Самостоятельная работа на тему:

Моделирование и визуализация случайных данных

Студент: Уколов Алексей

Группа: П2-3

Преподаватель: к.ф.-м.н. Мещеряков В.В.

Москва 2019

Оглавление

Оглавление

Предисловие

Генерирования и визуализации случайных чисел с равномерным распределением.

Линейный конгруэнтный метод- один из методов генерации псевдослучайных чисел. Применяется в простых случаях и не обладает криптографической стойкостью. Входит в стандартные библиотеки различных компиляторов. Линейный конгруэнтный метод был предложен Д.Г. Лемером в 1949.Суть метода заключается в вычислении последовательности случайных чисел x_n, полагая

$$x_{n-1} = (aX_n + c) \bmod m$$

Где m-модуль ,a-множитель,c-приращение.

Используемые модули; math – библиотека математических функций, арифметических и алгебраических вычислений и преобразований

sys – модуль, предоставляющий информацию о константах, функциях и методах интерпретатора Python

random – библиотека генераторов псевдослучайных чисел

NumPy – расширение языка Python средствами численных методов, поддержка больших многомерных массивов и матриц, библиотека высокоуровневых математических функций для обработки массивов и матриц

Matplotlib – расширение для 2D- и 3D-визуализации данных, обрабатываемых языковыми средствами Python и NumPy

SymPy – расширение для символьных вычислений, обеспечивающих аналитические возможности методов компьютерной алгебры

PyDOE – расширение для формирования матриц дизайна эксперимента

Глава 1 программное генерирование и визуализация случайных чисел

Программными генераторами случайных чисел называют реализации детерминированных численных алгоритмом, позволяющих создавать конечные или периодически повторяющиеся ряды псевдослучайных чисел.

Псевдослучайные числа с достаточно длинными периодами и малыми значениями корреляций полагают аппроксимирующими случайные числа, являющиеся некоррелированными результатами физических измерений, и наряду с ними широко используют в практике статистического моделирования.

Псевдослучайные числа (в дальнейшем, где не потребуется обращать внимание на псевдослучайность, просто – случайные числа) могут быть определены на разных типах чисел, в различных числовых диапазонах и с различным характером распределения. В этом разделе будут рассмотрены решения задач с привлечением некоторых численных низкоуровневых и высокоуровневых средств генерирования случайных чисел,подчиняющихся равномерному распределению. В равномерном распределении U (a, b)случайные числа встречаются с равной вероятностью на заданном интервале [a; b].

Базовый характер распределения U (a, b) связан с тем, что он является основой для построения алгоритмов генерации случайных чисел с произвольным распределением.

1.1 Линейный конгруэнтный алгоритм

Этот алгоритм генерации случайных чисел основан на использовании итерационной процедуры вычисления числа

$$x_{k+1}=ax_k+c \bmod m.$$

где параметры a, c и m – целые числа. Операция mod m определяет число xk+1 равным остатку от деления числа axk + c на число m. В случае c = 0 алгоритм имеет мультипликативный характер.

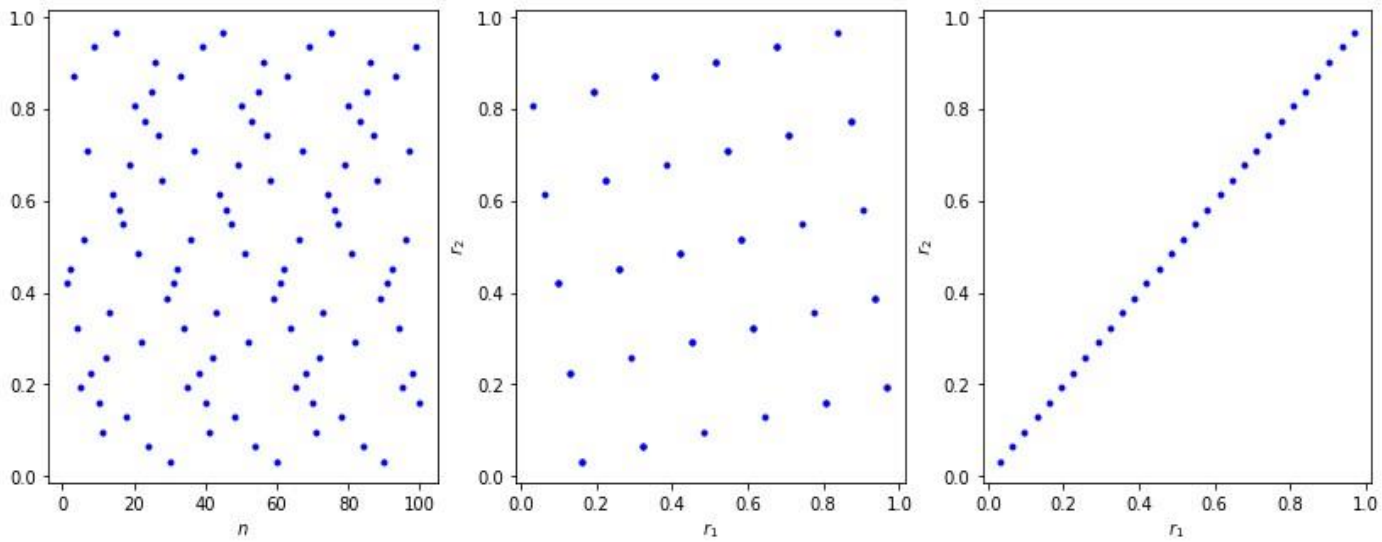
Задача 1. Построить в форме Python-модуля конгруэнтный мультипликативный 32-битовый генератор (Multiplicative congruential generator) случайных чисел, основывающийся на мультипликативном конгруэнтном алгоритме с параметрами a = 13, c = 0 и m = 31. В серии численных экспериментов дать иллюстрацию формирования ряда случайных чисел. Для построения листинга программного кода надо воспользоваться высокоуровневой процедурой % нахождения остатка от деления и предусмотреть нормировку случайных чисел.

Листинг из пособия:

```
import numpy as np #x0 -
стартовая точка #N - число
генерируемых чисел def
MCG_std(a,c,m,x0,N):     xi
= x0
x     = np.zeros(N); x =
x.astype(np.int32)     r = np.zeros(N)
for k in range(N):
xi    = (a*xi + c) % m          x[k] = xi
r[k] = xi/m  return x, r if __name__ ==
"__main__":   a = 13; c = 0; m = 31; x0 =
1; N = 12;
  #a = 7**5; c = 0; m = 2**31 - 1; x0 = 1; N = 10;
x, r = MCG_std(a,c,m,x0,N)   print('x = ',x)
  print('r = ',np.round(r,4))

IPython console: x = [13 14 27 10 6
16 22 7 29 5 3 8]
r = [ 0.4194 0.4516 0.8710 0.3226 0.1935 0.5161 0.7097 0.2258 0.9355 0.1613 0.0968 0.2581]
```

графическая иллюстрация ряда случайных чисел (из пособия) при

```
a = 13; c = 0; m = 31; x0 = 1; N = 12;
```



Листинг 1.2. Скрипт MCG_std_subplot.py из пособия

```
import MCG_std as MC import
numpy as np
import matplotlib.pyplot as plt a = 13;
c = 0; m = 31; x0 = 1; N = 100;
#a = 7**5; c = 0; m = 2**31 - 1; x0 = 1; N = 100;
x, r = MC.MCG_std(a,c,m,x0,N);  fig = plt.gcf()
fig.set_size_inches(14,5)

plt.subplot(131)
plt.plot(np.linspace(1,N,N),r, 'b.' )
plt.xlabel('$n$'),  plt.ylabel('$r$')

plt.subplot(132)
r1 = r[0:N/2]; r2 = r[N/2:N + 1]
plt.plot(r1,r2, 'b.' )
plt.xlabel('$r_1$'),  plt.ylabel('$r_2$')
 plt.subplot(133)
N = 60 x,
r = MC.MCG_std(a,c,m,x0,N); r1 = r[0:N/2]; r2 = r[N/2:N + 1]
plt.plot(r1,r2, 'b.' ) plt.xlabel('$r_1$'),
plt.ylabel('$r_2$')
```

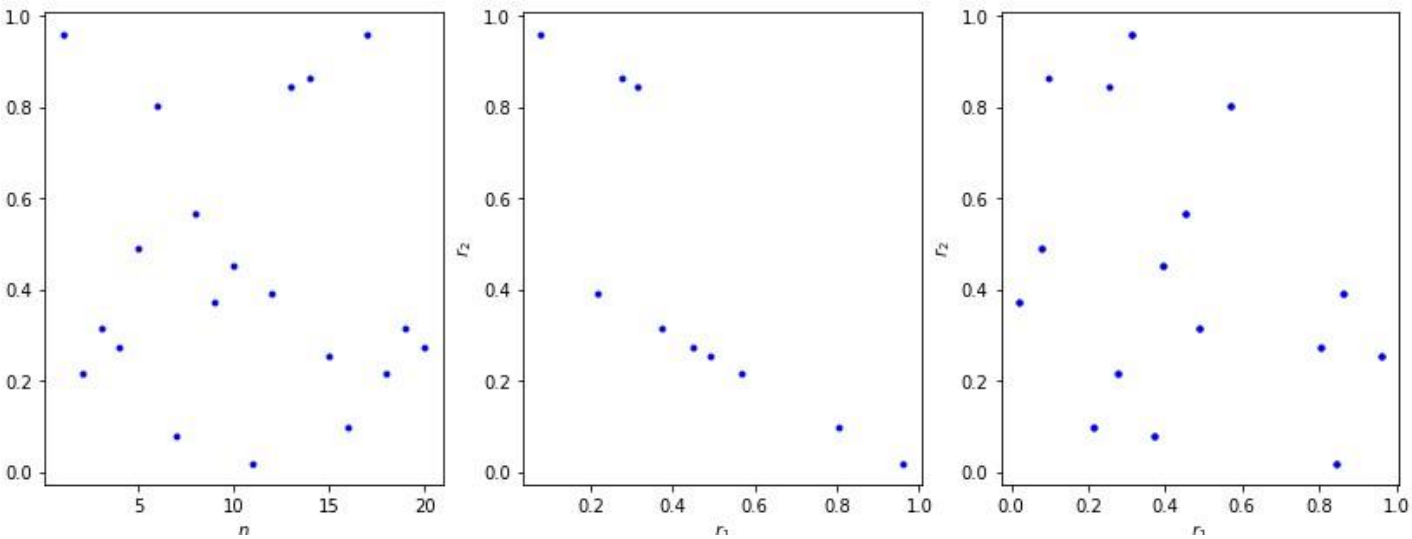
Своя проработка с другими исходными данными

```
import numpy as np def
MCG_std(a,c,m,x0,N):
  xi = x0
  x = np.zeros(N); x = x.astype(np.int32)
r = np.zeros(N)   for k in range(N):
xi = (a*xi + c) % m     x[k] = xi
r[k] = xi/m  return x, r if __name__ ==
"__main__":  a = 20; c = 0; m = 51; x0 =
5; N = 20;  x, r = MCG_std(a,c,m,x0,N)
print('x = ',x)
 print('r = ',np.round(r,4))
```

```
IPython console: x =  [49 11 16 14 25 41  4 29 19 23  1 20 43 44 13  5 49
11 16 14] r =  [0.9608 0.2157 0.3137 0.2745 0.4902 0.8039 0.0784 0.5686
0.3725 0.451  0.0196 0.3922 0.8431 0.8627 0.2549 0.098  0.9608 0.2157 0.3137
0.2745]
```

графическая иллюстрация ряда случайных чисел при

```
a = 20; c = 0; m = 51; x0 = 5; N = 20;
```



Листинг графической иллюстрации

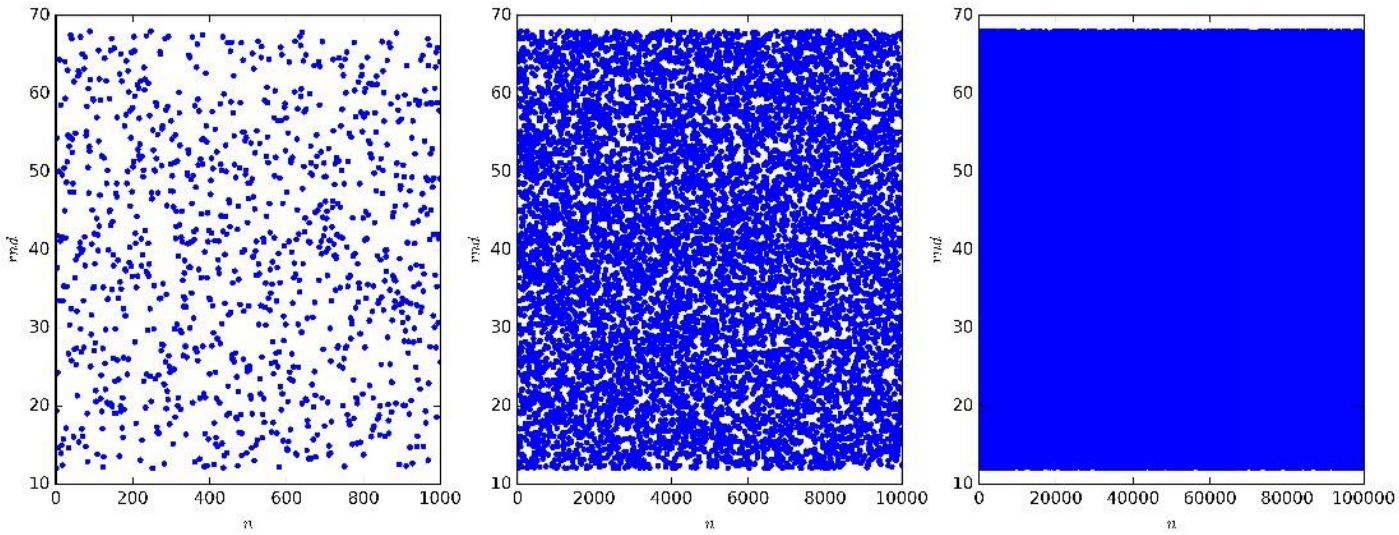
```
import MCG_std as MC  import numpy as
np  import matplotlib.pyplot as plt
a = 20; c = 0; m = 51; x0 = 5; N =
20;

x, r = MC.MCG_std(a,c,m,x0,N);
  fig = plt.gcf()
fig.set_size_inches(14,5
)
 plt.subplot(131)
plt.plot(np.linspace(1,N,N),r, 'b.' )
plt.xlabel('$n$'),
plt.ylabel('$r$')  plt.subplot(132)

r1 = r[0:N//2]; r2 = r[N//2:N + 1]
plt.plot(r1,r2, 'b.' )
plt.xlabel('$r_1$'),
plt.ylabel('$r_2$')

plt.subplot(133)  N
= 60
x, r = MC.MCG_std(a,c,m,x0,N);  r1
= r[0:N//2]; r2 = r[N//2:N + 1]
plt.plot(r1,r2, 'b.' )
plt.xlabel('$r_1$'),
plt.ylabel('$r_2$')
```

1.2 Задача с отсутствием псевдослучайных чисел

Задача 2. Основываясь на линейном конгруэнтном алгоритме в параметрами a = 7 5 , m = 231 − 1, построить 32-битовый генератор равномерно распределённых чисел на вещественном интервале [low; $high$], где low = 12, $high$ = 68. Дать графическую иллюстрацию числу N = 103 , N = 104 и N = 105 случайных чисел.

Листинг 2.1. Модуль MCG_rnd.py
```
import numpy as np
a = 7**5; c = 0; m = 2**31 - 1;
 def
MCG_rnd(low,high,x0,N):
xi = x0
  x = np.zeros(N); x = x.astype(np.int32)
r = np.zeros(N)   for k in range(N):
xi = (a*xi + c) % m     x[k] = xi
r[k] = xi/m
  rnd = low + (high - low)*r
return rnd
```

Рис. 1.3. Иллюстрация случайных чисел генератора MCG_rnd.py. из пособия



Листинг 2.2. Скрипт MCG_rnd_subplot.py из пособия

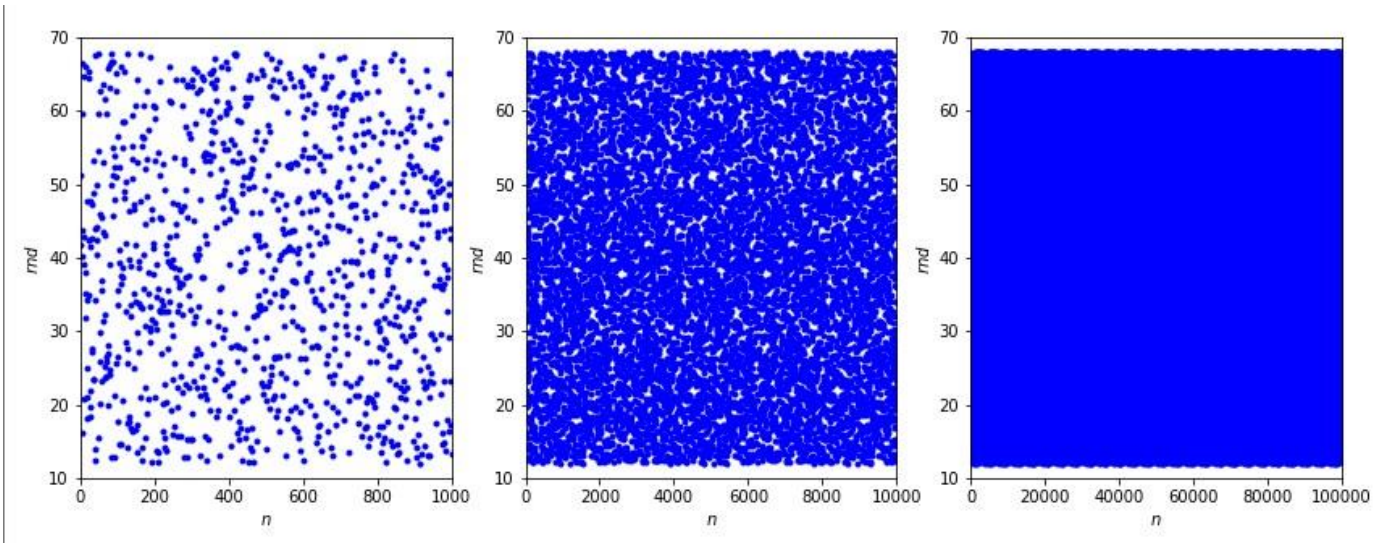
```
import MCG_rnd as MC  import
numpy as np
import matplotlib.pyplot as plt

low = 12; high = 68; x0 = 1;

fig = plt.gcf()
fig.set_size_inches(14,5)
 for n in
range(3,6):
 plt.subplot(1, 3, n - 2)
N = 10**n
 rnd = MC.MCG_rnd(low,high,x0,N)
plt.plot(np.linspace(1,N,N),rnd, 'b.' )
plt.axis([0, N, 10 ,70])
plt.xlabel('$n$'),
 plt.ylabel('$rnd$ ')
```

Своя проработка задачи с псевдослучайными числами

```
import numpy as np
a = 9**23; c = 0; m = (2**23) - 1; def
MCG_rnd(low,high,x0,N):
  xi = x0
  x = np.zeros(N); x = x.astype(np.int32)
r = np.zeros(N)   for k in range(N):
xi = (a*xi + c) % m    x[k] = xi    r[k]
= xi/m
  rnd = low + (high - low)*r
return rnd
```

графическая иллюстрация псевдослучайных чисел при

```
a = 9**23; c = 0; m = (2**23) - 1;
```



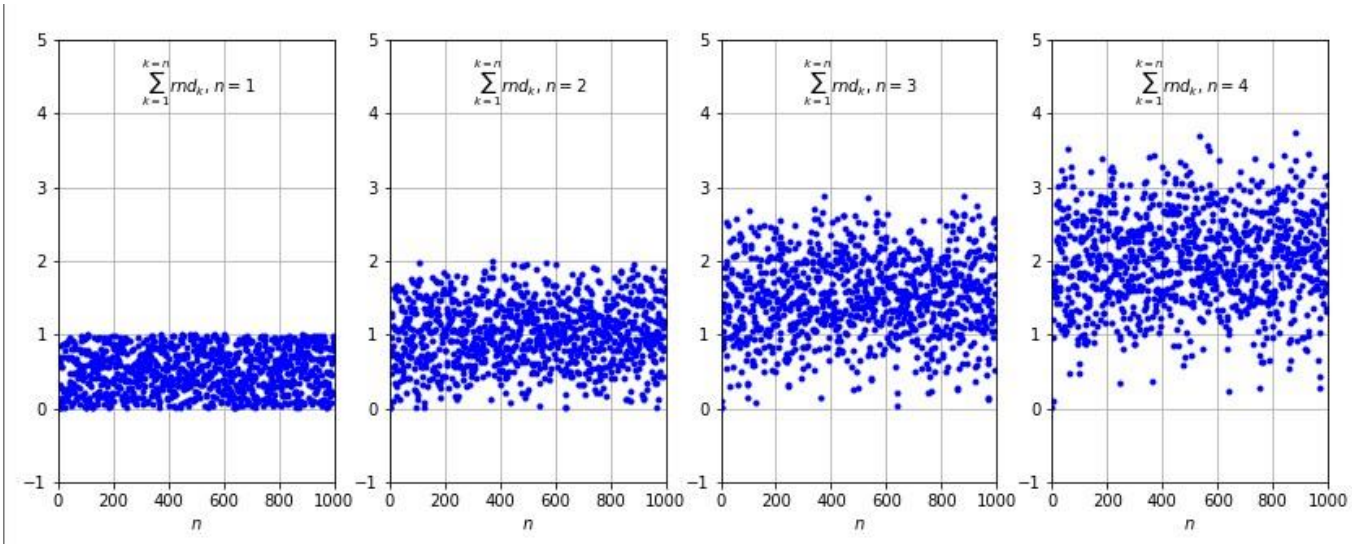
Листинг скрипта

```
import MCG_rnd as MC  import
numpy as np
import matplotlib.pyplot as plt

low = 12; high = 68; x0 = 1;

fig = plt.gcf()
fig.set_size_inches(14,5)
 for n in
range(3,6):
  plt.subplot(1, 3, n - 2)
N = 10**n
  rnd = MC.MCG_rnd(low,high,x0,N)
plt.plot(np.linspace(1,N,N),rnd, 'b.' )
plt.axis([0, N, 10 ,70])
plt.xlabel('$n$'),    plt.ylabel('$rnd$
')
```

1.3 Задача суммы равномерно распределённых псевдослучайных чисел.

```
import MCG_rnd as MC  import
numpy as np
import matplotlib.pyplot as plt
N = 10**3 low = 0; high =
1; fig = plt.gcf()
fig.set_size_inches(14,5
) n = np.linspace(1,N,N)
rnd = np.zeros(N)  for k
in range(1,5):
  plt.subplot(1,4,k) x0 = 10**k    rnd
= rnd + MC.MCG_rnd(low,high,x0,N)
plt.plot(n,rnd, 'b.' )
 (plt.text(300,4.3,
 '$\sum_{k=1}^{k=n}{rn{{d}_{k}}}$, $n =$' + str(k)))
plt.xlabel('$n$')    plt.axis([0, N, -1 ,5])
plt.grid()
```

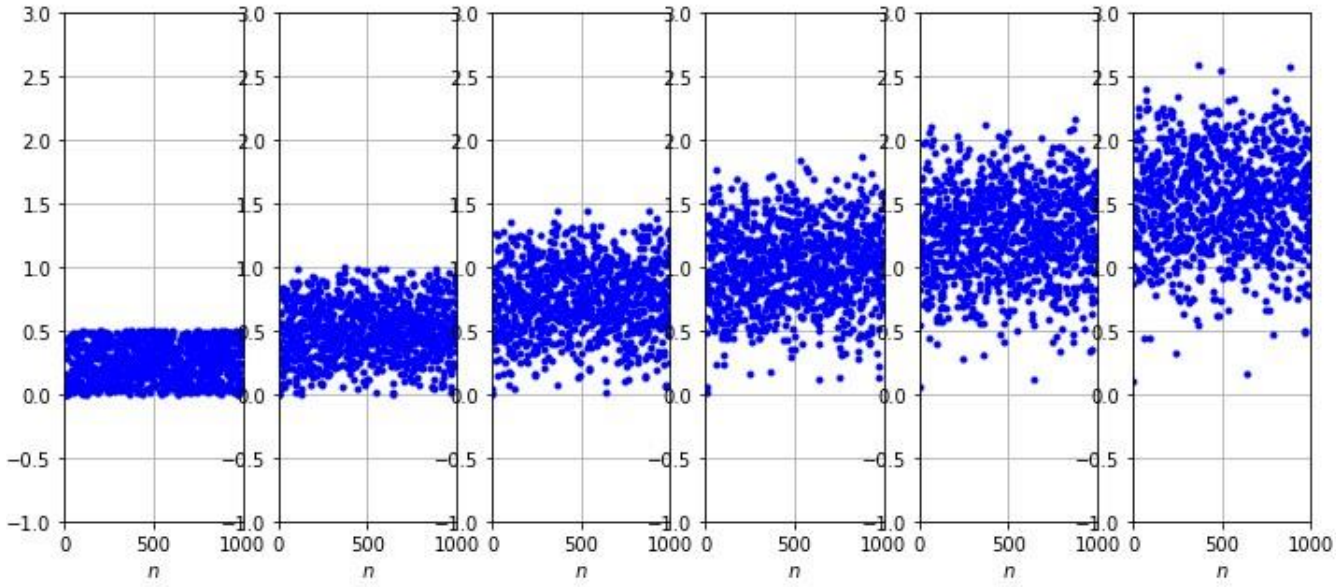
Рис. 1.4. Суммы равномерно распределённых случайных чисел из пособия



Задача со своими исходными значениями

```
import MCG_rnd as MC import
numpy as np
import matplotlib.pyplot as plt
N = 10**3 low = 0; high = 0.5;
fig = plt.gcf()
fig.set_size_inches(14,5) n =
np.linspace(1,N,N) rnd =
np.zeros(N) for k in
range(1,7):  plt.subplot(1,7,k)
x0 = 10**k
 rnd = rnd + MC.MCG_rnd(low,high,x0,N)
plt.plot(n,rnd, 'b.' )
 (plt.text(300,4.3,
'$\sum_{k=1}^{k=n}{rn{{d}_{k}}}$, $n =$' + str(k)))
plt.xlabel('$n$')  plt.axis([0, N, -1 ,3])
plt.grid()
```

график иллюстрирующий распределение псевдослучайных чисел из кода



1.4 Проверка равномерности распределения

Для проверки равномерности распределения случайных чисел генератора MCG_rnd найти абсолютную величину относительной разностиполного числа N = 10**6 и удвоенного числа значений, меньших 0.5, для стартовых точек x_{01} = 10**2 , x_{02} = 10**4 , x_{03} = 10**6 и x_4 = 10*8

$$\frac{\Delta n}{N} \quad \frac{|N - 2n(rnd < 0.5)|}{N} =$$

листинг проверки

```
import MCG_rnd as MC
import numpy as np
  N = 10**2 print('N
=',N) low = 0; high
= 1; for m in
range(1,5):
       x0 =
10**(2*m)
  rnd = MC.MCG_rnd(low,high,x0,N)
n  = 0   for k in range(N):     if
rnd[k] < 0.5:          n = n + 1 r
=   abs(N   -   2*n)/N   x0_n_r   =
np.array([x0,n,r])  print('x0_n_r
=',x0_n_r)
```

(листинг rnd.py)

```
   import numpy as np
   a = 9**23; c = 0; m = (2**23) - 1; def
   MCG_rnd(low,high,x0,N):
     xi = x0
     x = np.zeros(N); x = x.astype(np.int32)
   r = np.zeros(N)   for k in range(N):
   xi = (a*xi + c) % m    x[k] = xi    r[k]
   = xi/m
     rnd = low + (high - low)*r
   return rnd
```

результат вывода консоли:
при 10**2 N = 100
x0_n_r = [1.0e+08 3.9e+01 2.2e-01]

при 10**4 N
= 10000
x0_n_r = [1.000e+08 4.935e+03 1.300e-02]

при 10**6 N
= 1000000
x0_n_r = [1.00000e+08 4.96383e+05 7.23400e-03]

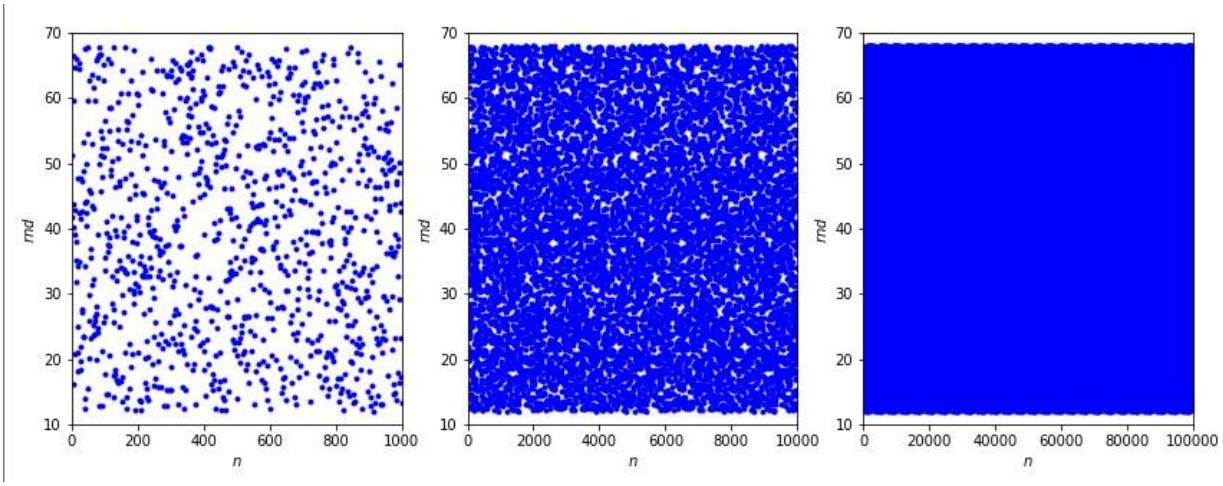
при 10**8 N =
100000000
ничего не выдал

1.5 Высокоуровневый генератор языка Python

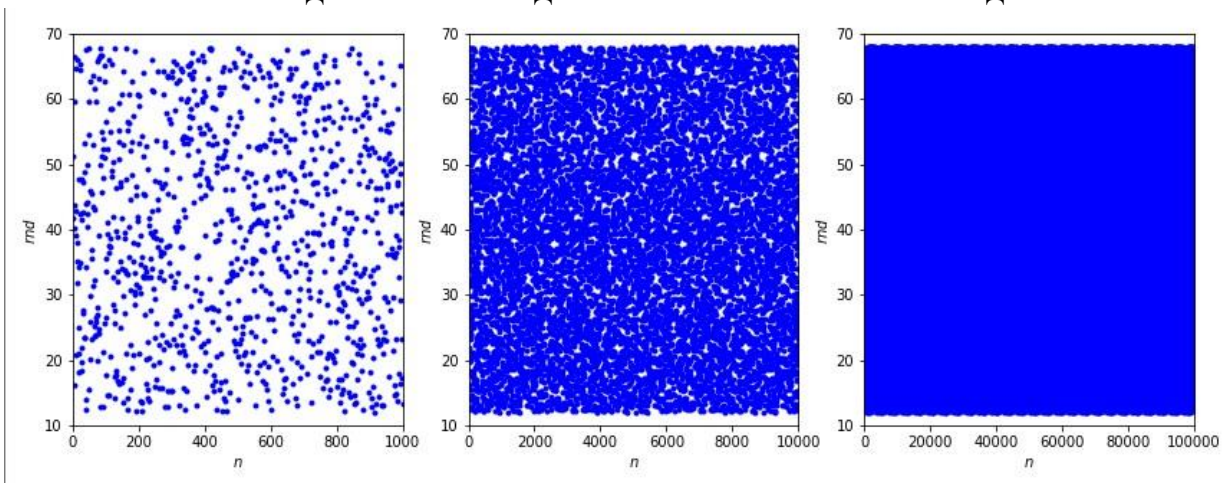
Высокоуровневый генератор языка Python В большинстве современных программных сред и, в том числе, в языке Python для генерирования случайных чисел используется алгоритм Mersenne Twister. Этот алгоритм способен теоретически воспроизвести ряд псевдослучайных чисел с периодом 2 19937 − 1, который значительно превышает максимальную величину целого 2 63 − 1, реализуемого в современных 64-битовых вычислительных системах:

```
import random
import matplotlib.pyplot as plt
low = 12; high = 68; fig =
plt.gcf()
fig.set_size_inches(14,5) for k
in range(3,6):
 plt.subplot(1, 3, k - 2)
N = 10**k  n = range(N)
 rnd = [random.uniform(low,high) for i in n]
plt.plot(n, rnd, 'b.')  plt.axis([0, N, 10
,70])  plt.xlabel( ' $n$ ' ),  plt.ylabel( '
$rnd$ ' )
```

РЕЗУЛЬТАТ ВЫСОКОУРОВНЕГО ГЕНЕРАТОРА



РЕЗУЛЬТАТ ИЗ ЗАДАЧИ ПО РАСПРЕДЕЛЕНИЮ В ПРОСТРАНСТВЕ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ



1.6 Задача генератор T-wister

Для проверки равномерности распределения случайных чисел генератора numpy.random.uniform найти по формуле задачи 4 абсолютную величину относительной разности полного числа N = 10**6 и удвоенного числа значений, меньших 0.5, для стартовых точек x_{01} = 10**2 , x_{02} = 10**4 , x_{03} = 10**6 и x_{04} = 10**8 .

Листинг N
= 10**6
print('N =',N)
low = 0; high = 1;

for m in range(1,5):
x0 = 10**(2*m)
np.random.seed(x0)
rnd = np.random.uniform(low,high,N); #print(rnd)
n = 0 for k in range(N): if rnd[k] < 0.5: n =
n + 1 r = abs(N - 2*n)/N x0_n_r =
np.array([x0,n,r])
print('x0_n_r =',x0_n_r)

вывод консоли: N = 1000000 x0_n_r =
[1.0000e+02 5.0014e+05 2.8000e-04] x0_n_r =
[1.00000e+04 5.00331e+05 6.62000e-04] x0_n_r =
[1.00000e+06 4.99994e+05 1.20000e-05] x0_n_r =
[1.00000e+08 4.99481e+05 1.03800e-03]

Глава 2 задачи о блуждании по прямой

2.1

Классическая вероятность Вероятность $P\ (A)$ случайного события A равна отношению числа m исходов, благоприятствующих этому событию, к полному числу N равновозможных исходов, т.е. $P\ (A) = m\ N$. (2.1) В этом определении $m \leq N$. Поэтому вероятность события должна удовлетворять неравенствам $0 \leq P\ (A) \leq 1$. Для события из суммы $m1 + m2 + ... + mn = N$ исходов вероятность $P\ (A) = 1$.
(материал из пособия)

2.2

Статистическая вероятность Статистической вероятностью $P\ (A)$ события A называют относительную частоту этого события в N проведённых испытаниях: $P\ (A) = m\ N$, (2.2) где m число испытаний, в которых появилось событие A. (материал из пособия)

2.3

Общая формулировка задачи о блуждании Найти число возможных траекторий частицы, которая в каждый временной интервал $\Delta t = 1$ с равной вероятностью либо перемещается вдоль прямой линии в одну сторону на шаг $a = 1$, либо остаётся на месте. В модельном эксперименте найти вероятности случайного перемещения частицы на m шагов за n временных интервалов при равной вероятности заданных событий (материал из пособия)

Возникновение понятия и теории вероятностей[

Первые работы о вероятности относятся к 17 веку. Такие как переписка французских учёных Б. Паскаля, П. Ферма (1654 год) и нидерландского учёного Х. Гюйгенса (1657 год) давшего самую раннюю из известных научных трактовок вероятности[5]. По существу Гюйгенс уже оперировал понятием математического ожидания. Швейцарский математик Я. Бернулли, установил закон больших чисел для схемы независимых испытаний с двумя исходами (посмертно, 1713 год).
В XVIII в. — начале XIX в. теория вероятностей получает развитие в работах А. Муавра (Англия, 1718 год), П. Лаплас (Франция), К. Гаусса (Германия) и С. Пуассона (Франция). Теория вероятностей начинает применяться в теории ошибок наблюдений, развившейся в связи с потребностями геодезии и астрономии, и в теории стрельбы. Необходимо отметить, что закон распределения ошибок по сути предложил Лаплас сначала как экспоненциальная зависимость от ошибки без учета знака (в 1774 год), затем как экспоненциальную функцию квадрата ошибки (в 1778 году). Последний закон обычно называют распределением Гаусса или нормальным распределением. Бернулли (1778 год) ввел принцип произведения вероятностей одновременных событий. Адриен Мари Лежандр (1805) разработал метод наименьших квадратов.

Во второй половине XIX в. развитие теории вероятностей связано с работами русских математиков П. Л. Чебышёва, А. М. Ляпунова и А. А. Маркова (старшего), а также работы по математической статистике А. Кетле (Бельгия) и Ф. Гальтона (Англия) и статистической физике Л. Больцмана (в Австрии), которые создали основу для существенного расширения проблематики теории вероятностей. Наиболее распространённая в настоящее время логическая (аксиоматическая) схема построения основ теории вероятностей разработана в 1933 советским математиком А. Н. Колмогоровым. (материал из wikipedia)

2.4 Генерация блуждания и визуализация траекторий

Листинг 7.1. Модуль steps.py из пособия

```
import numpy as np
 def binrnd(n):     rnd =
(np.append(np.array([0]),
                   np.random.random_integers(0, 1, n)))
return rnd
 def
xm(n):
  rnd = binrnd(n);   rnd_temp =
np.array(rnd)   x = rnd_temp
for k in range(n):       x[k +
1] = x[k + 1] + x[k]   return x,
rnd
 if __name__ ==
"__main__":
    n = 10     x, rnd
= xm(n)     print('rnd
=',rnd)     print('x
=',x) консоль ipython:
  rnd = [0 1 0 1 1 0 1 1 0 0 1]
  x = [0 1 1 2 3 3 4 5 5 5 6]
```

Листинг 7.2. Скрипт random_walk.py из пособия

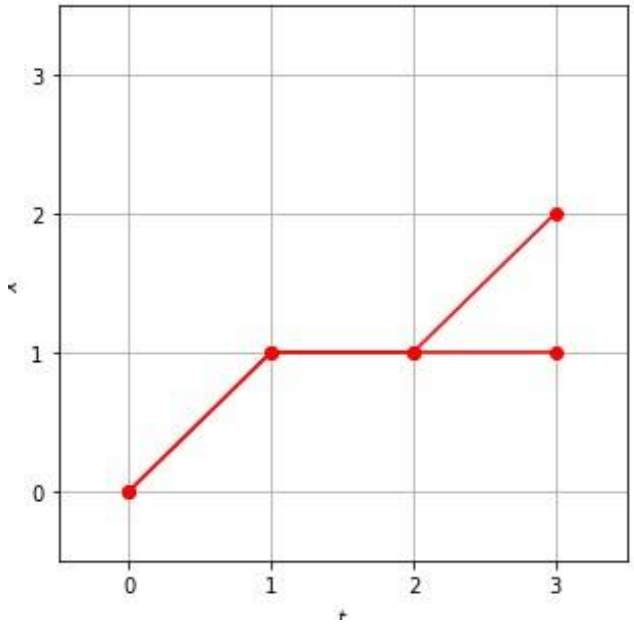
```
import steps import
numpy as np
import matplotlib.pyplot as plt

n = 3; N = 2

# Графическое окно plt.close('all')
fig = plt.gcf()
fig.set_size_inches(5,5)
plt.xlabel('$t$'); plt.ylabel('$x$')
plt.grid()
 for j in
range(N):
   x, rnd = steps.xm(n)
   t = np.linspace(0,n,n + 1); t = t.astype(np.int32)
plt.plot(t,x,'ro-')    if n <= 20:
       plt.xlim(-.5,n + 0.5); plt.ylim(-.5,n + 0.5)
plt.xticks(np.arange(0,n + 1,1))
plt.yticks(np.arange(0,n + 1,1)) if N <= 10:
print('binrnd',j+1,'=',rnd)
print('x',j+1,'=',x)
```

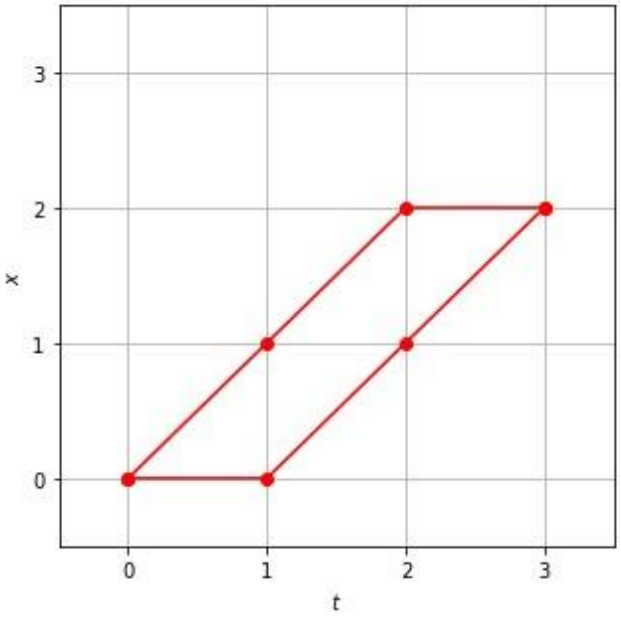
```
 консоль ipython:
binrnd 2 = [0 1 0 0]
x 2 = [0 1 1 1]
```

полученный результат в результате исполнения кода

```
binrnd 2 = [0 1 0 0]
x 2 = [0 1 1 1]
```

binrnd 2 = [0 0 1 1]
x 2 = [0 0 1 2]





Типы переменных листингов

Имя	Тип	Размер	
N	int	1	2
j	int	1	1
n	int	1	10
rnd	int32	(11,)	[0 1 0 ... 0 0 1]
t	int32	(4,)	[0 1 2 3]
x	int32	(11,)	[0 1 1 ... 5 5 6]

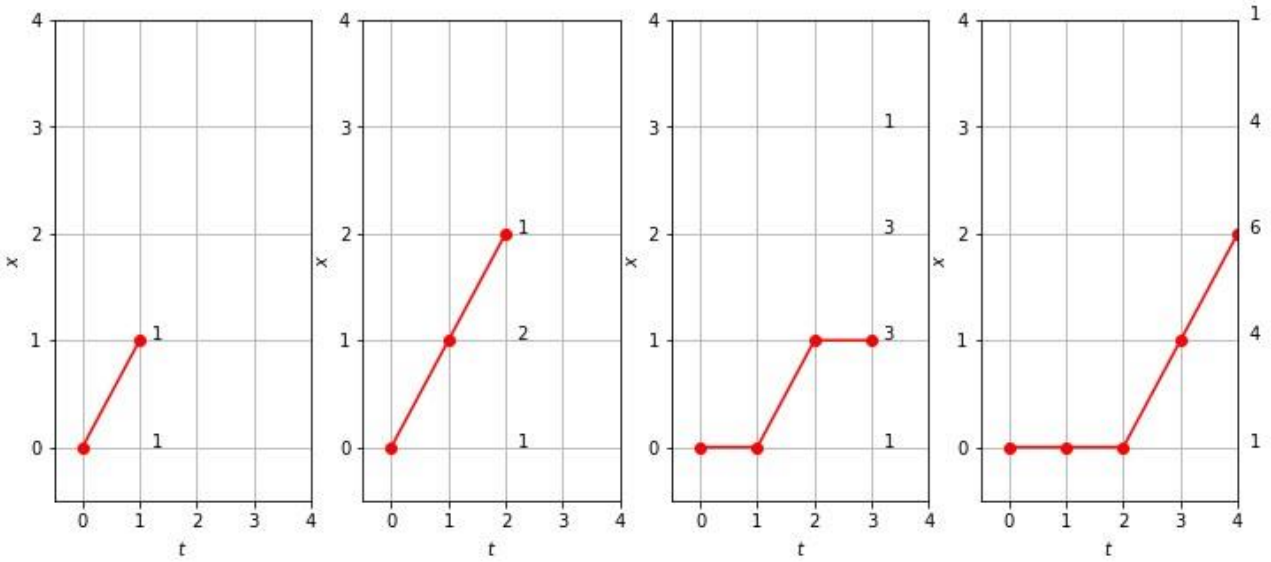
2.5 Число траекторий и биномиальные коэффициенты

В математике **биномиальные коэффициенты** — это коэффициенты в разложении бинома Ньютона по степеням *x*. Коэффициент при обозначается или и читается «биномиальный коэффициент из *n* по *k*» (или «числосочетаний из *n* по *k*», читается как «це из *n* по *k*»):для натуральных степеней .
Биномиальные коэффициенты могут быть также определены для произвольных действительных чисел . В случае произвольного действительного числа биномиальные коэффициенты определяются как коэффициенты разложения выражения в бесконечный степенной ряд:
Для неотрицательных целых *a* все коэффициенты с индексами *k>a* в этом ряду являются нулевыми (т.е.), и поэтому данное разложение представляет собой конечную сумму (1).
В комбинаторике биномиальный коэффициент для неотрицательных целых чисел *n* и *k* интерпретируется как количество сочетаний из *n* по *k*, то есть количество всех подмножеств (выборок) размера *k* в *n*элементном множестве.
Биномиальные коэффициенты часто возникают в задачах комбинаторики и теории вероятностей.
Обобщением биномиальных коэффициентов являются мультиномиальные коэффициенты.

Задача.

. Установить в численном статистическом эксперименте возможное число траекторий при перемещении частицы в течение временных интервалов

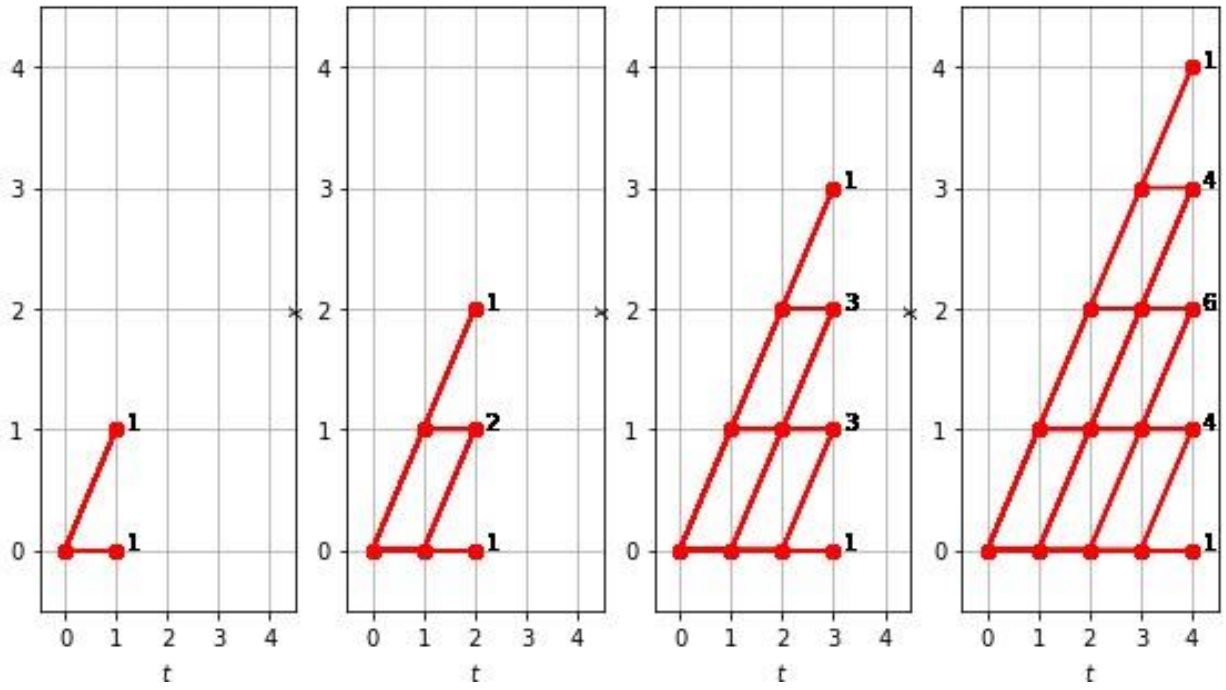
```
import matplotlib.pyplot as plt
import numpy as np import steps
 plt.close('all')
fig = plt.gcf()
fig.set_size_inches(12,5)
 N = 7 nmax = 4 for n in
range(1,nmax + 1):    for j
in range(N):
       x, rnd = steps.xm(n)
       t = np.linspace(0,n,n + 1); t = t.astype(np.int32)
plt.subplot(1,4,n)    plt.plot(t,x,'ro-')
plt.grid(True)
   plt.xlim(-.5,3.5); plt.ylim(-.5,3.5)
plt.xlabel('$t$'); plt.ylabel('$x$')
plt.xticks(np.arange(0,5,1))
plt.yticks(np.arange(0,5,1))    if n ==
1:
        plt.text(1.2,0,'1')
plt.text(1.2,1,'1')    elif n
== 2:
        plt.text(2.2,0,'1')
plt.text(2.2,1,'2')
plt.text(2.2,2,'1')    elif n
== 3:
        plt.text(3.2,0,'1')
plt.text(3.2,1,'3')
plt.text(3.2,2,'3')
plt.text(3.2,3,'1')    elif
n==4:
plt.text(4.2,0,'1')
plt.text(4.2,1,'4')
plt.text(4.2,2,'6')
plt.text(4.2,3,'4')
plt.text(4.2,4,'1')
```



Возможные траектории для $N = 7$ запусков блуждания частицы.

```
import matplotlib.pyplot as plt
import numpy as np
import steps

plt.close('all')
fig = plt.gcf()
fig.set_size_inches(12,5)

N = 90
nmax = 4
for n in range(1,nmax + 1):
    for j in range(N):
        x, rnd=steps.xm(n)
        t = np.linspace(0,n,n + 1); t = t.astype(np.int32)
plt.subplot(1,5,n)
plt.plot(t,x,'ro-')
plt.grid(True)
plt.xlim(-.5,4.5); plt.ylim(-.5,4.5)
plt.xlabel('$t$'); plt.ylabel('$x$')
plt.xticks(np.arange(0,5,1))
plt.yticks(np.arange(0,5,1))
if n == 1:
plt.text(1.2,0,'1')
plt.text(1.2,1,'1')
elif n == 2:
plt.text(2.2,0,'1')
plt.text(2.2,1,'2')
plt.text(2.2,2,'1')
elif n == 3:
plt.text(3.2,0,'1')
plt.text(3.2,1,'3')
plt.text(3.2,2,'3')
plt.text(3.2,3,'1')
elif n==4:
plt.text(4.2,0,'1')
plt.text(4.2,1,'4')
plt.text(4.2,2,'6')
plt.text(4.2,3,'4')
plt.text(4.2,4,'1')
```

графическое отображение кода при n = 90



2.6 наиболее вероятная длинна пробега

В такой интерпретации числа траекторий следует полагать, что случаю n = 0 соответствует одна траектория.Биномиальные коэффициенты и, следовательно, найденные числа траекторий можно записать в форме треугольника Паскаля . Рисунок из пособия

Биномиальные коэффициенты, стоящие на пересечении n-ой строки и m-ого столбца треугольника

$$0001$$
$$0011$$
$$0121$$
$$1331$$

Биномиальные коэффициенты, стоящие на пересечении n-ой строки и m-ого столбца треугольника обозначают величиной $C_n{}^m$,Полагая

$$Cnm = (10 \quad \begin{matrix} m = 0 \\ m \neq 0 \end{matrix})$$

получим правило вычисления коэффициентов:

$C_n{}^m$=$C_n{}^{m_+{}^-{}_1{}^1}$+$C_n{}^{m_{-1}}$ и правило вычисления полного числа состояний

$$m{=}n$$

$$Nn = \sum Cnm$$
$$m{=}0$$

для заданного значения n. Не прибегая к аналитическим выкладкам, но на основании численного суммирования элементов строк треугольника Паскаля видно, что полное число состояний

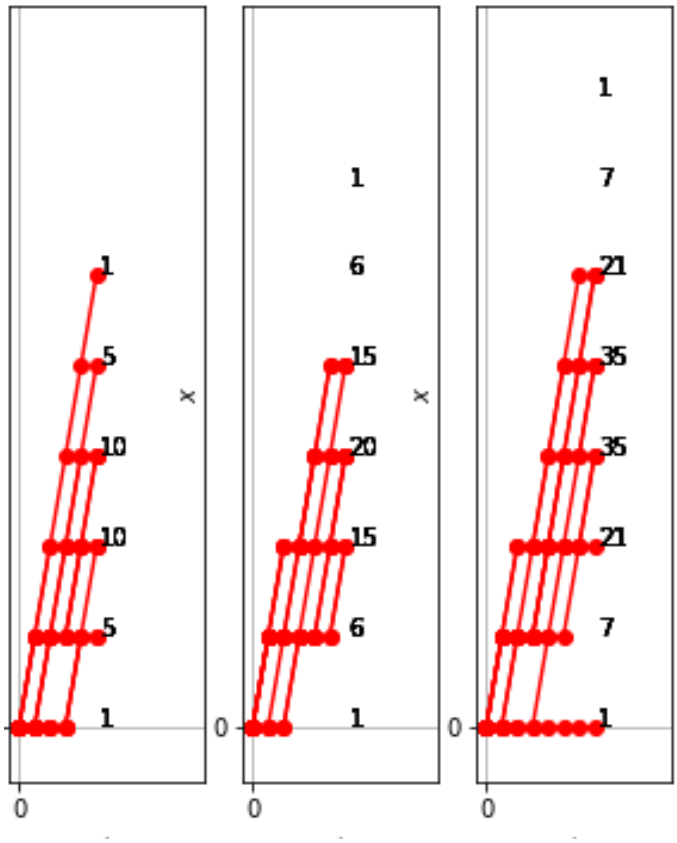
N_n=2^n

Величины биномиальных коффициентов определяются числом комбинаций

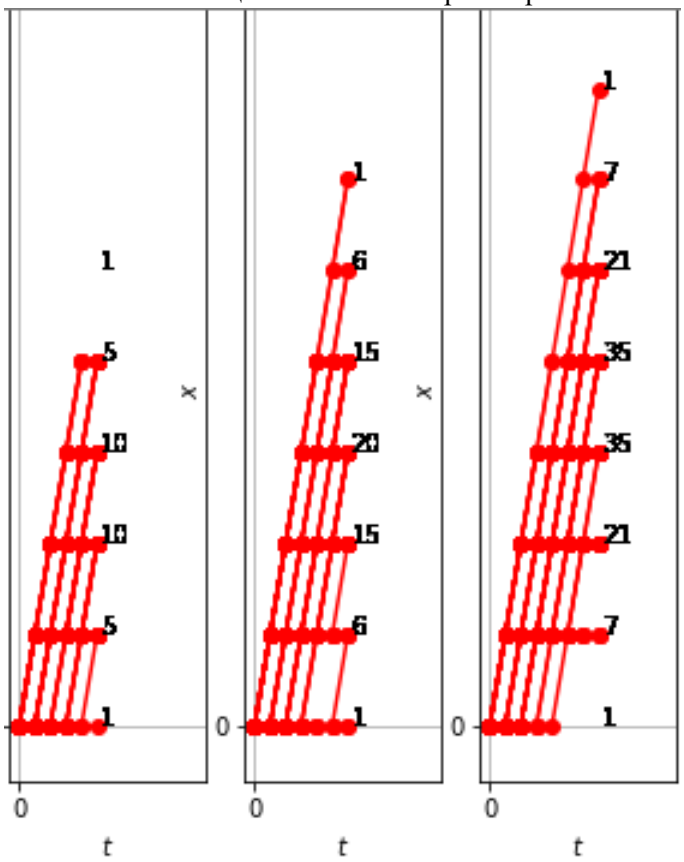
$$C^nm{=}_M\_\_\_\_\_!(_NN-!\ M)_!$$

(Материал взят из пособия)

Реализация N=10 числа траекторий



Реализация N=50 числа траекторий



2.7 Распределение классической вероятности

Задача 10.

Найти вероятности смещения частицы на m шагов за n временных интервалов, основываясь на определении классической вероятности и полагая числа траекторий распределенными по правилу, определяющему биномиальные коэффициенты. Дать графическую иллюстрацию распределениям вероятностей.

$$P_n(\mathrm{m})=\frac{C_n^m}{N}=\frac{n!}{2^n(n-m)!}$$

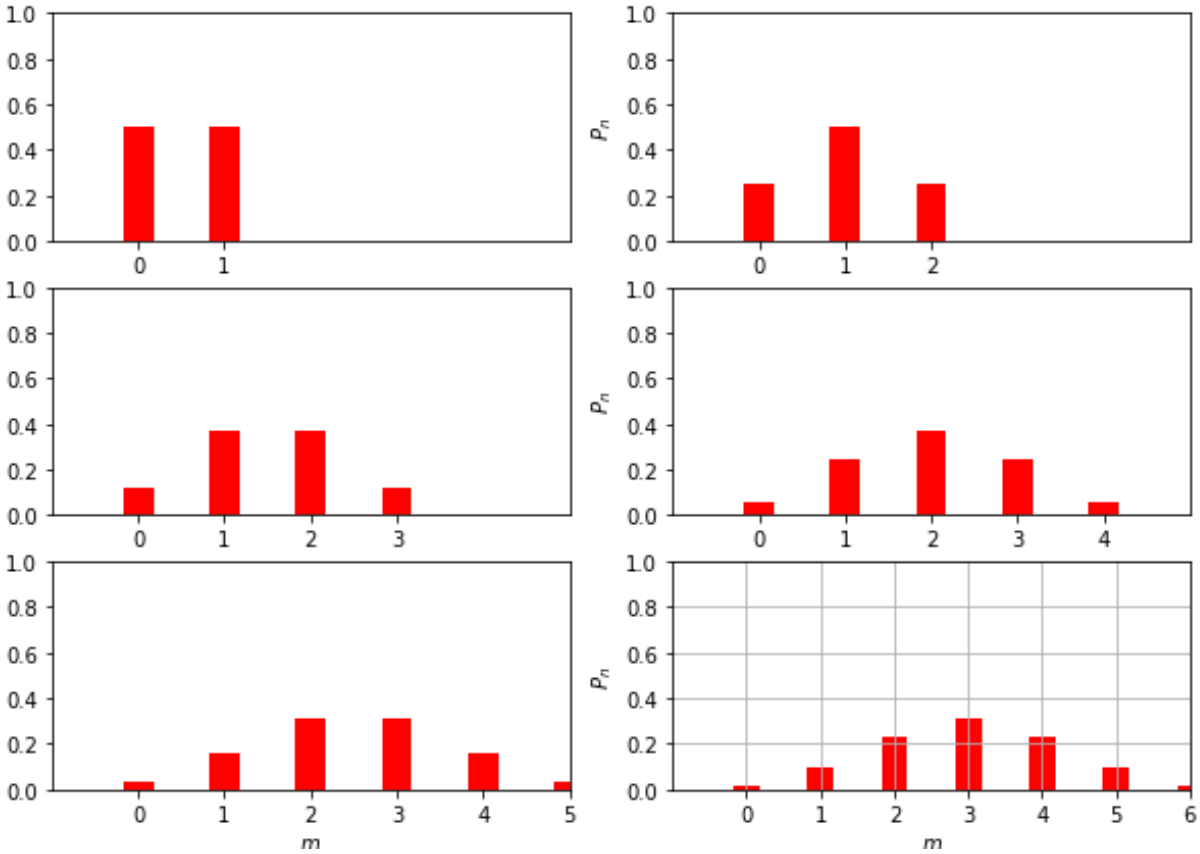
Задача 10. Найти вероятности смещения частицы на m шагов за n временных интервалов, основываясь на определении классической вероятности и полагая числа траекторий распределенными по правилу, определяющему биномиальные коэффициенты. Дать графическую иллюстрацию распределениям вероятностей. (Материал взят из пособия)

Код binopascal.py

```
import numpy as np
import math

def binopascal(n):
  def binomial_coefficient(n, m):
    Cnm = (math.factorial(n)/math.factorial(m)/
            math.factorial((n - m)))
    Cnm = int(Cnm)
    return Cnm

  def pascal(n):
   Cm = np.zeros(n + 1);
   Cm = Cm.astype(int);
   for m in range(n+1):
        Cmn = binomial_coefficient(n, m);
        Cm[m] = np.array(Cmn);
   return Cm

  Cm = pascal(n);
  N = int(np.sum(Cm));
  Pm = Cm/N;
  sumPm = np.sum(Pm);
  return Cm, N, Pm, sumPm

if __name__ == "__main__":
   n = 6
   Cm, N, Pm, sumPm = binopascal(n)
   print('Cm =',Cm)
   print('N =',N) # N = 2**n, % проверка
   print('Pm =',Pm)
   print('sumPm =',sumPm)

   результат консоли ipython:
   Cm = [ 1  6 15 20 15  6  1]
   N = 64
   Pm = [0.015625 0.09375  0.234375 0.3125   0.234375 0.09375  0.015625]
```

```
sumPm = 1.0
```

Графическая иллюстрация распределения вероятности для пробега частицы на n = 1, 2, 3, 4, 5, 6.



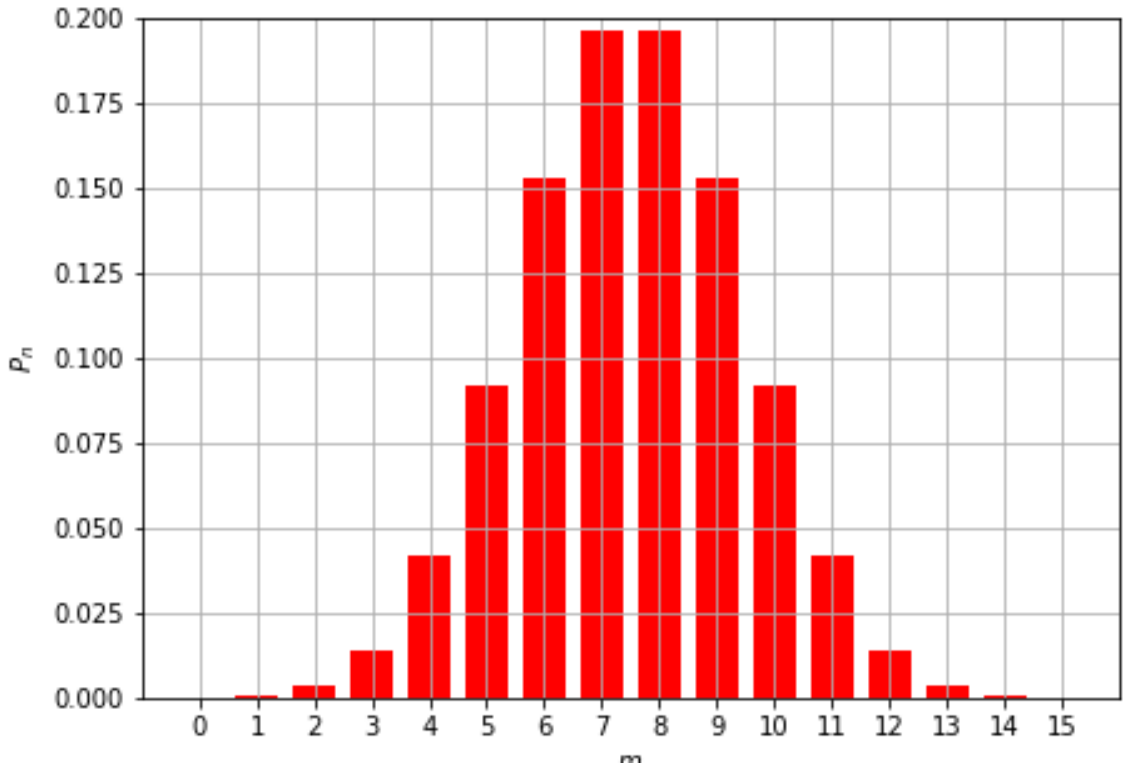
код распределения вероятностей для пробега частицы данных графиков

```
import binopascal as bin
import matplotlib.pyplot as plt
import numpy as np

fig = plt.gcf()
fig.set_size_inches(10,7)

nmax = 6
for n in range(1,nmax + 1):
   Cm, N, Pm, sumPm = bin.binopascal(n);
   plt.subplot(nmax/2,2,n)
   m = np.arange(n + 1)
   width = 0.35
   plt.bar(m, Pm, width, align='center', color='red')
   plt.axis([-1, 5, 0, 1.0])
   plt.xticks(np.arange(0,n + 1,1))
   plt.xlabel('$m$')
   plt.ylabel('$P_n$')
plt.grid()
```

Распределение вероятностей для случая n = 15.

```
import binopascal as bin
import matplotlib.pyplot as plt
import numpy as np

fig = plt.gcf()
fig.set_size_inches(7,5)

n = 15
Cm, N, Pm, sumPm = bin.binopascal(n);
m = np.arange(n + 1)
width = 0.75

plt.bar(m, Pm, width, align='center', color='red')
plt.xticks(np.arange(0,n + 1,1))
plt.ylim(0,0.2)
plt.xlim(-1,n + 1)
plt.xlabel('$m$')
plt.ylabel('$P_n$')
plt.grid()
```

Графическая иллюстрация распределения вероятностей для случия n=15



Этот результат численных экспериментов подтверждает результаты представленные в коде binopascal.py

2.8 Распределение статистической вероятности

Задача 11. В численном статистическом эксперименте найти распределение числа траекторий блуждающей частицы по величинам координат m в конечный момент времени n. Дать графическую иллюстрацию типичным распределениям статистических вероятностей.

```
import matplotlib.pyplot as plt
import numpy as np
import steps

fig = plt.gcf()
fig.set_size_inches(5,5)

n = 4; N = 60
counter = np.zeros(n + 1); counter = counter.astype(np.int32)
for j in range(N):
    x, rnd = steps.xm(n)
    t = np.linspace(0,n,n + 1); t = t.astype(np.int32)
    for i in range(n + 1):
        if x[n] == i:
            counter[i] = counter[i] + 1
print('n =',n,', N =',N)
print('counter =',counter)
m = np.arange(n + 1)
width = 0.5
plt.bar(m, counter/N, width, align='center', color='red')
if n <= 20:
     plt.xticks(np.arange(0,n + 1,1))
     plt.xlim(-.5,n + 0.5);
plt.xlabel('$m$')
plt.ylabel('$P_{stat}$')
plt.grid()
```

вывод консоли Ipython:

```
n = 4 , N = 60
counter = [ 2 13 28 15  2]

n = 8 , N = 120
counter = [ 1  5 17 20 40 28  6  2  1]

n = 16 , N = 240
counter = [ 0  0  0  5  5 12 29 37 49 40 31 26  3  2  1  0  0]

n = 32 , N = 480
counter = [ 0  0  0  0  0  0  0  0  1  4  7 16 26 40 63 62 53 59 56 44 24  9  9  5
  2  0  0  0  0  0  0  0  0]
```

Графики распределения чисел траекторий в консоли

График распределения n = 4 , N = 60

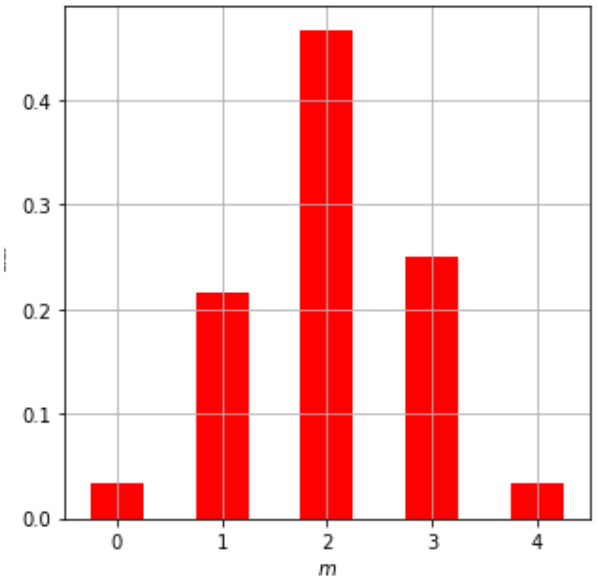


График распределения n = 8 , N = 120

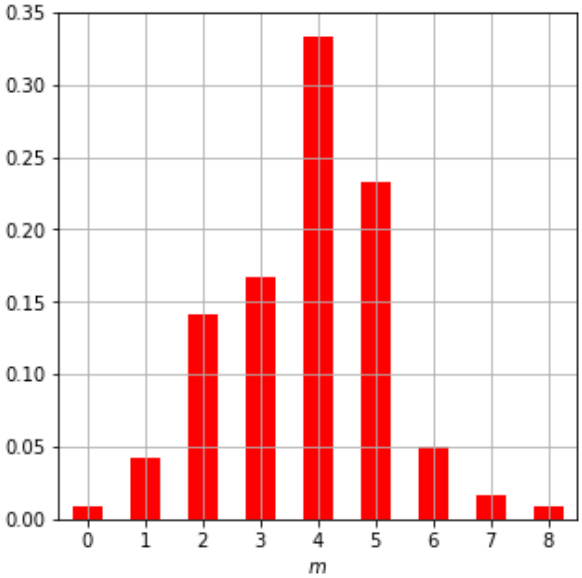


График распределения n = 16 , N = 240

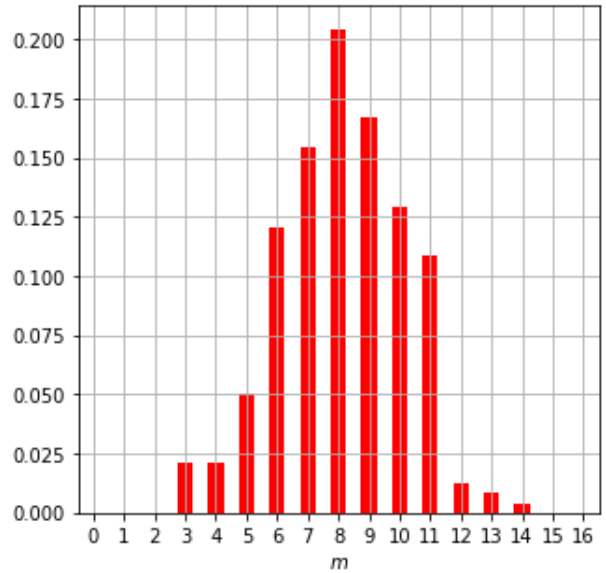
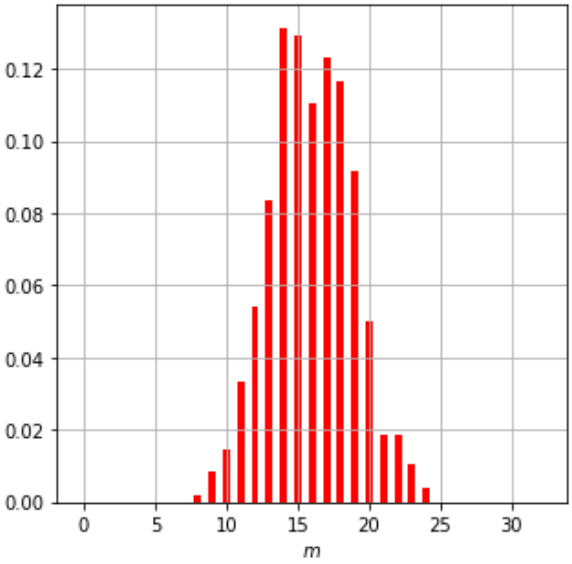


График распределения n = 32 , N = 480



\