Приложение 1

Программа для python для расчёта первого обратного момента распределения Обобщенная геометрическая случайная величина

```
import numpy as np
import scipy
import matplotlib.pyplot as plt
import texttable

lamda = np.linspace(0, 1, 10)[:-1]
A_values = np.arange(1, 4)

def funct_hypergeo(A_values, lamda):
    moment = []
    for A in A_values:
        a = A + 1
        b = 2
        c = A + 2
        moment_for_A = [scipy.special.hyp2f1(a, b, c, l) for l in lamda]
        moment.append(moment_for_A)
    return moment
def inverse_hypergeo(lamda, A_values, func_geo):
    inverse_results = []
    for A, moment_for_A in zip(A_values, func_geo):
        inverse_result_for_A = ((((1 - l) ** 2) / (A + 1)) * moment for l, moment
in zip(lamda, moment_for_A)]
        inverse_results.append(inverse_result_for_A)
    return inverse_results

# Вычислить обратную функцию для каждого значения
first_inverse_moment = inverse_hypergeo(lamda, A_values, func_geo)


tableau = [[ " A "," Параметр Лямбды ", "Обратный момент E(1 / X + A)"]]

plt.figure()
for interger, ivs in zip(A_values, first_inverse_moment):
    tableau.append(["A = {0}".format(interger), "", ""])
    for l, iv in zip(lamda, ivs):
        tableau.append(["", l, iv])
    plt.plot(lamda, ivs, 'o-', label=f'A={interger}')

plt.legend()
plt.xlabel('lambda')
plt.ylabel('Обратный гипергеометрический момент\n E(1/X + A)')
plt.title('Обратные гипергеометрические моменты для различных A')
plt.show()
```

```
# Распечатайте таблицу результатов
results_table = texttable.Texttable()
results_table.add_rows(tableau)
print(results_table.draw())
```

Приложение 2

Программа на python для вычисления биномиального распределения случайной величины с первым обратным моментом

```
import numpy as np
from scipy.integrate import quad
import matplotlib.pyplot as plt
import texttable

p = 0.5
n_values = np.arange(50, 250, 8)
n_values
def first_integrat(x, p, n):
    return (p * np.exp(-x) + 1 - p) ** n

def moment_Bin(n, p):
    result, error = quad(first_integrat, 0, np.inf, args=(p, n))
    return result

# Вычислите обратные биномиальные моменты для каждого n
inv_binomial = np.array([moment_Bin(n, p) for n in n_values])

#print("Reverse moment: \n", inv_binomial)

# Создание таблицы для отображения результатов
table = texttable.Texttable()
table.add_rows([['n', 'Обратный момент E(1/ X + 1)']] + list(zip(n_values,
inv_binomial)))
print(table.draw())

# Построение графика результатов
plt.plot(n_values, inv_binomial, 'o-', label='Обратный момент')
plt.xlabel('n')
plt.ylabel('Обратный биномиальный момент')
plt.title('Обратный биномиальный момент  при n')
plt.legend()
plt.show()
```

```

p = 0.5
A_values = np.arange(1, 5)

def integrate(x, A, p, n):
    return np.exp(-A * x) * (p * np.exp(-x) + 1 - p) ** n

def inverse_moment(A, p, n):
    result, error = quad(integrate, 0, np.inf, args=(A, p, n))
    return result
```

```
# Создайте матрицу, содержащую обратные моменты для каждой комбинации A и n
moments = np.zeros((len(A_values), len(n_values)))

# Вычислите моменты
for i, A in enumerate(A_values):
    for j, n in enumerate(n_values):
        moments[i, j] = inverse_moment(A, p, n)

# Отображение результатов в таблице
table = texttable.Texttable()
table.set_max_width(0)
rows = [['n\\A'] + list(A_values)]
for j, n in enumerate(n_values):
    rows.append([n] + list(moments[ : ,j]))
table.add_rows(rows)
print(table.draw())

# Построение графика результатов
fig, ax = plt.subplots(figsize=(8, 6))

for i, A in enumerate(A_values):
    ax.plot(n_values, moments[i, :], 'o-', label=f'A = {A}')

ax.set_xlabel('n')
ax.set_ylabel('Inverse Moments')
ax.set_title('Обратные моменты для различных значений A и n')
ax.legend()

plt.show()
```

Приложение 3

Программа на python для вычисления обратной момент для распределения Пуассона случайной величины.

```
import numpy as np
from scipy.integrate import quad
import matplotlib.pyplot as plt
import texttable

lamb = np.random.choice(np.arange(1,30), 20, replace=False)
lamb.sort()
lamb

def integrand(x, l):
    return np.exp(l * (np.exp(-x) - 1))

# Определить функцию moment_Poiss для вычисления интеграла
def moment_Poiss(l):
    result, error = quad(integrand, 0, np.inf, args=(l))
    return result

inv_poissona = np.array([moment_Poiss(l) for l in lamb])



# Создание таблицы для отображения результатов
table = texttable.Texttable()
table.add_rows([['lambda', 'Обратный момент E(1/ X)']] + list(zip(lamb,
inv_poissona)))
print(table.draw())

# Построение графика результатов
plt.plot(lamb, inv_poissona, 'o-', label='Обратный момент')
plt.xlabel('lambda')
plt.ylabel('Обратный Пуассон момент')
plt.title('Обратный Пуассон момент  при lambda')
plt.legend()
plt.show()
```

```
A_values = np.arange(1, 5)


def integrand(x,A , l):
    return np.exp(-A * x) * np.exp(l * (np.exp(-x) - 1))

# Определить функцию moment_Poiss для вычисления интеграла
def moment_Poiss(A, l):
    result, error = quad(integrand, 0, np.inf, args=(A , l))
```

```
    return result

moments = np.zeros((len(A_values), len(lamb)))


# Вычислите моменты
for i, A in enumerate(A_values):
    for j, l in enumerate(lamb):
        moments[i, j] = moment_Poiss(A, l)

# Отображение результатов в таблице
table = texttable.Texttable()
table.set_max_width(0)
rows = [['lambda\\A'] + list(A_values)]
for j, l in enumerate(lamb):
    rows.append([l] + list(moments[ :,j]))
table.add_rows(rows)
print(table.draw())

# Построение графика результатов
fig, ax = plt.subplots(figsize=(8, 6))

for i, A in enumerate(A_values):
    ax.plot(lamb, moments[i, :], 'o-', label=f'A = {A}')

ax.set_xlabel('lambda')
ax.set_ylabel('Обратный  момент')
ax.set_title('Обратные моменты для различных значений A и lambda')
ax.legend()

plt.show()
```

Приложение 4

Программа на python для вычисления обратного момента для гамма-распределения случайной величины

```
import numpy as np
from scipy.integrate import quad
import matplotlib.pyplot as plt
import texttable


alph_values = np.random.choice(np.arange(2, 20), 15, replace=False)
alph_values.sort()
bet_values = [2, 3, 4]


def integrate(x, a, b):
    return  ((1 + x / b) ** (-a))

def inverse_moment( a, b):
    result, error = quad(integrate, 0, np.inf, args=( a, b))
    return result

# Вычисление первого момента для каждой пары (альфа, бета)
first_moment = [[inverse_moment(a, b) for b in bet_values] for a in alph_values]


table = texttable.Texttable()
table.set_max_width(0)
rows = [['alpha \\ beta'] + bet_values]

# Заполнение строк Альфа-значениями и вычисленными моментами
rows = [['alpha \\ beta'] + [str(b) for b in bet_values]]

for i, a in enumerate(alph_values):
    row = [a] + [first_moment[i][j] for j in range(len(bet_values))]
    rows.append(row)
table.add_rows(rows)
print(table.draw())

# Построить график для каждой Альфы
first_moment_array = np.array(first_moment)  # Преобразовать список в массив
numpy для индексации
for j, b in enumerate(bet_values):
    plt.plot(alph_values, first_moment_array[:, j], marker='o', linestyle='-',
label=f'beta = {b}')

plt.xlabel('Альфа-значения')
plt.ylabel('обратный момент E(1/X)')
plt.title(f'обратный момент E(1/X) по альфа при различных бета')
plt.legend()
```

```
plt.grid(True)
plt.show()
```

```
A_values = 1


def integrate(x, A, a, b):
    return np.exp(-A * x) * ((1 + x / b) ** (-a))

def inverse_moment(A, a, b):
    result, error = quad(integrate, 0, np.inf, args=(A, a, b))
    return result

# Вычисление первого момента для каждой пары (альфа, бета)
first_moments = [[inverse_moment(A_values, a, b) for b in bet_values] for a in
alph_values]


plt.figure(figsize=(8,6))
table = texttable.Texttable()
table.set_max_width(0)
rows = [['alpha \\ beta'] + bet_values]

# Заполнение строк Альфа-значениями и вычисленными моментами
rows = [['alpha \\ beta'] + [str(b) for b in bet_values]]

for i, a in enumerate(alph_values):
    row = [a] + [first_moments[i][j] for j in range(len(bet_values))]
    rows.append(row)
table.add_rows(rows)
print(table.draw())

# Построить график для каждой Альфы
first_moment_array = np.array(first_moment)  # Преобразовать список в массив
numpy для индексации
for j, b in enumerate(bet_values):
    plt.plot(alph_values, first_moment_array[:, j], marker='o', linestyle='-',
label=f'beta = {b}')

plt.xlabel('Альфа-значения')
plt.ylabel('обратный момент E(1/X + 1)')
plt.title(f'обратный момент E(1/X + A) по альфа при различных бета и A = 1')
plt.legend()
plt.grid(True)
plt.show()
```