10.16 11.13 12.16 13.5 14.12 15.1

10.16) Определить производящую функцию распределения Пуассона

,

где m принимает целые неотрицательные значения.

Воспользуемся функцией производящей функции вероятности.

Моделирование:

import random  
import math  
import matplotlib.pyplot as plt  
  
# Функция для вычисления производящей функции распределения Пуассона  
def poisson\_generating\_function(a, u):  
 return math.exp(a \* (u - 1))  
  
# Функция для генерации случайных чисел по распределению Пуассона  
def generate\_poisson\_random\_variable(a, n):  
 random\_numbers = []  
 for \_ in range(n):  
 L = math.exp(-a)  
 p = 1.0  
 m = 0  
 while p > L:  
 p \*= random.random()  
 m += 1  
 random\_numbers.append(m - 1)  
 return random\_numbers  
  
# Параметры  
a = 5 # параметр распределения Пуассона  
n = 10000 # количество испытаний  
u = 0.5 # значение u для производящей функции  
  
# Генерация случайных чисел  
random\_numbers = generate\_poisson\_random\_variable(a, n)  
  
# Подсчет частоты появления каждого значения m  
frequency = {}  
for num in random\_numbers:  
 if num in frequency:  
 frequency[num] += 1  
 else:  
 frequency[num] = 1  
  
# Нормализация частот для получения эмпирической производящей функции  
empirical\_gf = sum([(a \*\* m / math.factorial(m) \* math.exp(-a)) \* (u \*\* m) for m in frequency])  
  
# Вывод результатов  
print(f"Теоретическая производящая функция G(u) = {poisson\_generating\_function(a, u)}")  
print(f"Эмпирическая производящая функция G(u) = {empirical\_gf}")  
  
# График для визуализации распределения  
plt.hist(random\_numbers, bins=30, density=True, alpha=0.6, color='g', label='Эмпирическое распределение')  
x = range(0, 20)  
y = [a \*\* m / math.factorial(m) \* math.exp(-a) for m in x]  
plt.plot(x, y, 'r--', label='Теоретическое распределение')  
plt.title('Распределение Пуассона')  
plt.xlabel('Значение m')  
plt.ylabel('Плотность вероятности')  
plt.legend()  
plt.show()

Этот код использует метод обратного преобразования для генерации случайных чисел по распределению Пуассона и сравнивает эмпирическую производящую функцию с теоретической.

Результат:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

11.13) Вероятность того, что отказ прибора произойдет при числе неработоспособных элементов X = m равна:

а) для прибора A

;

б) для прибора B

Найти математическое ожидание числа неработоспособных элементов, приводящих к отказам каждого из приборов

Найдём мат. ожидания по определению.

Моделирование:

import random  
import math  
  
# Функции для вычисления вероятностей отказа приборов A и B  
def probability\_A(a, m):  
 return 1 - math.exp(-a \* m)  
  
def probability\_B(a, m):  
 if m == 0:  
 return 0  
 else:  
 return 1 - math.exp(-a \* (m - 1))  
  
# Функции для моделирования числа неработоспособных элементов  
def simulate\_A(a, n):  
 count = 0  
 for \_ in range(n):  
 m = 0  
 while random.random() > probability\_A(a, m):  
 m += 1  
 count += m  
 return count / n  
  
def simulate\_B(a, n):  
 count = 0  
 for \_ in range(n):  
 m = 0  
 while random.random() > probability\_B(a, m):  
 m += 1  
 count += m  
 return count / n  
  
# Параметры  
a = 3.75 # параметр a  
n = 100000 # количество испытаний  
  
# Моделирование  
M\_A\_simulated = simulate\_A(a, n)  
M\_B\_simulated = simulate\_B(a, n)  
  
# Аналитическое решение  
M\_A\_analytic = 1 / (1 - math.exp(-a))  
M\_B\_analytic = 1 / (1 - math.exp(-a)) + 1  
  
# Вывод результатов  
print(f"Моделирование для прибора A: M(m) = {M\_A\_simulated}")  
print(f"Аналитическое решение для прибора A: M(m) = {M\_A\_analytic}")  
print(f"Моделирование для прибора B: M(m) = {M\_B\_simulated}")  
print(f"Аналитическое решение для прибора B: M(m) = {M\_B\_analytic}")

Этот код включает в себя функции для вычисления вероятностей отказа приборов A и B, функции для моделирования числа неработоспособных элементов для каждого прибора, а также вычисление аналитических решений для математических ожиданий.

Код генерирует случайные числа, используя метод обратного преобразования, и сравнивает результаты моделирования с теоретическими значениями математических ожиданий.

Результат:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

12.16) Найти функцию распределения длины хорды, проведенной через точку пересечения окружности с ее диаметром единичной длины, под углом к нему равномерно распределенным в интервале [0; π/2 ].

Изображение выглядит как круг, зарисовка, линия, диаграмма

Автоматически созданное описание

Пусть Х – случайная величина, представляющая длину хорды, требуется найти функцию распределения F(x). Длина хорды связана с углом a как x = cos(a).

Угол а распределён равномерно в интервале , значение cos(a) в этом интервале убывает при убывании значения а. Следовательно

Теперь выразим F(x) как вероятность того, что угол а >= arccos(x):

Так как угол равномерно распределён в интервале , вероятность того, что а меньше, чем arccos(x) равна отношению длины дуги, содержащей углы меньше arccos(x), к полной длине дуги. Таким образом:

*Моделирование:*

import random  
import math  
import matplotlib.pyplot as plt  
  
# Аналитическое решение  
def analytical\_solution(x):  
 return 1 - (2 / math.pi) \* math.acos(x)  
  
# Генерация случайных углов и вычисление длин хорд  
def simulate\_chord\_lengths(num\_samples):  
 angles = [random.uniform(0, math.pi / 2) for \_ in range(num\_samples)]  
 chord\_lengths = [math.cos(angle) for angle in angles]  
 return chord\_lengths  
  
# Моделирование распределения длин хорд  
num\_samples = 100000  
chord\_lengths = simulate\_chord\_lengths(num\_samples)  
  
# Вычисление эмпирической функции распределения  
empirical\_cdf = []  
sorted\_lengths = sorted(chord\_lengths)  
for x in range(len(sorted\_lengths)):  
 empirical\_cdf.append(x / num\_samples)  
  
# Вычисление аналитической функции распределения  
analytical\_cdf = [analytical\_solution(x) for x in sorted\_lengths]  
  
# Построение графиков  
plt.figure(figsize=(10, 6))  
plt.plot(sorted\_lengths, empirical\_cdf, label='Empirical CDF')  
plt.plot(sorted\_lengths, analytical\_cdf, label='Analytical CDF', linestyle='--')  
plt.xlabel('Chord Length')  
plt.ylabel('Cumulative Probability')  
plt.legend()  
plt.title('Comparison of Empirical and Analytical CDFs')  
plt.show()

Этот код генерирует случайные углы, вычисляет длины хорд, строит эмпирическую функцию распределения (ECDF) и сравнивает ее с аналитическим решением. График показывает, как эмпирическая функция распределения соотносится с аналитической функцией распределения. Эмпирическая функция распределения близка к аналитической функции распределения.

Изображение выглядит как текст, снимок экрана, линия, График

Автоматически созданное описание

Для наглядного сравнения значений эмпирической и аналитической функций распределения, мы можем вывести несколько значений для различных длин хорд. Давайте модифицируем предыдущий код, чтобы он выводил значения для нескольких выбранных длин хорд.

import random  
import math  
  
# Аналитическое решение  
def analytical\_solution(x):  
 return 1 - (2 / math.pi) \* math.acos(x)  
  
# Генерация случайных углов и вычисление длин хорд  
def simulate\_chord\_lengths(num\_samples):  
 angles = [random.uniform(0, math.pi / 2) for \_ in range(num\_samples)]  
 chord\_lengths = [math.cos(angle) for angle in angles]  
 return chord\_lengths  
  
# Моделирование распределения длин хорд  
num\_samples = 100000  
chord\_lengths = simulate\_chord\_lengths(num\_samples)  
  
# Вычисление эмпирической функции распределения  
sorted\_lengths = sorted(chord\_lengths)  
empirical\_cdf = [i / num\_samples for i in range(num\_samples)]  
  
# Вычисление аналитической функции распределения  
analytical\_cdf = [analytical\_solution(x) for x in sorted\_lengths]  
  
# Вывод значений для нескольких длин хорд  
selected\_lengths = [0.1, 0.3, 0.5, 0.7, 0.9]  
for length in selected\_lengths:  
 index = next((i for i, x in enumerate(sorted\_lengths) if x >= length), None)  
 if index is not None:  
 empirical\_value = empirical\_cdf[index]  
 analytical\_value = analytical\_solution(length)  
 print(f"Chord Length: {length:.2f}, Empirical CDF: {empirical\_value:.4f}, Analytical CDF: {analytical\_value:.4f}")  
 else:  
 print(f"Chord Length: {length:.2f}, not found in simulation data")

Этот код выводит значения эмпирической и аналитической функций распределения для выбранных длин хорд. Если длина хорды не найдена в моделируемых данных, код сообщит об этом.

Результат:

Изображение выглядит как текст, программное обеспечение, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

13.5) Плотность вероятности случайных амплитуд A боковой качки корабля определяется формулой (закон Рэлея)

*,*

где— дисперсия угла крена.

Одинаково ли часто встречаются амплитуды, меньшие и большие ее математического ожидания?

Пусть случайная величина - значение амплитуды боковой качки, функция распределения этой случайной величины принимает вид:

Вычислим математическое ожидание случайной величины :

Отсюда найдём вероятность события:

;

Следовательно, ;

import random  
import math  
  
# Функция для вычисления плотности вероятности по закону Рэлея  
def Rayleigh\_pdf(a, sigma):  
 return (a / (sigma \*\* 2)) \* math.exp(-(a \*\* 2) / (2 \* (sigma \*\* 2)))  
  
# Функция для генерации случайных чисел по закону Рэлея  
def generate\_Rayleigh\_random\_variable(sigma, n):  
 random\_numbers = []  
 for \_ in range(n):  
 u = random.random()  
 a = sigma \* math.sqrt(-2 \* math.log(1 - u))  
 random\_numbers.append(a)  
 return random\_numbers  
  
# Параметры  
sigma = 1 # параметр σ  
n = 100000 # количество испытаний  
  
# Генерация случайных чисел  
random\_numbers = generate\_Rayleigh\_random\_variable(sigma, n)  
  
# Вычисление математического ожидания по закону Рэлея  
a\_prime = math.sqrt(math.pi / 2) \* sigma  
  
# Подсчет частоты амплитуд меньше и больше математического ожидания  
less\_than\_a\_prime = sum(1 for a in random\_numbers if a < a\_prime)  
greater\_than\_a\_prime = n - less\_than\_a\_prime  
  
# Вычисление вероятностей  
P\_less\_than\_a\_prime = less\_than\_a\_prime / n  
P\_greater\_than\_a\_prime = greater\_than\_a\_prime / n  
  
# Аналитическое решение  
P\_less\_than\_a\_prime\_analytic = 1 - math.exp(-math.pi / 4)  
P\_greater\_than\_a\_prime\_analytic = math.exp(-math.pi / 4)  
  
# Вывод результатов  
print(f"Моделирование: P(a < a') = {P\_less\_than\_a\_prime}")  
print(f"Аналитическое решение: P(a < a') = {P\_less\_than\_a\_prime\_analytic}")  
print(f"Моделирование: P(a > a') = {P\_greater\_than\_a\_prime}")  
print(f"Аналитическое решение: P(a > a') = {P\_greater\_than\_a\_prime\_analytic}")  
print(f"Моделирование: Отношение P(a < a') / P(a > a') = {P\_less\_than\_a\_prime / P\_greater\_than\_a\_prime}")  
print(f"Аналитическое решение: Отношение P(a < a') / P(a > a') = {P\_less\_than\_a\_prime\_analytic / P\_greater\_than\_a\_prime\_analytic}")

Этот код генерирует случайные числа, подчиняющиеся закону Рэлея, и сравнивает частоту встречаемости амплитуд, меньших и больших математического ожидания, с аналитическим решением.

Результат:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

14.12) На плоскости проведены две параллельные прямые, расстояние между ними L. На эту же плоскость бросается круг радиуса R. Случайные отклонения центра круга от линий, в направлении им перпендикулярном, распределены нормально. Центр рассеивания расположен на расстоянии b от одной из линий во внешнюю сторону, а среднее квадратическое отклонение равно σ. Определить при одном бросании: а) вероятность накрытия кругом хотя бы одной прямой; б) вероятность накрытия обеих прямых, если L = 10 м, R = 8 м, b = 5 м, σ = 14,8 м.

Изображение выглядит как круг, диаграмма, графическая вставка, линия

Автоматически созданное описание

Рассмотрим СВ Х – расстояние между центром круга и ближайшей к центру рассеивания линией. Из схемы видно, что для накрытия кругом хотя бы одной прямой множество значений х:

;

Найдём вероятность:

Для накрытия кругом обоих прямых множество x:

import random  
import math  
  
def simulate\_throws(L, R, b, sigma, num\_throws):  
 hits\_at\_least\_one = 0  
 hits\_both = 0  
  
 for \_ in range(num\_throws):  
 # Случайное отклонение центра круга  
 offset = random.gauss(0, sigma)  
  
 # Координаты левой и правой линий  
 left\_line = -L / 2 + offset  
 right\_line = L / 2 + offset  
  
 # Случайная точка внутри круга  
 x = random.uniform(-R, R)  
 y = random.uniform(-R, R)  
  
 # Проверка накрытия кругом линий  
 hit\_left = abs(x) <= R and y <= left\_line and y >= left\_line - 2 \* R  
 hit\_right = abs(x) <= R and y <= right\_line and y >= right\_line - 2 \* R  
  
 if hit\_left or hit\_right:  
 hits\_at\_least\_one += 1  
 if hit\_left and hit\_right:  
 hits\_both += 1  
  
 # Вероятности накрытия кругом линий  
 prob\_at\_least\_one = hits\_at\_least\_one / num\_throws  
 prob\_both = hits\_both / num\_throws  
  
 return prob\_at\_least\_one, prob\_both  
  
# Параметры задачи  
L = 10  
R = 8  
b = 5  
sigma = 14.8  
num\_throws = 1000000 # Количество бросков для моделирования  
  
# Моделирование  
prob\_at\_least\_one, prob\_both = simulate\_throws(L, R, b, sigma, num\_throws)  
  
# Вывод результатов  
print(f"А) Вероятность накрытия кругом хотя бы одной прямой: {prob\_at\_least\_one:.4f}")  
print(f"Аналитическое решение: 0.5202")  
print(f"Б) Вероятность накрытия обеих прямых: {prob\_both:.4f}")  
print(f"Аналитическое решение: 0.1282")

Результат:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

15.1) Курс корабля составляет с линией минного заграждения случайный угол θ, все значения которого равномерно распределены в интервале (θ1, θ2). Найти вероятность подрыва корабля на контактной мине, если ширина корабля b, а расстояние между соседними минами равно l (углы θ1 и θ2 удовлетворяют условиям . Размерами можно пренебречь)

Изображение выглядит как линия, зарисовка, рисунок, диаграмма

Автоматически созданное описание

Рассмотрим СВ θ – угол между курсом корабля и линией минного заграждения. Вероятность подрыва А найдём по ФПВ:

;

Где – вероятность подрыва корабля при пересечении. Найдём эту вероятность.

;

Распределение угла θ нетрудно найти.

;

В соответствии с условием интегрируем итоговую функцию в интервале (θ1, θ2).

Моделирование:

Для моделирования рассматриваемой задачи введем параметры:

l = 80 – расстояние между минами

b = 20 – ширина корабля

s = 100 – расстояние между кораблем и минным заграждением

𝛩1 = 15 – левая граница интервала

𝛩2 = 85 – правая граница интервала

Start – положение первой мины

𝛩 - значения угла равномерно распределены в интервале (Θ1, Θ2)

Алгоритм решения с большим числом опытов:

Данный алгоритм рассматривает множество чисел опытов: . В течение

каждого опыта строится массив мин, первое значение которого выбирается случайно в интервале [0; l/2] - start = randi([0 1 / 2]);, строится массив случайных углов - Teta(i) = randi([15,85]);. Находится точка в которой корабль пересекает линию мин и проверяется не подорвался ли он. В случае благоприятного события увеличивается счетчик. После прохождения высчитывается вероятность событий как отношение количества появления событий к общему числу опытов.

Изображение выглядит как текст, снимок экрана, Шрифт, алгебра

Автоматически созданное описание

Результат программы:

Изображение выглядит как текст, Шрифт, дизайн, инструмент

Автоматически созданное описание

Если подставить постоянные значения в ответ аналитического решения получится такой же ответ. Из этого можно судить о правильности экспериментального решения.