Введение в высшую математику

Урок 5. Видеоурок. Элементы теории вероятностей

Задание выполнил Соковнин ИЛ

Практическое задание к уроку 5

1. Напишите код, моделирующий выпадение поля в рулетке(с учетом поля зеро).
2.
   1) Напишитекод,проверяющийлюбуюизтеоремсложения или умножения вероятности на примере рулетки или подбрасывания монетки.
   2) Сгенерируйте десять выборок случайных чисел $x0, \dots, x9$ и постройте гистограмму распределения случайной суммы $+x0+\dots+x9$.
3.
   1) Дополните код Монте-Карло последовательности независимых испытаний расчетом соответствующих вероятностей (через биномиальное распределение) и сравните результаты.
   2) Повторите расчеты биномиальных коэффициентов и вероятностей k успехов в последовательности из n независимых испытаний, взяв другие значения n и k.
4. (не обязательно, но желательно) Из урока по комбинаторике повторите расчеты, сгенерировав возможные варианты перестановок для других значений n и k.
5. (необязательно) Дополните код расчетом коэффициента корреляции x и y по формуле

1. Задание

Напишите код, моделирующий выпадение поля в рулетке(с учетом поля зеро).

В [1]:
```
%matplotlib inline
import numpy as np
```

В [2]:
```
for i in range(0, 5):
    a = input()
    num = np.random.uniform(0, 38)
    if num==0:
        print(f"num={int(num)}: green")
    elif num<19:
         print(f"num={int(num)}: red")
    else:
        print(f"num={int(num)}: black")
```

```
num=8: red

num=25: black

num=16: red

num=14: red

num=0: red
```

2. Задание

2.1 Напишите код, проверяющий любую из теорем сложения или умножения вероятности на примере рулетки или подбрасывания монетки.

У рулетки 37 секторов: 18 красных, 18 чёрных и зеро.

События выпадения 0, чёрного или красного цвета несовместны, поэтому по теореме сложения вероятностей, получаем:

1. Вероятность выпадения 0: P(0)=1/37
2. Вероятность выпадения чёрного: P('black')=18/37 (событие: выпадение любой из 18 чёрных ячеек)
3. Вероятность выпадения красного: P('red')=18/37 (событие: выпадение любой из 18 красных ячеек)

События выпадения 0, чёрного или красного цвета, образуют полную группу несовместных событий. Поэтому:

4. P(0) + P('black') + P('red') = 1

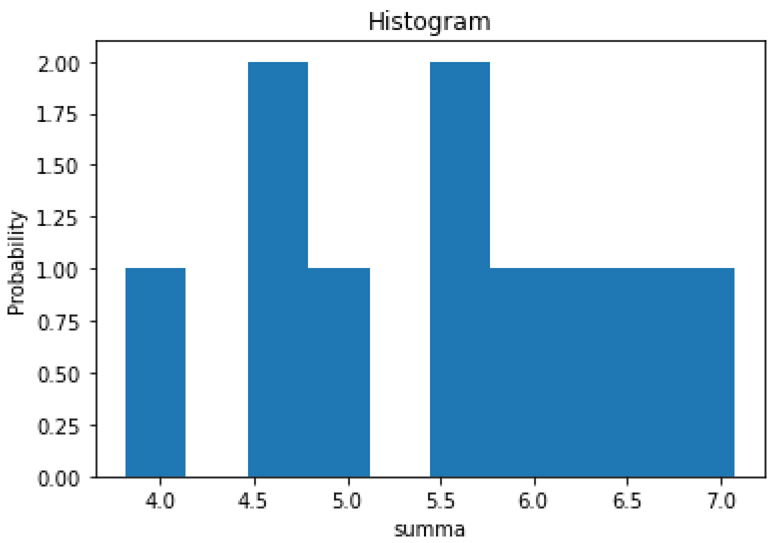
2.2 Сгенерируйте десять выборок случайных чисел x0,...,x9 и постройте гистограмму распределения случайной суммы +x0+...+x9 .

В [3]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
```

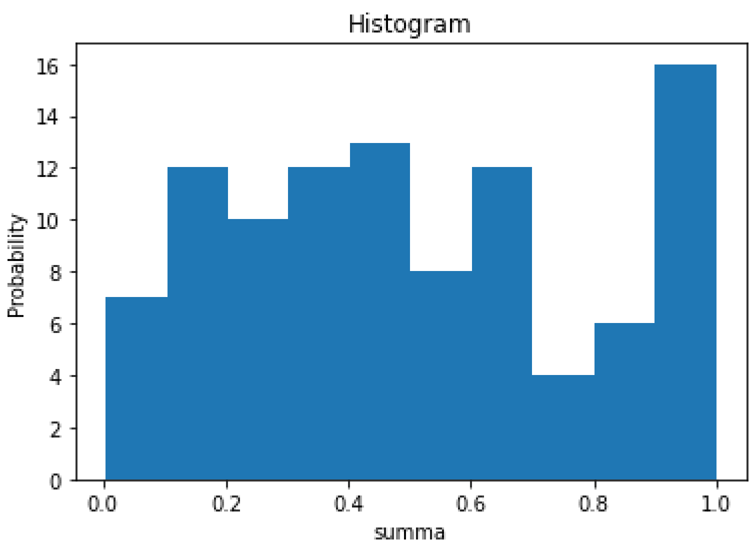
В [4]:
```
def plot_gist(X, num_bins = 10):
    allsums=X

    n, bins, patches = plt.hist(allsums, num_bins)
    plt.xlabel('summa')
    plt.ylabel('Probability')
    plt.title('Histogram')
```

В [5]:
```
# 1.
X = np.random.rand(10)

# Генерация псевдослучайных векторов и нахождение их суммы
for i in range(10):
    x = np.random.rand(10)
    for j in range(10):
        X[j] += x[j]

print('X=', X)

plot_gist(X)
```

```
X= [4.88846925 7.07909244 4.74313357 4.49959394 5.82661612 6.14724309
 5.59856476 6.64173125 3.81002139 5.69084101]
```

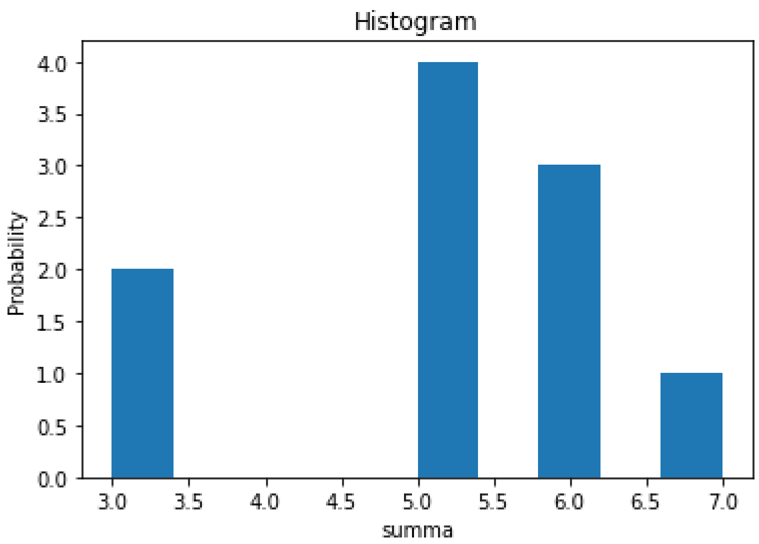


В [6]:
```
# 2.
X = []
for i in range(10):
    x = np.random.rand(10)
    X.extend(x)

# print('X=', X)

plot_gist(X)
```



В [7]:
```
# 3

# Сгенерируйте десять выборок случайных чисел х0,...,х9
n=10

a = np.random.randint(0, 2, n)
b = np.random.randint(0, 2, n)
c = np.random.randint(0, 2, n)
d = np.random.randint(0, 2, n)
e = np.random.randint(0, 2, n)
f = np.random.randint(0, 2, n)
g = np.random.randint(0, 2, n)
h = np.random.randint(0, 2, n)
k = np.random.randint(0, 2, n)
l = np.random.randint(0, 2, n)
X = a + b + c + d + e +f + g + h + k +l
print('X=', X)

# и постройте гистограмму распределения случайной суммы +х0+...+х9 .
plot_gist(X)
```

```
X= [6 6 5 5 5 3 7 5 6 3]
```



В [ ]:

3. Задание

1) Дополните код Монте-Карло последовательности независимых испытаний расчетом соответствующих вероятностей (через биномиальное распределение) и сравните результаты.

2) Повторите расчеты биномиальных коэффициентов и вероятностей k успехов в последовательности из n независимых испытаний, взяв другие значения n и k.

В [8]:
```
import sys
import math
from itertools import product
```

В [9]:
```
# Метод Монте Карло для модели последовательных независимых испытаний
# Методом Монте Карло мы хотим оценить скакой вероятностью при четырёх испытаниях выпадет ровно 2 раза орёл и 2 раза реш
# Посчитать по формуле Бернули вероятность 2-х успехов на 4-х испытаниях и сравнить с тем, что выдаёт метод Монтекарло
```

В [10]:
```
# k - количество успехов
# n - количество испытаний
# k/n - оценка вероятности

k, n = 0, 10000

a = np.random.randint(0, 2, n)
b = np.random.randint(0, 2, n)
c = np.random.randint(0, 2, n)
d = np.random.randint(0, 2, n)
x = a + b + c + d
# оцениваем вероятность выпадения равного количества орлов и решек при 4 испытаниях (2 орла и 2 решки)
for i in range(0, n):
    if x[i] == 2:
        k = k + 1
v=(0.5**k)*(0.5**(n-k))
# print(a, b, c, d)
# print(x)
print(k, n, k/n, v)
# print(k, n, k/n)
```

```
3731 10000 0.3731 0.0
```

В [11]:
```
from itertools import product

for p in product("01",repeat=4):
    print(''.join(p))
```

```
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
```

1)

Дополните код Монте-Карло последовательности независимых испытаний расчетом соответствующих вероятностей (через биномиальное распределение) и сравните результаты.

Расчёт по формуле бернули

k=2 - количество успехов
n=4 - количество испытаний
$c_n^k = \frac{n!}{k!(n-k)!} = \frac{4!}{2!(4-4)!} = 6$
$p = 1/2$ - вероятность наступления успеха в одном испытании
$q = 1 - p = 1/2$ - вероятность неудачи в одном испытании
$P_n(k) = c_n^k p^k q^{n-k} = 6 \cdot \frac{1}{4} \cdot \frac{1}{4} = \frac{3}{8}$

В [12]:
```
import math

def fact(n):
    '''Нахождение факториала числа n!'''

    return math.factorial(n)

def bernuli(n, k, p):
    '''
     Расчёт вероятности по формуле Бернули
     k - количество успехов<br>
     n - количество испытаний
     p - вероятность наступления успеха в одном испытании
    '''

    c_nk = fact(n)/(fact(k)*fact(n-k))
    q=(1-p)
    P_nk=c_nk*(p**k)*(q**(n-k))

    return c_nk, P_nk
```

В [13]:
```
k=2
n=4
p=1/2

print('Расчёт вероятности по формуле Бернули:')
print(f'k={k}, n={n}, p={p}, P_n(k)={bernuli(n, k, p)}')
```

```
Расчёт вероятности по формуле Бернули:
k=2, n=4, p=0.5, P_n(k)=(6.0, 0.375)
```

2)

Повторите расчеты биномиальных коэффициентов и вероятностей k успехов в последовательности из n независимых испытаний, взяв другие значения n и k.

В [14]:
```
# x=list(range(11, 17))
# x
```

В [15]:
```
for n in range(3,8):
    for k in range(n):
        c_nk, P_nk = bernuli(n, k, p)
        print(f'k={k}, n={n}, c_n^k={c_nk}, P_n(k)={P_nk}')
```

```
k=0, n=3, c_n^k=1.0, P_n(k)=0.125
k=1, n=3, c_n^k=3.0, P_n(k)=0.375
k=2, n=3, c_n^k=3.0, P_n(k)=0.375
k=0, n=4, c_n^k=1.0, P_n(k)=0.0625
k=1, n=4, c_n^k=4.0, P_n(k)=0.25
k=2, n=4, c_n^k=6.0, P_n(k)=0.375
k=3, n=4, c_n^k=4.0, P_n(k)=0.25
k=0, n=5, c_n^k=1.0, P_n(k)=0.03125
k=1, n=5, c_n^k=5.0, P_n(k)=0.15625
k=2, n=5, c_n^k=10.0, P_n(k)=0.3125
k=3, n=5, c_n^k=10.0, P_n(k)=0.3125
k=4, n=5, c_n^k=5.0, P_n(k)=0.15625
k=0, n=6, c_n^k=1.0, P_n(k)=0.015625
k=1, n=6, c_n^k=6.0, P_n(k)=0.09375
k=2, n=6, c_n^k=15.0, P_n(k)=0.234375
k=3, n=6, c_n^k=20.0, P_n(k)=0.3125
k=4, n=6, c_n^k=15.0, P_n(k)=0.234375
k=5, n=6, c_n^k=6.0, P_n(k)=0.09375
k=0, n=7, c_n^k=1.0, P_n(k)=0.0078125
k=1, n=7, c_n^k=7.0, P_n(k)=0.0546875
k=2, n=7, c_n^k=21.0, P_n(k)=0.1640625
k=3, n=7, c_n^k=35.0, P_n(k)=0.2734375
k=4, n=7, c_n^k=35.0, P_n(k)=0.2734375
k=5, n=7, c_n^k=21.0, P_n(k)=0.1640625
k=6, n=7, c_n^k=7.0, P_n(k)=0.0546875
```

4. Задание

В [16]:
```
from itertools import product
from itertools import permutations
from itertools import combinations
```

(не обязательно, но желательно) Из урока по комбинаторике повторите расчеты, сгенерировав возможные варианты **перестановок** для других значений n и k .

В [17]:
```
# Число перестановок n=3 обьектов (количество размещений n обьектов из n) (n!)
i = 0
for p in permutations("012", 3):
    print(''.join(str(x) for x in p))
    i += 1

print(f'Количество перестановок n! = {i}')
print(f'Количество перестановок n! = {fact(3)}')
```

```
012
021
102
120
201
210
Количество перестановок n! = 6
Количество перестановок n! = 6
```

В [18]:
```
# Число перестановок n=4 объектов (количество размещений n объектов из n) (n!)
i = 0
for p in permutations("0124", 4):
    # print(''.join(str(x) for x in p))
    i += 1

print(f'Количество перестановок n! = {i}')
print(f'Количество перестановок n! = {fact(4)}')
```

```
Количество перестановок n! = 24
Количество перестановок n! = 24
```

В [19]:
```
# Число перестановок n=6 объектов (количество размещений n объектов из n) (n!)
i = 0
for p in permutations("012345", 6):
    # print(''.join(str(x) for x in p))
    i += 1

print(f'Количество перестановок n! = {i}')
print(f'Количество перестановок n! = {fact(6)}')
```

```
Количество перестановок n! = 720
Количество перестановок n! = 720
```

В [ ]:

В [ ]:

В [ ]:

В [20]:
```
# Число сочетаний 3-х объектов из 2-х
i = 0
for p in product("01",repeat=3):
    print(''.join(p))
    i += 1

print(f'Количество перестановок n! = {i}')
# print(f'Количество перестановок n! = {fact(6)}')
```

```
000
001
010
011
100
101
110
111
Количество перестановок n! = 8
```

В [21]:
```
# Число сочетаний
i = 0
for p in product("012",repeat=3):
    print(''.join(p))
    i += 1

print(f'Количество перестановок n! = {i}')
# print(f'Количество перестановок n! = {fact(6)}')
```

```
000
001
002
010
011
012
020
021
022
100
101
102
110
111
112
120
121
122
200
201
202
210
211
212
220
221
222
Количество перестановок n! = 27
```

В [22]:
```
# Число размещений 4-х объектов из 4-х (количество перестановок) (4!=24)
# for p in permutations("0123", 4):
#     print(''.join(str(x) for x in p))
```

В [23]:
```
# Число размещений 2-х объектов из 4-х (12)
# for p in permutations("0123", 2):
#     print(''.join(str(x) for x in p))

# Число размещений 3-х объектов из 4-х (12)
i = 0
for p in permutations("0123", 3):
    print(''.join(str(x) for x in p))
    i += 1

print(f'Количество перестановок n! = {i}')
# print(f'Количество перестановок n! = {fact(6)}')
```

```
012
013
021
023
031
032
102
103
120
123
130
132
201
203
210
213
230
231
301
302
310
312
320
321
Количество перестановок n! = 24
```

В [24]:
```
# Число сочетаний 2-х объектов из 4-х
# for p in combinations("0123", 2):
#      print(''.join(p))
i = 0

# Число сочетаний 3-х объектов из 4-х
for p in combinations("0123", 3):
    print(''.join(p))
    i += 1

print(f'Количество перестановок n! = {i}')
# print(f'Количество перестановок n! = {fact(6)}')
```

```
012
013
023
123
Количество перестановок n! = 4
```

В [25]:
```
# Число сочетаний 4-х объектов из 2-х
i=0
for p in product("01",repeat=4):
    print(''.join(p))
    i += 1

print(f'Количество перестановок n! = {i}')
# print(f'Количество перестановок n! = {fact(6)}')
```

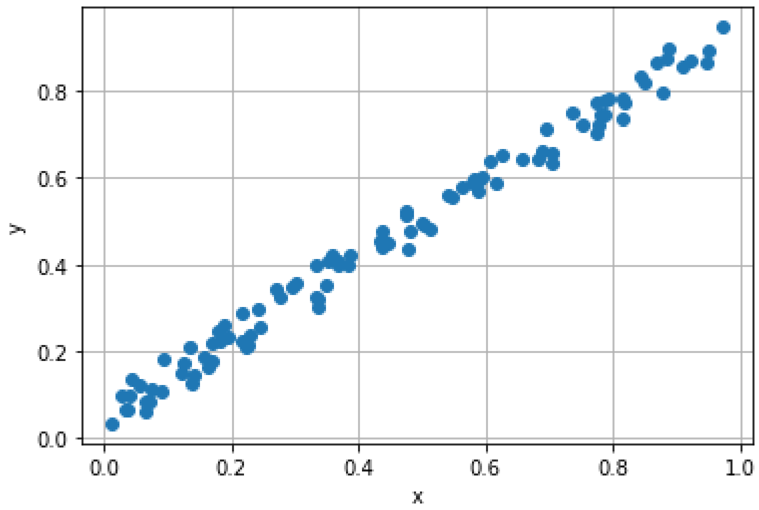
```
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
Количество перестановок n! = 16
```

5. Задание

(необязательно) Дополните код расчетом коэффициента R корреляции x и y по формуле

В [26]:
```
def correlation(x, y):
    """Расчёт коэффициента корреляции """

    x_bar = sum(x) / len(x)
    y_bar = sum(y) / len(y)

    var_x = sum((x_i - x_bar)**2 for x_i in x)
    var_y = sum((y_i - y_bar)**2 for y_i in y)

    assert len(x) == len(y)
    numerator = sum((x_i - x_bar) * (y_i - y_bar) for x_i, y_i in zip(x, y))
    denominator = math.sqrt(var_x * var_y)

    return numerator / denominator
```

В [35]:
```
n = 100
r = 0.9
x = np.random.rand(n)
y = r*x + (1 - r)*np.random.rand(n)
plt.plot(x, y, 'o')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.show()
xm=np.mean(x)
ym=np.mean(y)

# Дополните код расчетом коэффициента корреляции x и y по формуле
R= correlation(x, y)
R1 = np.corrcoef(x, y)[0, 1]

c = np.corrcoef(x, y)  # Матрица коэффициентов корреляции переменных x и y
c2 = np.sum(x)
(np.sum(x)*np.sum(y) - n*np.sum(x*y))/(np.sum(x)*np.sum(x) - n*np.sum(x*x))

print(f'R={R}')
print(f'x_cp={xm}, y_cp={ym}, R={R}, R1={R1}\n\n')
print(c)
```

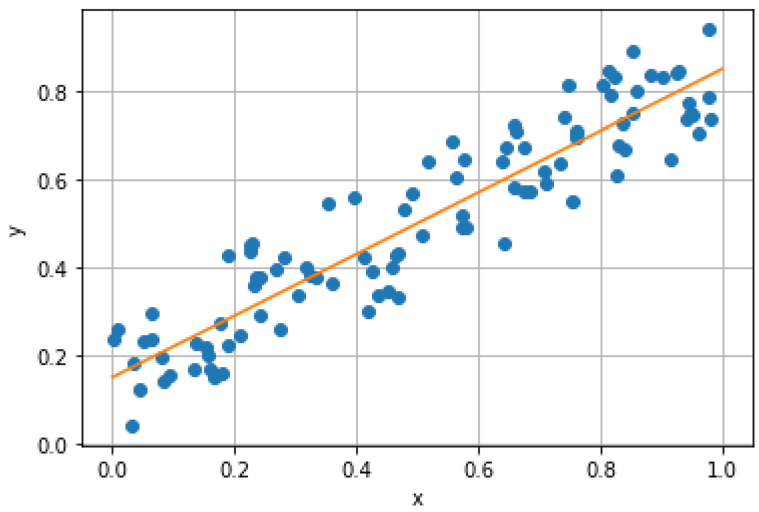



```
R=0.9937196605350629
x_cp=0.4494045108715573, y_cp=0.45739722618120887, R=0.9937196605350629, R1=0.9937196605350632


[[1.         0.99371966]
 [0.99371966 1.        ]]
```

В [ ]:

В [34]:
```
n = 100
r = 0.7
x = np.random.rand(n)
y = r*x + (1 - r)*np.random.rand(n)
plt.plot(x, y, 'o')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)

a = (np.sum(x)*np.sum(y) - n*np.sum(x*y))/(np.sum(x)*np.sum(x) - n*np.sum(x*x))
b = (np.sum(y) - a*np.sum(x))/n
xm=np.mean(x)
ym=np.mean(y)

# Дополните код расчетом коэффициента корреляции x и y по формуле
R= correlation(x, y)
R1 = np.corrcoef(x, y)[0, 1]

# Коэффициенты регрессии a и b, равны коэффициентам a1 и b1 библиотеки numpy
A = np.vstack([x, np.ones(len(x))]).T
a1, b1 = np.linalg.lstsq(A, y)[0]
print(f'x_cp={xm}, y_cp={ym}, R={R}, R1={R1}\n\n')
print(a, b)
print(a1, b1)

plt.plot([0, 1], [b, a + b])
plt.show()

# Прямая проведённая по методу наименьших квадратов, через случайные точки x и y
```

```
<ipython-input-34-f5b5666361ea>:21: FutureWarning:

`rcond` parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input matri
x dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old, explicitly pa
ss `rcond=-1`.


x_cp=0.5010117772430928, y_cp=0.5021378678527383, R=0.9280845669010986, R1=0.9280845669010986


0.6994758417135863 0.15169223325720624
0.6994758417135877 0.15169223325720568
```



В [ ]:

В [ ]:

1. Задание

Напишите код, моделирующий выпадение поля в рулетке(с учетом поля зеро).

Вкратце: что нужно знать. В рулетку можно выиграть в краткосрочной перспективе, но когда играешь долго, ты всегда теряешь деньги. Почему так:

- У рулетки 37 секторов: 18 красных, 18 чёрных и зеро. Самая простая ставка — на красное или на чёрное. Если ставка сработала, то она удваивается. Если не сработала, то ставка сгорает.
- По идее, в половине случаев ставка должна принести прибыль, поэтому играть в эту игру должно быть выгодно. Поставил большую сумму, удвоил, забрал. На первый взгляд, шанс — 50%.
- Но на самом деле красное или чёрное выпадают не в половине случаев, а чуть реже — за счёт зеро. Из-за него вероятность красного или чёрного не 50%, а 48,6%.
- Получается, при идеально честных рулеточных условиях проигрывать мы будем всегда немного чаще, чем выигрывать. И в долгосрочной перспективе мы будем всегда немного терять деньги.

Смоделируем классическую рулетку как в казино с отрицательным матожиданием.

В [29]:
```
# подключаем модуль случайных чисел
import random

# подключаем модуль для графиков
import plotly
import plotly.graph_objs as go

# сколько денег будет на старте для каждой стратегии
startmoney = 1000000

# коэффициент ставки
c1 = 0.001

# количество побед и проигрышей
win = 0
loose = 0

# количество игр, сыгранных по первой стратегии
games = 0

# статистика для первой стратегии
balance1 = []
games1 = []
```

В [30]:
```
# статистика для второй стратегии
balance2 = []
games2 = []

# статистика для третьей стратегии
balance3 = []
games3 = []
```

В [31]:
```
# начинаем играть с полной суммой
# первая стратегия — отрицательное матожидание, как в казино
money = startmoney
# пока у нас ещё есть деньги
while money > 0:
    # ставка —  постоянная часть от первоначальной суммы
    bet = startmoney * c1
    # если ставка получилась больше, чем у нас осталось денег — ставим всё, что осталось, чтобы не уйти в минус
    if bet > money:
        bet = money
    # после ставки количество денег уменьшилось
    money -= bet

    # записываем очередную игру в статистику — деньги и номер игры
    balance1.append(money)
    games1.append(len(games1)+1)

    # крутим рулетку, на которой 18 чёрных чисел, 18 красных и одно зеро. Мы ставим на чёрное
    ball = random.randint(1,37)
    # пусть первые 18 будут чёрными — для простоты алгоритма
    # если наша ставка сыграла — мы попали в нужный диапазон
    if ball in range (1,19):
        # получаем назад нашу ставку в двойном размере
        money += bet * 2
        # увеличиваем количество побед
        win += 1
    else:
        # иначе — увеличиваем количество проигрышей
        loose += 1
games = win + loose
# выводим результат игры по первой стратегии
print("Выиграно ставок: " + str(win) + " (" + str(win/games * 100) + "%). " + " Проиграно ставок: " + str(loose)  + " ("

# началась вторая стратегия, тоже стартуем с полной суммой
# вторая стратегия — с нулевым матожиданием
money = startmoney
# обнуляем статистику
```

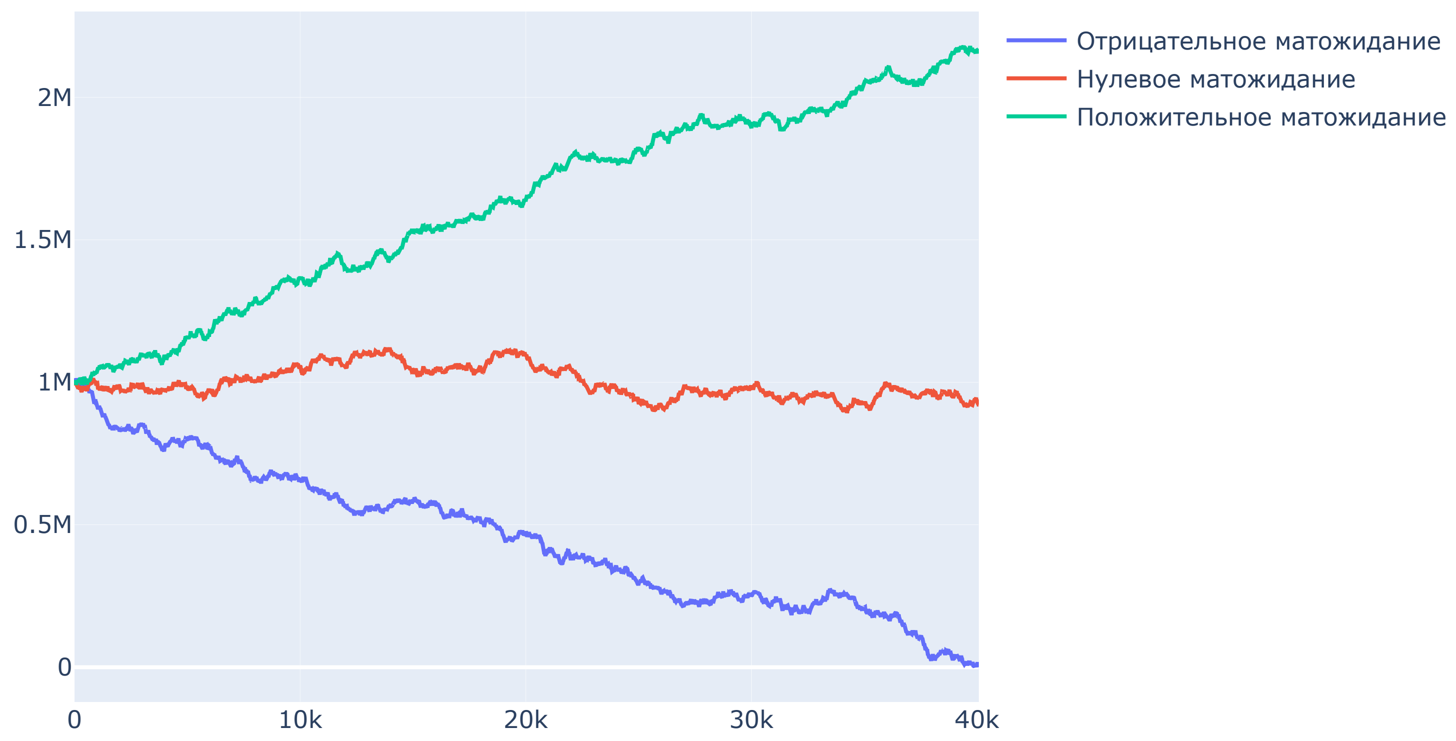
```
Выиграно ставок: 19531 (48.75193450152264%).  Проиграно ставок: 20531 (51.24806549847736%).
```

В [32]:
```
win = 0
loose = 0

# начинаем играть с полной суммой
money = startmoney
# играем, пока есть деньги или пока мы не сыграем столько же игр, как и в первый раз
while (money > 0) and (win + loose < games):
    # ставка –  постоянная часть от первоначальной суммы
    bet = startmoney * c1
    # если ставка получилась больше, чем у нас осталось денег — ставим всё, что осталось, чтобы не уйти в минус
    if bet > money:
        bet = money
    # после ставки количество денег уменьшилось
    money -= bet

    # записываем очередную игру в статистику — деньги и номер игры
    balance2.append(money)
    games2.append(len(games2)+1)

    # крутим рулетку, на которой 18 чёрных чисел, 18 красных. Так как всего поровну, матожидание будет равно нулю.
    # Ставим, как и в прошлом случае, на чёрное
    ball = random.randint(1,36)
    # пусть первые 18 будут чёрными — для простоты алгоритма
    # если наша ставка сыграла — мы попали в нужный диапазон
    if ball in range (1,19):
        # получаем назад нашу ставку в двойном размере
        money += bet * 2
        # увеличиваем количество побед
        win += 1
    else:
        # иначе — увеличиваем количество проигрышей
        loose += 1

# выводим результат игры по второй  стратегии
print("Выиграно ставок: " + str(win) + " (" + str(win/games * 100) + "%). " + " Проиграно ставок: " + str(loose)  + " ("



# началась третья стратегия, тоже стартуем с полной суммой
# третья стратегия — с положительным матожиданием
money = startmoney
# обнуляем статистику
win = 0
loose = 0

# начинаем играть с полной суммой
money = startmoney
# играем, пока есть деньги или пока мы не сыграем столько же игр, как и в первый раз
while (money > 0) and (win + loose < games):
    # ставка –  постоянная часть от первоначальной суммы
    bet = startmoney * c1
    # если ставка получилась больше, чем у нас осталось денег — ставим всё, что осталось, чтобы не уйти в минус
    if bet > money:
        bet = money
    # после ставки количество денег уменьшилось
    money -= bet

    # записываем очередную игру в статистику — деньги и номер игры
    balance3.append(money)
    games3.append(len(games3)+1)

    # крутим рулетку, на которой 18 чёрных чисел, 17 красных. Так как чёрных больше, а мы ставим на чёрное, то матожидани
    # Ставим, как и в прошлом случае, на чёрное
    ball = random.randint(1,35)
    # пусть первые 18 будут чёрными — для простоты алгоритма
    # если наша ставка сыграла — мы попали в нужный диапазон
    if ball in range (1,19):
        # получаем назад нашу ставку в двойном размере
        money += bet * 2
        # увеличиваем количество побед
        win += 1
    else:
        # иначе — увеличиваем количество проигрышей
        loose += 1

# выводим результат игры по третьей стратегии
print("Выиграно ставок: " + str(win) + " (" + str(win/games * 100) + "%). " + " Проиграно ставок: " + str(loose)  + " ("
```

```
Выиграно ставок: 19992 (49.90265089111877%).  Проиграно ставок: 20070 (50.09734910888124%).
Выиграно ставок: 20614 (51.45524437122461%).  Проиграно ставок: 19448 (48.5447556287754%).
```

Строим график

В [33]:
```
# строим графики
fig = go.Figure()
# для первой стратегии
fig.add_trace(go.Scatter(x=games1, y=balance1, name = "Отрицательное матожидание"))
# для второй
fig.add_trace(go.Scatter(x=games2, y=balance2, name = "Нулевое матожидание"))
# и для третьей
fig.add_trace(go.Scatter(x=games3, y=balance3, name = "Положительное матожидание"))
# выводим графики в браузер
fig.show()
```



Моделируем игру в рулетку на Python - [https://thecode.media/plotly/ (https://thecode.media/plotly/)](https://thecode.media/plotly/)

В [ ]:
```
```