# Лабрраторная работа 3

## Акулич Кирилл 853504 (2 вариант)

$$f(x,y) = \frac{1}{2} * sin(x+y) \forall x,y : 0 \leq x,y \leq \frac{\pi}{2}$$

$$f(x) = \int_0^{\frac{\pi}{2}} \frac{1}{2} * sin(x+y)dy f(x) = \frac{1}{2} * (sin(x) + cos(x)) = \frac{\sqrt{2}}{2} * cos(x - \frac{\pi}{4})$$

Функцию распределения СВ x:

$$F(x) = \int_0^x f(x)dx = \int_0^z \frac{\sqrt{2}}{2} * cos(x - \frac{\pi}{4})dx = \frac{\sqrt{2}}{2} * (sin(x - \pi/4) + \frac{\sqrt{2}}{2})$$

Найдем функцию, обратную к данной

$$x = arcsin(\frac{2}{\sqrt{2}} * R_1 - \frac{1}{\sqrt{2}}) + \pi/4$$

Найдем условную функцию для y

$$f(y|x = x_1) = \frac{f(x_1, y)}{f(x_1)}$$

$$F(y|x = x_1) = \int_0^{\frac{\pi}{2}} \frac{1}{2} * \frac{sin(x_1 + y)}{f(x_1)}$$

$$F(y|x = x_1) = \frac{1}{2 * (f(x_1))} * (-cos(x_1 + y) - (-cos(x_1)))$$

$$cos(x_1) - 2 * f(x_1) * R_2 = cos(x_1 + y)$$

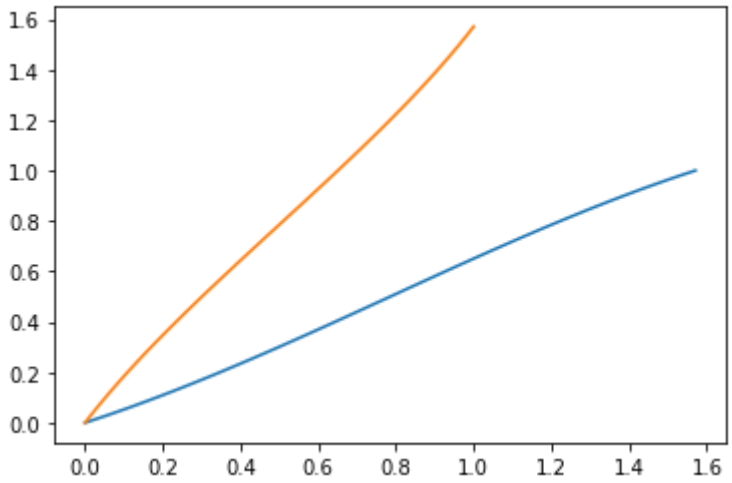
$$y = arccos(cos(x_1) - 2 * f(x_1) * R_2) - x_1$$

In [ ]:

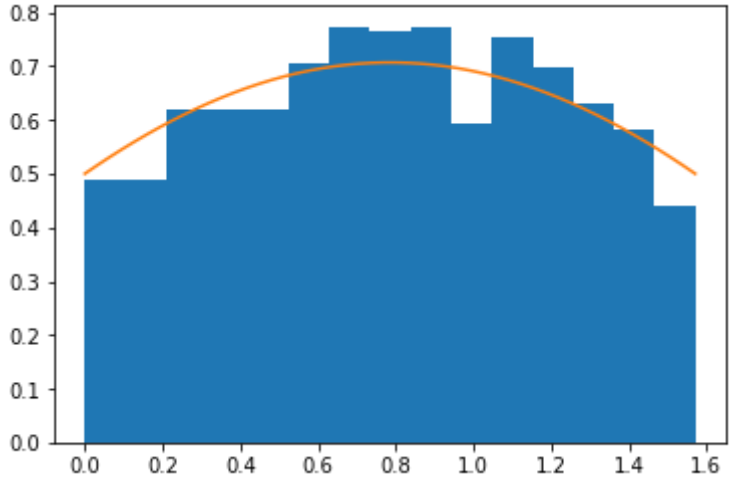
```
import numpy as np
import math
import random
import matplotlib.pyplot as plt

SQRT_2 = 2**(0.5)
SQRT_2_INV = 1 / SQRT_2

def f(x,y):
    return 0.5*np.sin(x+y)
def f_x(x):
    return SQRT_2_INV*np.cos(x-np.pi/4)
def F_X(x):
    return SQRT_2_INV*(math.sin(x-np.pi/4) + SQRT_2_INV)
def get_x(R1):
    return np.arcsin(SQRT_2*R1 - SQRT_2_INV) + np.pi/4

def get_y(R2,x1):
```

```
    return np.arccos(np.cos(x1) - 2* f_x(x1)*R2) - x1
def F_Y(y,x1):
    return 1/(2*f_x(x1))*(-np.cos(x1+y) +np.cos(x1))
def f_y(y,x1):
    return f(x1,y)/f_x(x1)
def vector_generator_2d():
    while True:
        R1 = random.random()
        R2 = random.random()
        x = get_x(R1)
        y = get_y(R2,x)
        yield (x,y)

def y_generator(x1):
    while True:
        R2 = random.random()
        y = get_y(R2,x1)
        yield y

x_y_points =list()
vector_2d_generator = vector_generator_2d()
for _ in range(1000):
    x,y = next(vector_2d_generator)
    x_y_points.append((x,y))
print("Проверим правильность нахождения обратных функций F(x) and inv(F(x))")
x_ls = np.linspace(0,np.pi/2)
_x = np.linspace(0,1)
plt.plot(x_ls,[F_X(x) for x in x_ls])
plt.plot(_x,[get_x(x) for x in _x])

plt.show()
print("Гистограмма X ")
plt.hist([vector[0] for vector in x_y_points],bins=15,density=True)
plt.plot(x_ls,[f_x(x) for x in x_ls])
plt.show()

x1 = 0.5
print(f"Проверим правильность нахождения обратных функцийц F(y) and inv(F(y))")
y_ls = np.linspace(0,np.pi/2)
_y = np.linspace(0,1)

y_gen = y_generator(x1)
y_points = [next(y_gen) for _ in range(1000)]

plt.plot(y_ls,[F_Y(y,x1) for y in y_ls])
plt.plot(_y,[get_y(y,x1) for y in _y])
plt.show()

plt.hist([y for y in y_points],bins=15,density=True)
plt.plot(y_ls,[f_y(y,x1) for y in y_ls])
plt.show()
```

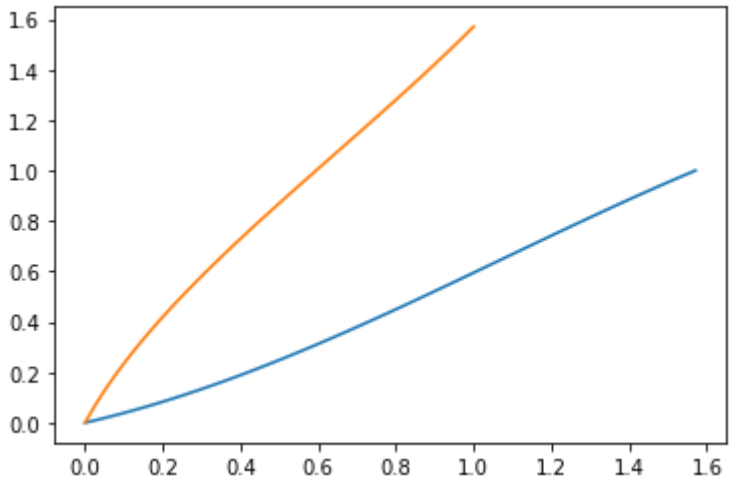
```
Проверим правильность нахождения обратных функций F(x) and inv(F(x))
```

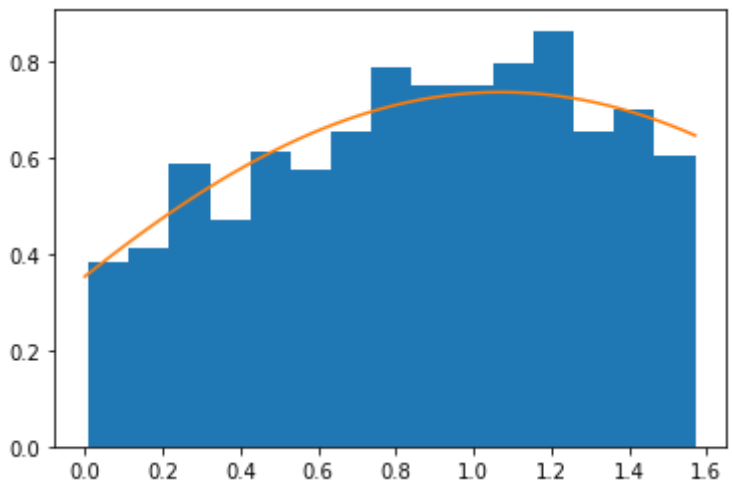



Гистограмма X




Проверим правильность нахождения обратных функцийц F(y) and inv(F(y)) x1:0.5



# Оценки случайной величины

## Теоретические:

### M(x),M(y):

Внутренняя неопределённая часть:

Сначала вычисляем внутренний неопределённый интеграл

$$\int \frac{x}{2}\sin(x+y)\,dy$$

=

$$-\frac{x\cos(x+y)}{2}$$

Внутренняя определённая часть:

Подставляем пределы интегрирования
y от 0 до pi/2

$$\int_0^{\frac{\pi}{2}} \frac{x}{2}\sin(x+y)\,dy$$

=

$$\frac{x\sin(x)}{2}+\frac{x\cos(x)}{2}$$

Внешняя неопределённая часть:

Потом вычисляем внешний интеграл

$$\int\left(\frac{x\sin(x)}{2}+\frac{x\cos(x)}{2}\right)dx$$

=

$$\frac{x\sin(x)}{2}-\frac{x\cos(x)}{2}+\frac{\sin(x)}{2}+\frac{\cos(x)}{2}$$

Внешняя определённая часть:

Подставляем пределы интегрирования
x от 0 до pi/2

$$\int_0^{\frac{\pi}{2}}\left(\frac{x\sin(x)}{2}+\frac{x\cos(x)}{2}\right)dx$$

=

$$\frac{\pi}{4}$$

### D(x),D(y):

Внутренняя неопределённая часть:

$$\int \frac{x^2}{2} \sin(x+y)\, dy$$

=

$$-\frac{x^2 \cos(x+y)}{2}$$

Внутренняя определённая часть:

y от 0 до pi/2

$$\int_0^{\frac{\pi}{2}} \frac{x^2}{2} \sin(x+y)\, dy$$

=

$$\frac{x^2 \sin(x)}{2} + \frac{x^2 \cos(x)}{2}$$

Внешняя неопределённая часть:

Потом вычисляем внешний интеграл

$$\int \left( \frac{x^2 \sin(x)}{2} + \frac{x^2 \cos(x)}{2} \right) dx$$

=

$$\frac{x^2 \sin(x)}{2} - \frac{x^2 \cos(x)}{2} + x \sin(x) + x \cos(x) - \sin(x) + \cos(x)$$

Внешняя определённая часть:

Подставляем пределы интегрирования
x от 0 до pi/2

$$\int_0^{\frac{\pi}{2}} \left( \frac{x^2 \sin(x)}{2} + \frac{x^2 \cos(x)}{2} \right) dx$$

=

$$-2 + \frac{\pi^2}{8} + \frac{\pi}{2}$$

## cor(x,y)

$$M(x,y) = -1 - \frac{\pi^2}{16} + \frac{\pi}{2} cor = -0.24$$

$$M(x), M(y) = \frac{\pi}{4}$$

$$D(x), D(y) = -2 + \frac{\pi}{4} + \frac{\pi}{16}$$

In [ ]:
```
print(f"Точечная оценка M(x) = {np.mean([vector[0] for vector in x_y_points])
print(f"Точечная оценка M(y) = {np.mean([vector[1] for vector in x_y_points])


print(f"Точечная оценка D(x) = {np.var([vector[0] for vector in x_y_points])}
print(f"Точечная оценка D(y) = {np.var([vector[1] for vector in x_y_points])}

print(f"Точечная оценка Corr(x,y): {np.corrcoef([vector[0] for vector in x_y_
```

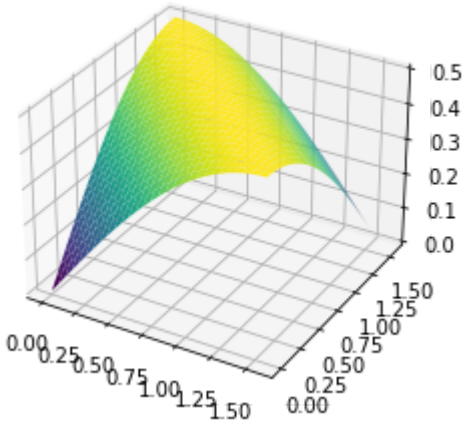
```
Точечная оценка M(x) = 0.7945878908159317
Точечная оценка M(y) = 0.7958963335452479
Точечная оценка D(x) = 0.1782249245128265
Точечная оценка D(y) = 0.18100243939276695
Точечная оценка Corr(x,y): -0.25868580784625805
```

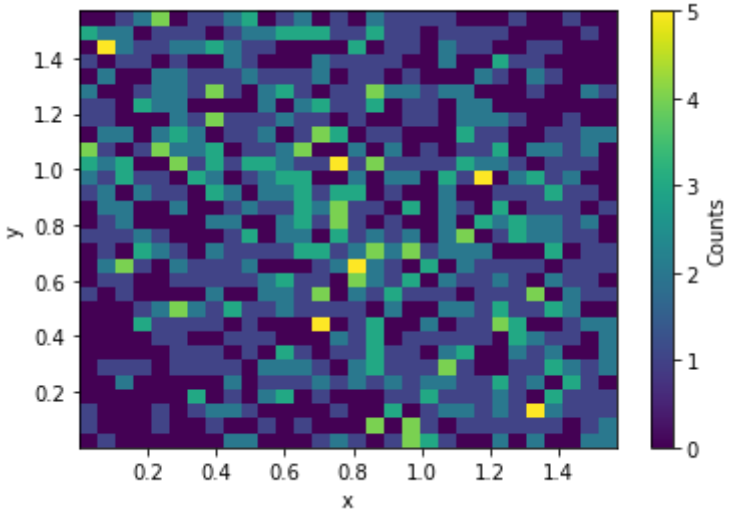
Интервальные оценки

In [ ]:
```
import scipy.stats as st

def m_confidence_interval(m_x, d_x, N, confidence_level = 0.95):
    normal_quantil = st.norm.ppf(confidence_level)
    return [m_x - np.sqrt(d_x / N) * normal_quantil, m_x + np.sqrt(d_x / N) *

def d_confidence_interval(d_x, N, confidence_level = 0.95):
    xi_plus, xi_minus = st.chi2.ppf((1-confidence_level)/2,N-1), st.chi2.ppf(
    return ((N - 1) * d_x / xi_minus, (N - 1) * d_x / xi_plus)


def arth_p(r):
    return np.log((1 + r) / (1 - r)) / 2.0

def th_z(z):
    e = math.exp(2 * z)
    return((e - 1) / (e + 1))

def r_confidence_interval(r, n, alpha):
    z = arth_p(r)
    se = 1.0 / math.sqrt(n - 3)
    z_crit = st.norm.ppf(1 - alpha/2)

    lo = z - z_crit * se
    hi = z + z_crit * se

    return (th_z(lo), th_z(hi))

M_X = np.mean([vector[0] for vector in x_y_points])
D_X = np.var([vector[0] for vector in x_y_points])
print(f"Интервальная оенка M(x) {m_confidence_interval(M_X,D_X,len(x_y_points
print(f"Интервальая оценка D(x) {d_confidence_interval(D_X,len(x_y_points))}"

M_Y =  np.mean([vector[1] for vector in x_y_points])
D_Y = np.var([vector[1] for vector in x_y_points])
print(f"Интервальная оценка M(Y) {m_confidence_interval(M_Y,D_Y,len(x_y_point
print(f"Интервальная оценка D(Y) {d_confidence_interval(D_Y,len(x_y_points))}

print(f"Иниервальная оценка Corr(x,y) {r_confidence_interval(np.corrcoef([vec
```

```
Интервальная оенка M(x) [0.7726289455345133, 0.8165468360973501]
Интервальая оценка D(x) (0.1635726366141294, 0.19494854371046808)
Интервальная оценка M(Y) [0.7737669421149712, 0.8180257249755246]
Интервальная оценка D(Y) (0.16612180549939462, 0.1979866848821288)
Иниервальная оценка Corr(x,y) (-0.3066037981451339, -0.2094599157419281)
```

In [ ]:
```
'''
fig = plt.figure()
ax = plt.axes(projection="3d")

z_points = [1/2]
x_points = np.cos(z_points) + 0.1 * np.random.randn(100)
y_points = np.sin(z_points) + 0.1 * np.random.randn(100)
ax.scatter3D(x_points, y_points, z_points, c=z_points, cmap='hsv');

plt.show()
'''

x = np.linspace(0, np.pi/2, 30)
y = np.linspace(0, np.pi/2, 30)

X, Y = np.meshgrid(x, y)
Z = 0.5*np.sin(X + Y)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none')
plt.show()

fig2 = plt.figure()
N,x_bins,y_bins,_ = plt.hist2d([vector[0] for vector in x_y_points], [vector[
plt.xlabel('x')
plt.ylabel('y')
cbar = plt.colorbar()
cbar.ax.set_ylabel('Counts')
plt.show()
```



In [ ]:

Теоретические значения СВ: M(X):

$$M(x) = \int_0^( \pi/2) f(x) = \pi/4 D(x) = M[X^2] - (M[X])^2 = \pi^2/(4 * \sqrt(2)) + \pi/\sqrt(2) - 4$$

# Дискретная СВ

In [ ]:

```
import numpy as np
def check_if_independent(_matrix):
    matrix = np.array(_matrix)
    x_rows_count,y_columns_count = np.shape(matrix)
    x_p = [sum(x_row) for x_row in matrix]
    y_p = [sum(matrix[:,i]) for i in range(y_columns_count)]
    for i in range(x_rows_count):
        for j in range(y_columns_count):
            if np.abs(x_p[i]*y_p[j] - matrix[i,j]) > 10e-10 :
                return False
    return True

def find_dependent_x(_matrix):
    matrix = np.array(_matrix)
    _,y_columns_count = np.shape(matrix)
    y_p = [sum(matrix[:,i]) for i in range(y_columns_count)]
    dependent_dist = []
    for y_col_indx in range(y_columns_count):
        dependent_dist.append(matrix[:,y_col_indx]/y_p[y_col_indx])
    return np.transpose(dependent_dist)

def find_dependent_y(_matrix):
    matrix = np.array(_matrix)
    x_rows_count,y_columns_count = np.shape(matrix)
    x_p = [sum(x_row) for x_row in matrix]
    dependent_dist = []
    for x_row_indx in range(x_rows_count):
        dependent_dist.append(matrix[x_row_indx]/x_p[x_row_indx])
    return np.array(dependent_dist)


x_p = [0.2, 0.3, 0.05, 0.1, 0.05, 0.1, 0.2]
y_p = [0.3, 0.2, 0.1, 0.05, 0.15, 0.2]
matrix = [[i * j for i in y_p] for j in x_p]

print(check_if_independent(matrix))
print("Условные матрицы распределения")
print("X|Y")
print(find_dependent_x(matrix))
print("Условные матрицы распределения")
print("Y|X")
print(find_dependent_y(matrix))
```

```
True
Условные матрицы распределения
X|Y
[[0.2  0.2  0.2  0.2  0.2  0.2 ]
 [0.3  0.3  0.3  0.3  0.3  0.3 ]
 [0.05 0.05 0.05 0.05 0.05 0.05]
```

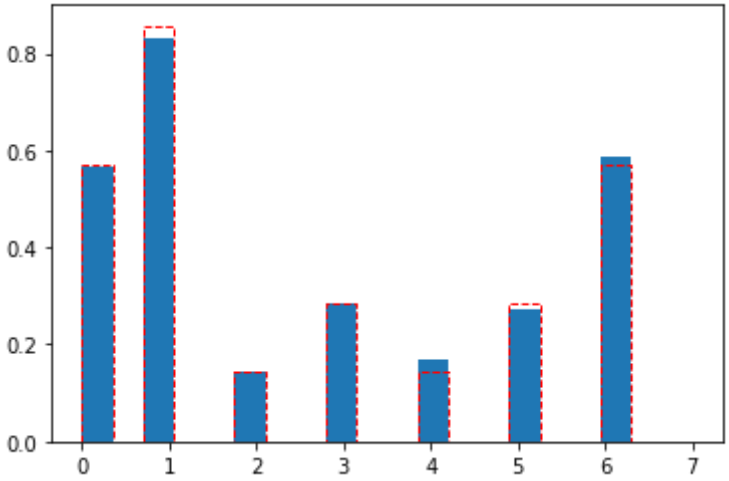
```
 [0.1  0.1  0.1  0.1  0.1  0.1 ]
 [0.05 0.05 0.05 0.05 0.05 0.05]
 [0.1  0.1  0.1  0.1  0.1  0.1 ]
 [0.2  0.2  0.2  0.2  0.2  0.2 ]]
Условные матрицы распределения
Y|X
[[0.3  0.2  0.1  0.05 0.15 0.2 ]
 [0.3  0.2  0.1  0.05 0.15 0.2 ]
 [0.3  0.2  0.1  0.05 0.15 0.2 ]
 [0.3  0.2  0.1  0.05 0.15 0.2 ]
 [0.3  0.2  0.1  0.05 0.15 0.2 ]
 [0.3  0.2  0.1  0.05 0.15 0.2 ]
 [0.3  0.2  0.1  0.05 0.15 0.2 ]]
```

In [ ]:

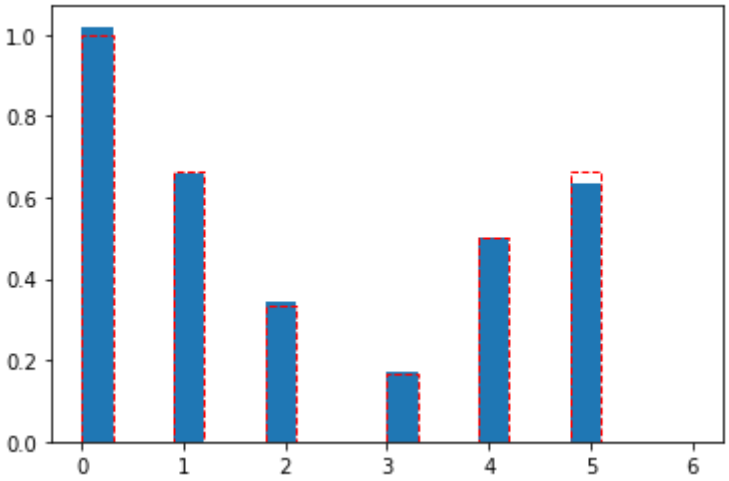
```
import random
import bisect
def discrete_vector_generator(_matrix):
    def _first(it,condition):
        return next(x for x in it if condition(x))
    def lot_method(p):
        _p = random.random()
        _cumsum = enumerate(np.cumsum(p))
        success_event = _first(_cumsum, lambda x : _p <=x[1])[0]
        return success_event

    matrix = np.array(_matrix)
    x_p = [sum(x_row) for x_row in matrix]
    y_dep = find_dependent_y(_matrix)
    print(x_p)
    while True:
        x_indx = lot_method(x_p)
        y_indx = lot_method(y_dep[x_indx])
        yield x_indx,y_indx

def get_p_x(_matrix):
    matrix = np.array(_matrix)
    x_p = [sum(x_row) for x_row in matrix]
    return x_p

def get_p_y(_matrix):
    matrix = np.array(_matrix)
    _,y_columns_count = np.shape(matrix)
    y_p = [sum(matrix[:,i]) for i in range(y_columns_count)]
    return y_p

gen = discrete_vector_generator(matrix)
seq = []

for i in range(5000):
    seq.append(next(gen))

x_seq = [s[0] for s in seq]
y_seq = [s[1] for s in seq]
p = [random.gauss(4,2) for _ in range(400)]
p_x = get_p_x(matrix)
print("X seq")
plt.hist(x_seq, bins=20, range=(0, len(p_x)), density=True)
plt.hist(np.arange(0, len(p_x)), bins=20, range=(0, len(p_x)),
                                        density=True, weights=p_x, histtype='
                                        facecolor='none', edgecolor='red', li
plt.show()
#plt.hist(get_p_x(matrix),bins = range(len(x_seq))  ,alpha=0.5, label='theore

print("Y seq")
```

```
p_y = get_p_y(matrix)
plt.hist(y_seq, bins=20, range=(0, len(p_y)), density=True)
plt.hist(np.arange(0, len(p_y)), bins=20, range=(0, len(p_y)),
                                        density=True, weights=p_y, histtype='
                                        facecolor='none', edgecolor='red', li
plt.show()
```

```
[0.2, 0.3, 0.05, 0.1, 0.05, 0.1, 0.2]
X seq
```




```
Y seq
```




In [ ]:

```
def get_theory_mean(p_values):
    return sum([i  * p for i, p in enumerate(p_values)])

def get_theory_d(p_values):
    exp_val = get_theory_mean(p_values)
    return sum([(i - exp_val) ** 2 * p for i, p in enumerate(p_values)])

mx, my = get_theory_mean(p_x), get_theory_mean(p_y)
dx, dy = get_theory_d(p_x), get_theory_d(p_y)

print(f"Теоретические значения математического ожидания: M(x)={mx} ,M(Y) ={my
print(f"Теоретические значения дисперсии: D(X)={dx} ,D(Y) ={dy}")

print(f"Точечная оценка МО: M(x) = {np.mean(x_seq)}")
print(f"Точечная оценка МО: M(y) = {np.mean(y_seq)}")


print(f"Точечная оценка дисперсии: D(x) = {np.var(x_seq)}")
print(f"Точечная оценка дисперсии: D(y) = {np.var(y_seq)}")

print(f"Коэффициент кореляции: Corr(x,y): {np.corrcoef(x_seq,y_seq)[0][1]}")
```

```
Теоретические значения математического ожидания: M(x)=2.6000000000000005 ,M
(Y) =2.1500000000000004
Теоретические значения дисперсии: D(X)=5.140000000000001 ,D(Y) =3.82750000000
00006
Точечная оценка МО: M(x) = 2.641
Точечная оценка МО: M(y) = 2.1126
Точечная оценка дисперсии: D(x) = 5.161319000000001
Точечная оценка дисперсии: D(y) = 3.7763212399999997
Коэффициент кореляции: Corr(x,y): 0.0016814658291630642
```

In [ ]:

```
M_X = np.mean([vector[0] for vector in seq])
D_X = np.var([vector[0] for vector in seq])
print(f"Интервальная оценка МО: M(x) {m_confidence_interval(M_X,D_X,len(seq))
print(f"Интервальнаяя оценка дисперсии: D(x) interval {d_confidence_interval(

M_Y =  np.mean([vector[1] for vector in seq])
D_Y = np.var([vector[1] for vector in seq])
print(f"Интервальная оценка МО: M(Y) {m_confidence_interval(M_Y,D_Y,len(seq))
print(f"Интервальнаяя оценка дисперсии: D(Y) interval {d_confidence_interval(

print(f"Интервальная оценка коэффициента корреляции: Corr interval {r_confide
```

```
Интервальная оценка МО: M(x) [2.5881527241289612, 2.693847275871039]
Интервальнаяя оценка дисперсии: D(x) interval (4.964808167767712, 5.369789802
5807875)
Интервальная оценка МО: M(Y) [2.0673960151945794, 2.1578039848054207]
Интервальнаяя оценка дисперсии: D(Y) interval (3.6325424831262496, 3.92885060
6951679)
Интервальная оценка коэффициента корреляции: Corr interval (-0.02158390465917
5767, 0.02494501617564394)
```