Математическое моделирование. Лабораторная работа 1.

Выполнил: Клебанов Д.А, группа 853501, Вариант 10

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sts
```

Задание 1

Написать программу, реализующую метод формирования двумерной случайной величины.

In [2]:
```
np.random.seed = 42
eps = 1e-6

class RVG_2D:
    def __init__(self, P, A, B):
        self.P = P
        self.A = A
        self.B = B

        row_sums = np.sum(P, axis=1)
        self._columns = np.cumsum(P, axis=1) / row_sums.reshape(-1, 1)  # aka l
        self._rows = np.cumsum(row_sums)  # aka q

    def rand(self):
        r_a, r_b = np.random.uniform(size=2)
        a = np.searchsorted(self._rows, r_a)
        b = np.searchsorted(self._columns[a], r_b)
        return self.A[a], self.B[b]
```

Задание 2

Выполнить статистическое исследование полученной величины

Задание 2.1

Построение эмпирической матрицы распределения

In [3]:
```
n, m = 4, 6
np.random.seed = 42
P = np.arange(n * m) / ((n * m) * (n * m - 1) / 2)
np.random.shuffle(P)
P = P.reshape(n, m)
A = np.arange(n)
B = np.arange(m) + n

assert abs(P.sum() - 1) < eps
```

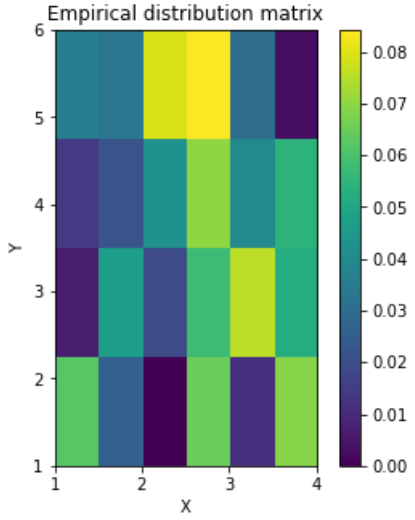
In [4]:
```
A_value_to_key = dict(zip(A, range(len(A))))
B_value_to_key = dict(zip(B, range(len(B))))
print(A_value_to_key, B_value_to_key)

n_trials = int(1e5)
gen = RVG_2D(P, A, B)
P_empirical = np.zeros((n, m))
trials = {'a': [], 'b': []}
for _ in range(n_trials):
    a, b = gen.rand()
    trials['a'].append(a)
    trials['b'].append(b)
    P_empirical[A_value_to_key[a]][B_value_to_key[b]] += 1
P_empirical /= n_trials
trials = {'a': np.array(trials['a']), 'b': np.array(trials['b'])}
print(P_empirical)

assert abs(P_empirical.sum() - 1) < eps
```

```
{0: 0, 1: 1, 2: 2, 3: 3} {4: 0, 5: 1, 6: 2, 7: 3, 8: 4, 9: 5}
[[0.06222 0.02597 0.      0.06509 0.01126 0.0689 ]
 [0.00671 0.04673 0.01953 0.05775 0.07569 0.0521 ]
 [0.01412 0.0214  0.04247 0.0702  0.03953 0.05452]
 [0.03596 0.03303 0.07911 0.08419 0.03002 0.0035 ]]
```

In [5]:
```
plt.figure(figsize=(5, 5))
c = plt.imshow(P_empirical, origin='lower', extent=(1, n, 1, m))
plt.colorbar(c)
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Empirical distribution matrix")
plt.show()
```



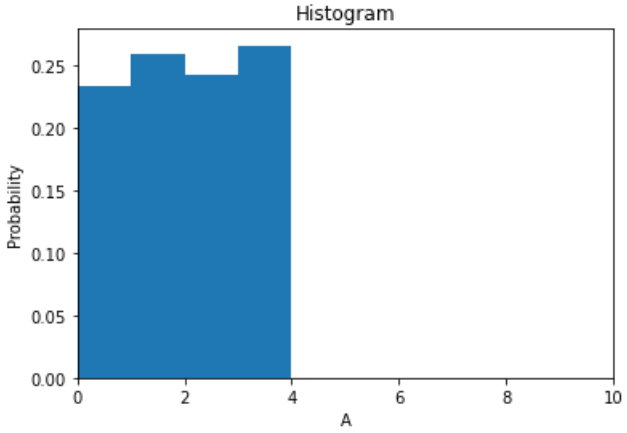
Задание 2.2

Построение гистограммы составляющих вектора

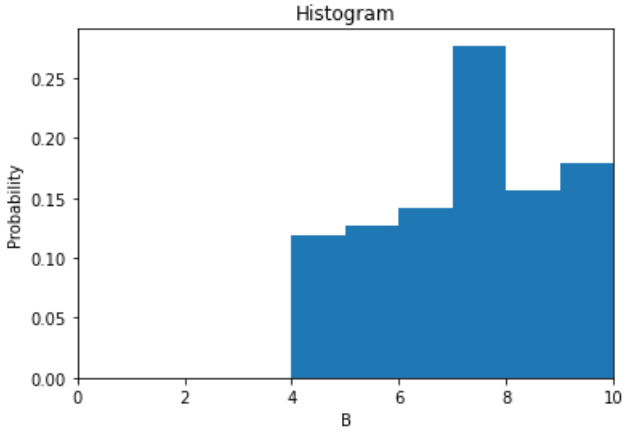
In [6]:
```
def plot_hist(x, weights, name):
    print(name, 'probabilities:', weights)
    plt.hist(x, bins=list(x) + [x[-1] + 1], weights=weights, align='mid', histtype='stepfilled')
    plt.xlim(0, n + m)
    plt.xlabel(name)
    plt.ylabel('Probability')
    plt.title('Histogram')
    plt.show()

plot_hist(A, P_empirical.sum(axis=1), 'A')
plot_hist(B, P_empirical.sum(axis=0), 'B')
```

```
A probabilities: [0.23344 0.25851 0.24224 0.26581]
```




```
B probabilities: [0.11901 0.12713 0.14111 0.27723 0.1565  0.17902]
```



Задание 2.3

Вычисление точечных, интервальных оценок.

In [7]:
```
from functools import partial


var = partial(np.var, ddof=1)
eval_on_trials = lambda f: list(map(f, map(lambda name: trials[name], sorted(trials.keys()))))
print('mean:', eval_on_trials(np.mean))
print('variance:', eval_on_trials(var))
```

```
mean: [1.54042, 6.76214]
variance: [1.2468786923869237, 2.5367479878798793]
```

Возьмем уровень значимости alpha = 0.05

In [8]:
```
def get_confidence_interval_for_mean(samples, alpha=0.05):
    mean = np.mean(samples)
    variance = var(samples)
    diff = np.sqrt(variance / len(samples)) * sts.norm.ppf(1 - alpha / 2)
    return mean - diff, mean + diff


def print_confidence_interval(interval, name, value_name):
    print(name, ':', interval[0], '<=', value_name, '<=', interval[1])


print('Confidence interval for mean (alpha=0.05):')
for name in ('a', 'b'):
    print_confidence_interval(get_confidence_interval_for_mean(trials[name]), name, 'mean')
```

```
Confidence interval for mean (alpha=0.05):
a : 1.5334991379502223 <= mean <= 1.5473408620497775
b : 6.752268418093375 <= mean <= 6.772011581906624
```

In [9]:
```
def get_confidence_interval_for_variance(samples, alpha=0.05):
    n = len(samples)
    numerator = (n - 1) * np.sqrt(var(samples))
    return (
        numerator / sts.chi2.ppf(1 - alpha / 2, n - 1),
        numerator / sts.chi2.ppf(alpha / 2, n - 1),
    )


print('Confidence interval for variance (alpha=0.95):')
for name in ('a', 'b'):
    print_confidence_interval(get_confidence_interval_for_variance(trials[name]), name, 'variance')
```

```
Confidence interval for variance (alpha=0.95):
a : 1.1069138754348609 <= variance <= 1.1264898552547198
b : 1.5788482571018965 <= variance <= 1.6067704851139932
```

Задание 2.4

Вычисление коэффициент корреляции

In [10]:
```
def get_correlation_coefficient(trials, names):
    means = eval_on_trials(np.mean)
    variances = eval_on_trials(var)
    return (trials['a'] - means[0]) @ (trials['b'] - means[1]) / len(trials['a'] - 1) / np.sqrt(varia

get_correlation_coefficient(trials, ('a', 'b'))
```

Out[10]:
```
-0.10853926294473097
```

Задание 3

Проверить гипотезы о соответствии полученных оценок характеристик случайной величины требуемым.

Проверим используя критерий хи-квадрат.

In [11]:
```
def test_chi_square(alpha=0.05):
    statistics = P.size * np.sum(np.nan_to_num((P - P_empirical) ** 2 / P))
    p_value = 1 - sts.chi2.cdf(statistics, P.size - 1)
    return statistics, p_value, p_value > alpha


test_chi_square()
```

```
D:\Programs\miniconda3\envs\ds37\lib\site-packages\ipykernel_launcher.py:2: RuntimeWarning: invalid
value encountered in true_divide
```

Out[11]:
```
(0.009661201789042529, 1.0, True)
```