**Prática 5 – MS**

**Grupo:**

Andressa Alvilino Ferreira Silva

Matheus Cunha Reis

Weuler Borges

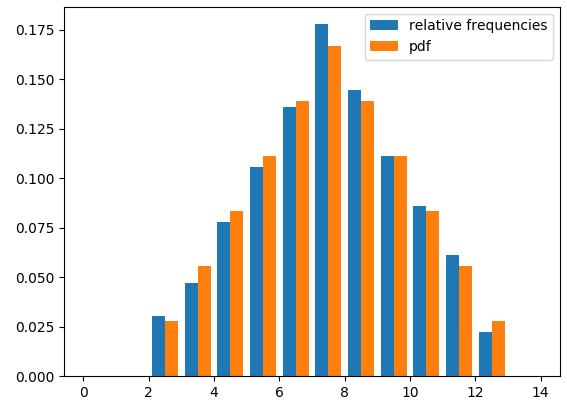
**6.1.1)** (a) Simulate rolling a pair of dice 360 times with five different seeds and generate five histograms of the resulting sum of the two up faces. Compare the histogram mean, standard deviation and relative frequencies with the corresponding population mean, standard deviation and pdf. (b) Repeat for 3600, 36 000, and 360 000 replications. (c) Comment.

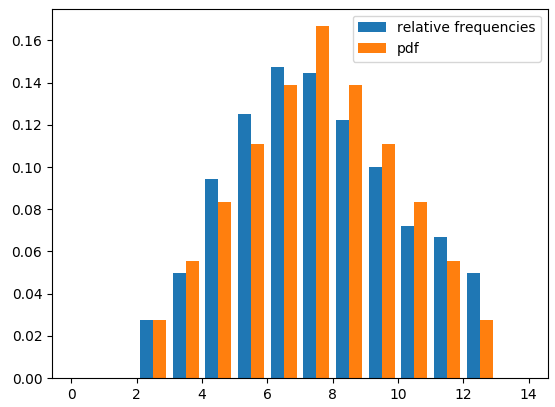
**a)** Códigos utilizados galileo.cpp generateHistogram.py

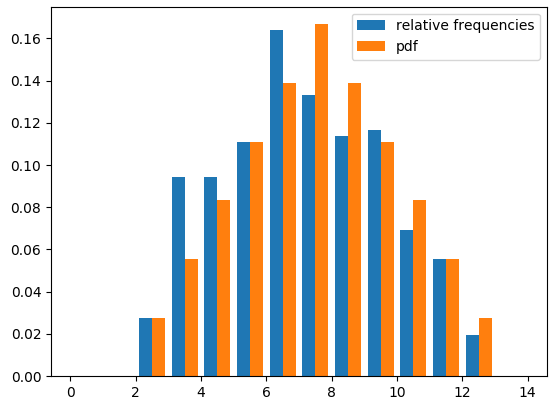
Population Mean

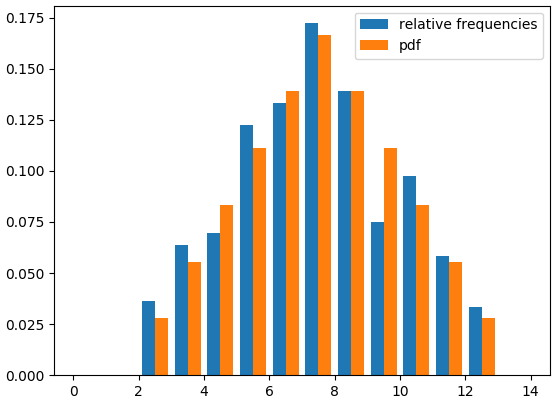
Standard Deviation

Seed = 12345, mean = 7, stdev = 2.38223

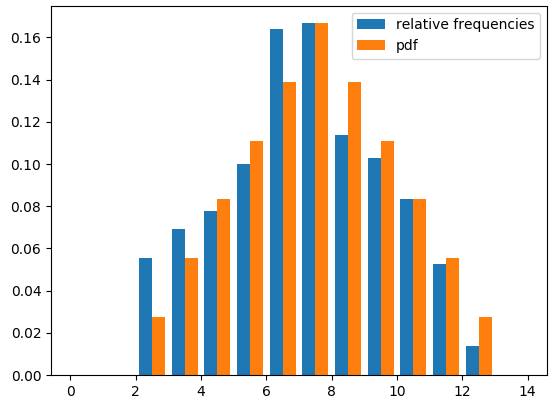


Seed = 124564, mean = 7, stdev = 2.54569

Seed = 242453, mean = 6, stdev = 2.57337

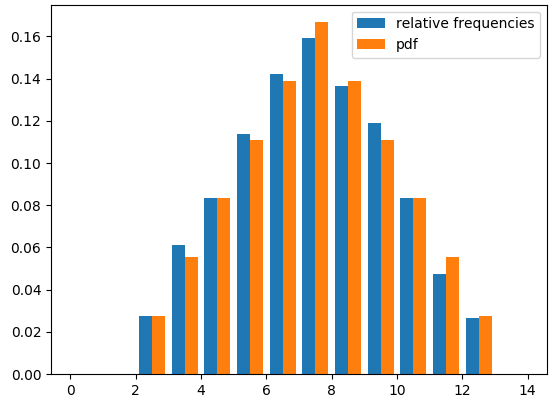
Seed = 3413124, mean = 6, stdev = 2.67758

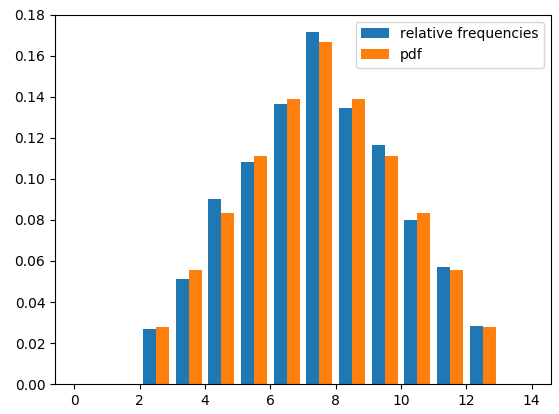
Seed = 98786465, mean = 6, stdev = 2.57391

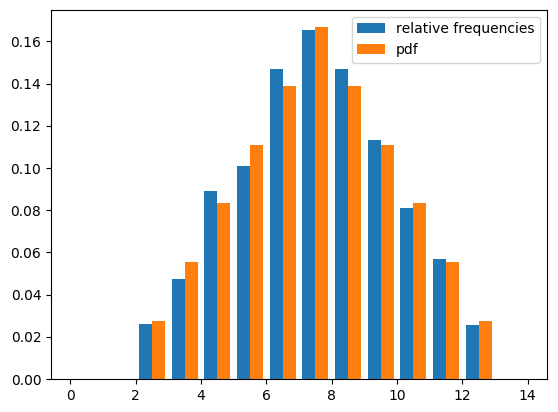


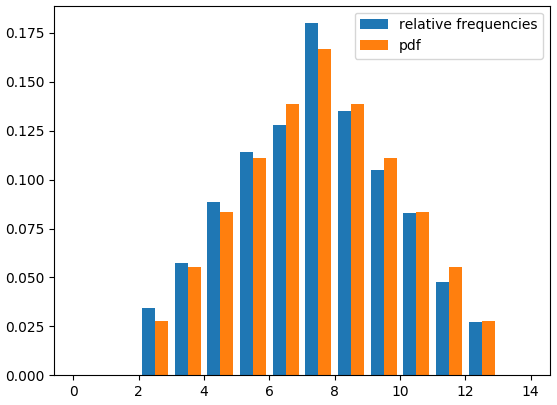
**b)** 3600 times

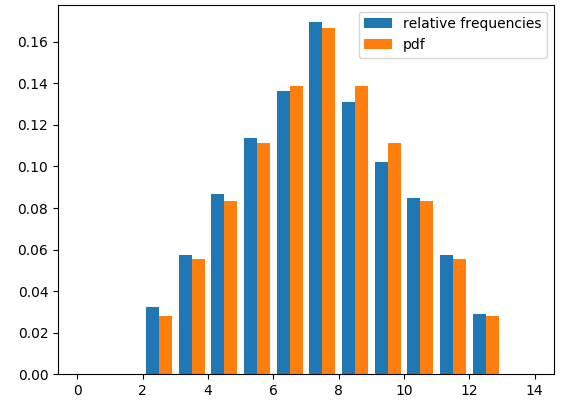
Seed = 12345, Mean = 7, Stddev = 2.37703



Seed = 124564, Mean = 7, Stddev = 2.41068

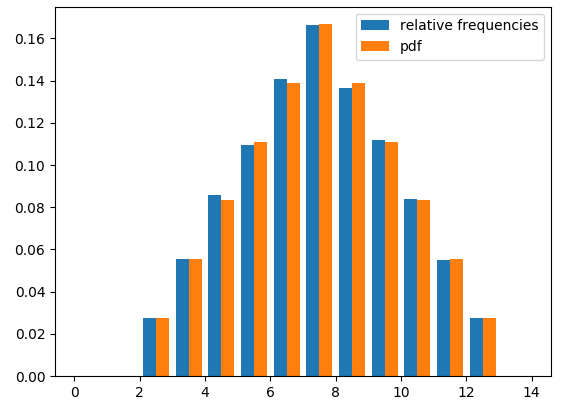
Seed = 242453, Mean = 6, Stddev = 2.58597

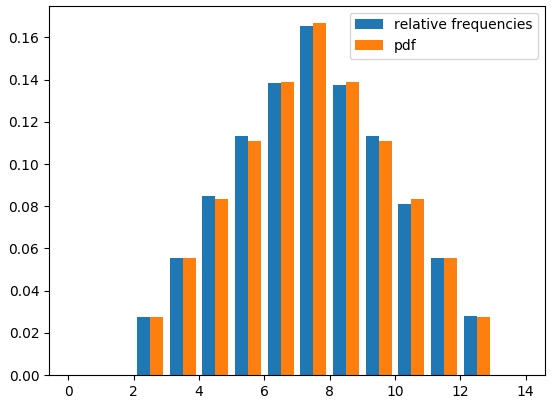
Seed = 3413124, Mean = 6, Stddev = 2.58897

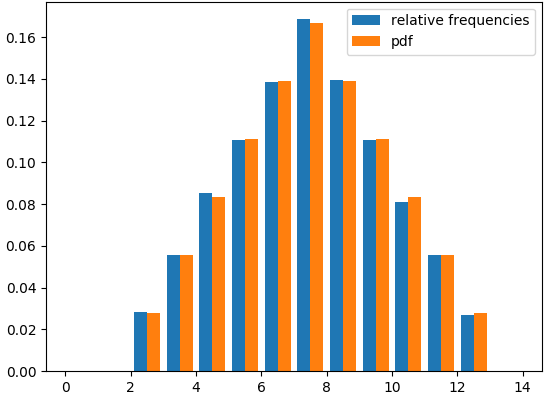
Seed = 98786465, Mean = 6, Stddev = 2.63439

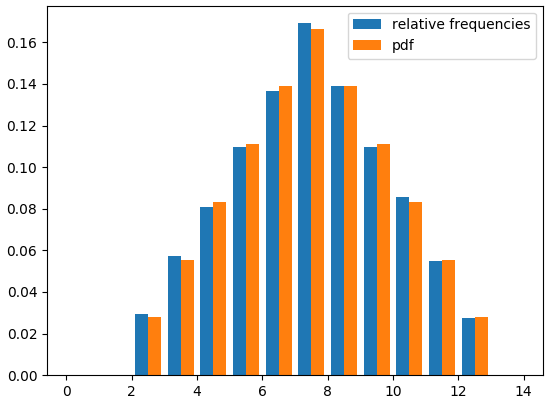
36000 times

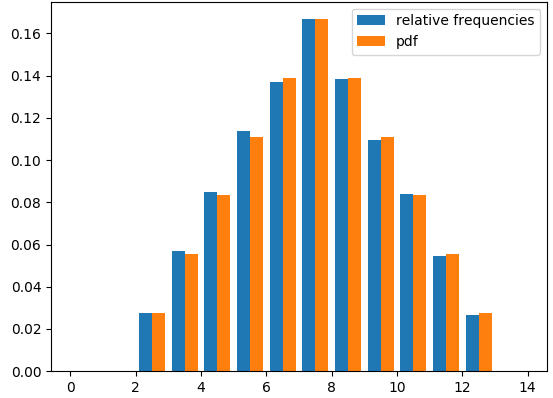
Seed = 12345, Mean = 6, Stddev = 2.60982



Seed = 124564, Mean = 6, Stddev = 2.61027

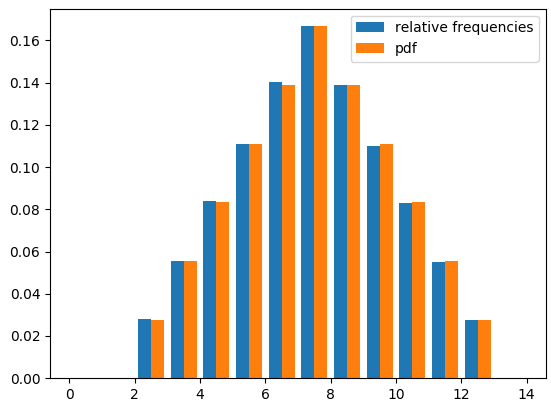
Seed = 242453, Mean = 6, Stddev = 2.6031

Seed = 3413124, Mean = 6, Stddev = 2.6206

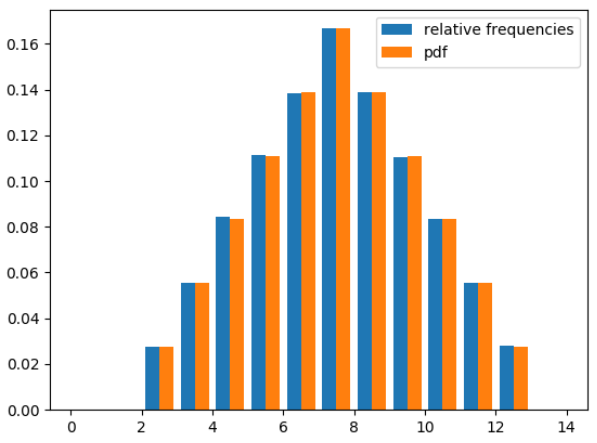
Seed = 98786465, Mean = 6, Stddev = 2.60353

360000 times

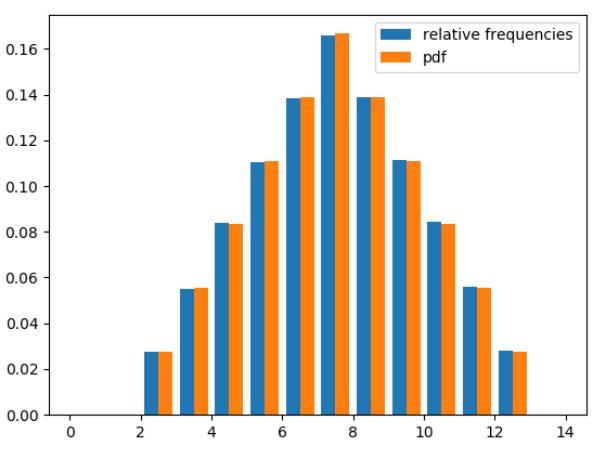
Seed = 12345, Mean = 6, Stddev = 2.60769



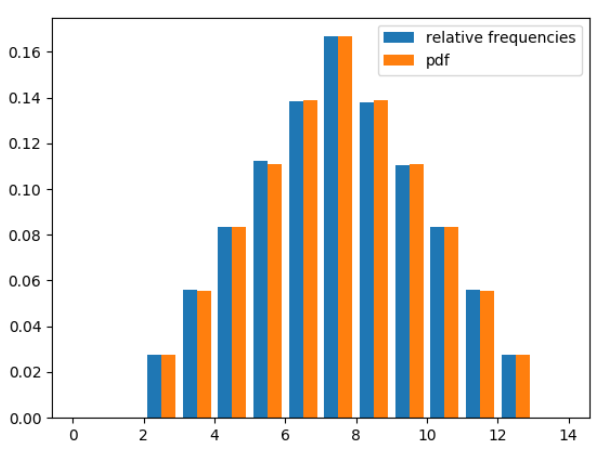
Seed = 124564, Mean = 6, Stddev = 2.61543



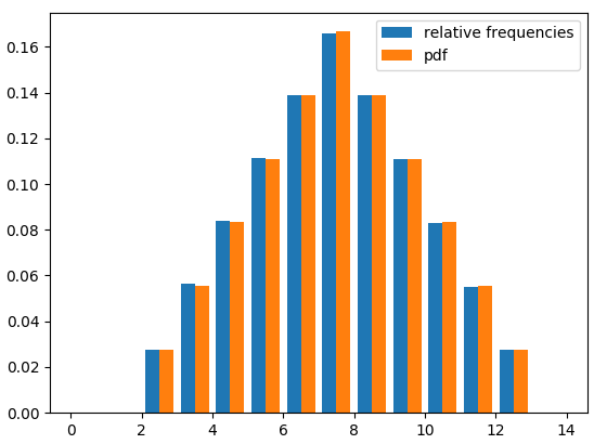
Seed = 242453, Mean = 7, Stddev = 2.41816



Seed = 3413124, Mean = 6, Stddev = 2.61549



Seed = 98786465, Mean = 6, Stddev = 2.61131



**c)** As médias encontradas ficaram entre 6 e 7, enquanto que os desvios padrões variaram de 2.38223 a 2.67758 onde a maioria deles ficou por perto de 2.60

**6.2.3)** Find the pdf associated with the random variate generation algorithm ...

Foi utilizado também para testes, o código main.c

Como a função Random() gera valores entre 0 e 1, então 0 < u < 1, podemos estimar o valor final da função.

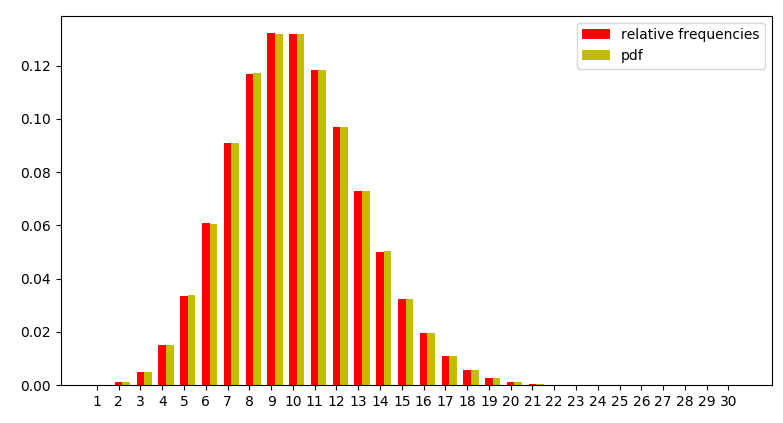
E portanto, . Podemos jogar o x = 4 na funcao para achar o u

2.0 \* u² = 1

Portanto,

**6.2.4)** (a) Generate a Poisson(9) random variate sample of size 1 000 000 using the appropriate generator function in the library rvgs and form a histogram of the results. (b) Compare the resulting relative frequencies with the corresponding Poisson(9) pdf using the appropriate pdf function in the library rvms. (c) Comment on the value of this process as a test of correctness for the two functions used.

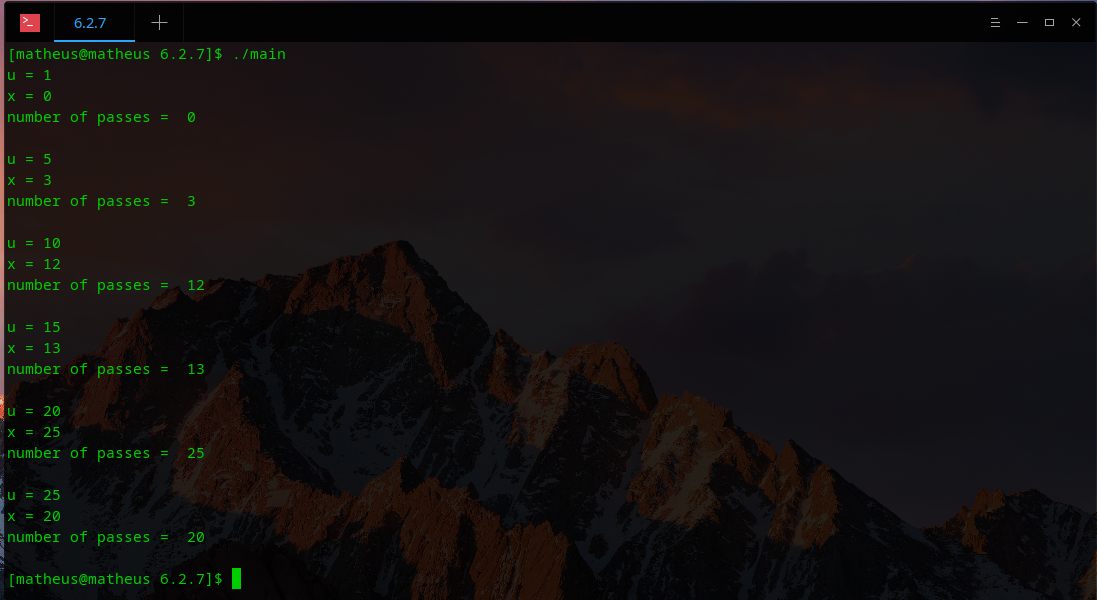
**a) e b)** Códigos usados generateRandom.cpp generateHistogram.py

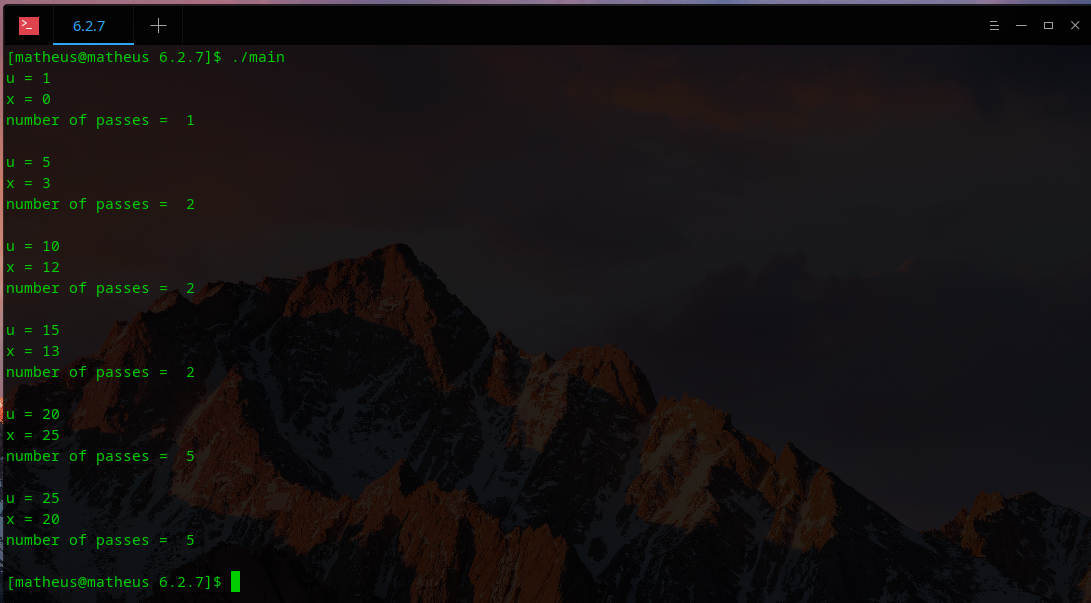


**c)** Como histograma gerado da frequência relativa é possível perceber que a diferença com o p.d.f de Poisson(9) é mínima e que os resultados obtidos são satisfatórios.

**6.2.7)** (a) Implement Algorithm 6.2.1 for a Poisson(μ) random variable and

use Monte Carlo simulation to verify that the expected number of passes through the while loop is μ. Use μ = 1, 5, 10, 15, 20, 25. (b) Repeat with Algorithm 6.2.2. (c) Comment. (Use the function cdfPoisson in the library rvms to generate the Poisson(μ) cdf values.)

**a)** Código usado algorithm621.cpp

**b)** Código usado algorithm622.cpp

**c)** Com o algoritmo 6.2.1 o número médio de passos para achar a variável aleatória não foi igual a , mas se aproxima bastante. Agora com o algoritmo 6.2.2 o número de passos diminuiu drasticamente o que mostra a eficiência do algoritmo.

**6.3.2)** The function GetDemand in program sis4 can return demand amounts outside the range 0, 1, 2. (a) What is the largest demand amount that this function can return? In some applications, integers outside the range 0, 1, 2 may not be meaningful, no matter how unlikely. (b) Modify GetDemand so that the value returned is correctly truncated to the range 0, 1, 2. Do not use acceptance-rejection. (c) With truncation, what is the resulting average demand per time interval and how does that compare to the average with no truncation?

**a)** Para gerar a demanda foi usado a função Geometric(0.2) calculado como:

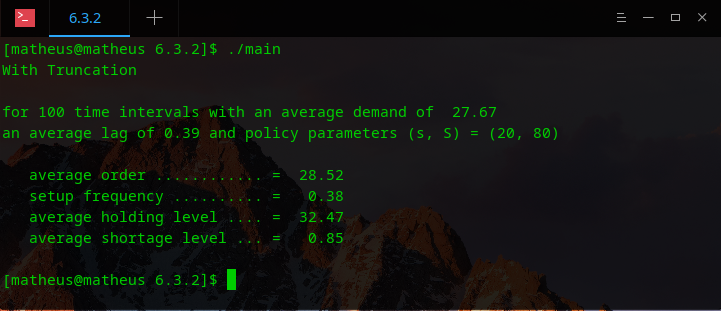
A variável Random que chamaremos de u, pode ter os valores 0 < u < 1

Assim, podemos ver que o x aumenta infinitamente, e quanto maior for o valor de u, maior será o valor de x.

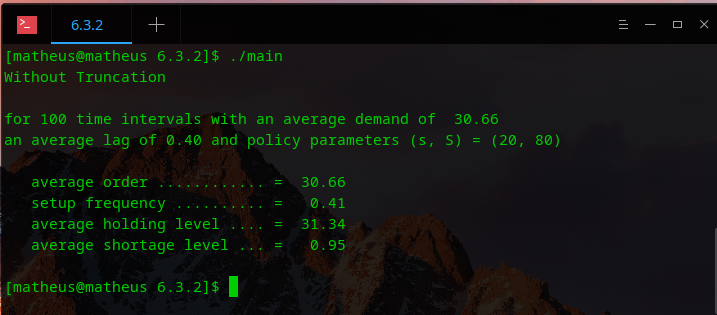
Portanto

**b)** Código usado sis4.c

**c)** With Truncation

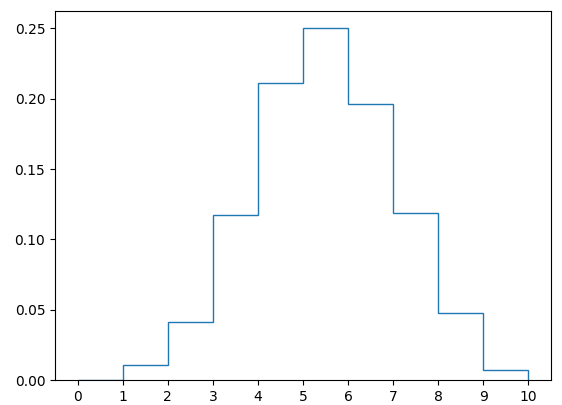
****

Without Truncation



**6.4.1)** (a) Simulate the tossing of a fair coin 10 times and record the number of heads. (b) Repeat this experiment 1000 times and generate a discrete-data histogram of the results. (c) Verify numerically that the relative frequency of the number of heads is approximately equal to the pdf of a Binomial (10, 0.5) random variable.

**a)** 

**b)** Código usado simulateCoin.c generateHistogram.py

**c)**

p.d.f of Binomial:

Relative Frequencies:

**6.4.5**) Verify numerically that the pdf of a Binomial (25, 0.04) random variable is virtually identical to the pdf of a Poisson(μ) random variable for an appropriate value of μ. Evaluate these pdf’s in two ways: by using the appropriate pdf functions in the library rvms and by using the Binomial (n , p) recursive pdf equations.

A relação de Binomial para Poisson pode ser expressa desse modo:

E no exemplo temos que e . Portanto

Código usado: generatepdf.cpp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Library | | Recursion | |
| x | Poisson | Binomial | Poisson | Binomial |
| 0 | 0.36788 | 0.36040 | 0.36788 | 0.36040 |
| 1 | 0.36788 | 0.37541 | 0.36788 | 0.37541 |
| 2 | 0.18394 | 0.18771 | 0.18394 | 0.18771 |
| 3 | 0.06131 | 0.05996 | 0.06131 | 0.05996 |
| 4 | 0.01533 | 0.01374 | 0.01533 | 0.01374 |
| 5 | 0.00307 | 0.00240 | 0.00307 | 0.00240 |
| 6 | 0.00051 | 0.00033 | 0.00051 | 0.00033 |
| 7 | 0.00007 | 0.00004 | 0.00007 | 0.00004 |
| 8 | 0.00001 | 0.00000 | 0.00001 | 0.00000 |
| 9 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 10 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 11 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 12 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 13 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 14 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 15 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 16 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 17 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 18 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 19 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 20 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 21 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 22 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 23 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 24 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 25 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |