado= 0.0529605263157895  n_inicial= 190 n_inicial + incr= 220"
## [1] "Error calculado= 0.0349282296650718  n_inicial= 220 n_inicial + incr= 250"
## [1] "Error calculado= 0.054  n_inicial= 250 n_inicial + incr= 280"
## [1] "Error calculado= 0.0415714285714286  n_inicial= 280 n_inicial + incr= 310"
## [1] "Error calculado= 0.0397465437788018  n_inicial= 310 n_inicial + incr= 340"
## [1] "Error calculado= 0.0413662239089184  n_inicial= 340 n_inicial + incr= 370"

## [1] 0.02591415
```