Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра ИТАС

Отчет

по лабораторной работе №1

«Решение задач моделирования с использованием имитации случайных величин на основе метода Монте-Карло»

Вариант 2

|  |  |
| --- | --- |
| Выполнили:  Ст. гр. 820601  Беляй А.А.  Гурин Н.С.  Кохнович М.О. | Проверил:  Севернев А. М. |

Минск 2021

# ЦЕЛЬ РАБОТЫ

1.1 Изучить алгоритмы имитации случайных величин на основе метода Монте-Карло и методы проверки этих алгоритмов.

1.2 Разработать алгоритмы имитации случайных величин, необходимых для решения задачи моделирования согласно варианту задания. Реализовать разработанные алгоритмы в виде подпрограмм на любом алгоритмическом языке.

1.3 Выполнить проверку всех алгоритмов имитации случайных величин, используемых в задании, на основе гистограммы и по критерию «хи-квадрат».

1.4 Изучить примеры решения задач моделирования с использованием имитации случайных величин на основе метода Монте-Карло.

1.5 Согласно варианту задания разработать алгоритм для решения задачи на основе метода Монте-Карло. Использовать разработанные и проверенные подпрограммы имитации случайных величин.

1.6 Выполнить три испытания разработанного алгоритма. Реализовать разработанный алгоритм в виде программы на любом алгоритмическом языке.

# УСЛОВИЕ ЗАДАЧИ

На горно-обогатительном комбинате выполняется очистка и переработка некоторой руды. Руда обрабатывается партиями по 5 тонн. Производительность установок для очистки и переработки руды представляет собой случайную величину, распределенную по гауссовскому закону; средняя производительность установки для очистки – 120 кг/час, установки для переработки – 70 кг/час, среднеквадратическое отклонение производительности – 10 кг/час (для обеих установок).

Доля примесей в руде (*S*), которые требуется удалить в процессе очистки, представляет собой случайную величину. На основании наблюдений установлено, что плотность распределения этой величины может быть приближённо задана графиком на рисунке 1.

Примеси, удалённые в ходе очистки, не поступают на переработку. Если в руде содержится более 20% примесей, то примерно в 70% случаев качество очистки оказывается недостаточным, и руда направляется на повторную очистку. Более двух раз очистка, как правило, не требуется.

Требуется определить:

– среднее время обработки одной партии руды;

– вероятность того, что для партии потребуется повторная очистка.

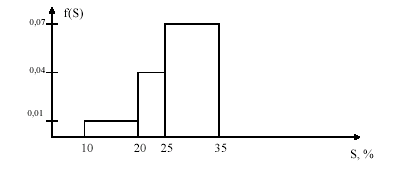


Рисунок 1 – Плотность распределения

# ХОД РАБОТЫ

Приведем программную реализацию предложенного алгоритма на языке *Python*, выполняющую 3 испытания.

Основные константы и переменные программы: *razmer\_partii* – размер партии руды; *m\_och* – средняя производительность установки для очистки; *m\_per* – средняя производительность установки для переработки; *sigma* – среднеквадратическое отклонение производительности для двух установок.

*import random*

*from math import sqrt*

*import matplotlib.pyplot as plt*

*import matplotlib as mpl*

*def build\_histogram(data\_values, data\_names, title):*

*dpi = 80*

*fig = plt.figure(dpi=dpi, figsize=(512 / dpi, 384 / dpi))*

*mpl.rcParams.update({'font.size': 10})*

*plt.title(title)*

*ax = plt.axes()*

*ax.yaxis.grid(True, zorder=1)*

*xs = range(len(data\_names))*

*plt.bar([x + 0.3 for x in xs], data\_values,*

*width=0.8, color='blue', alpha=0.8,*

*zorder=2)*

*plt.xticks(xs, data\_names)*

*fig.autofmt\_xdate(rotation=25)*

*plt.legend(loc='upper right')*

*fig.show()*

*def grafic\_raspr():*

*r = random.random()*

*x = None*

*if (r >= 0) and (r < 0.1):*

*x = (r + 0.1) / 0.01*

*elif (r >= 0.1) and (r < 0.3):*

*x = (r + 0.7) / 0.04*

*elif (r >= 0.3) and (r < 1):*

*x = (r + 1.45) / 0.07*

*return x*

*def gauss(m, sigma):*

*k = 6*

*rsum = 0*

*for i in range(k):*

*r = random.random()*

*rsum = rsum + r*

*x = m + sigma \* sqrt(12 / k) \* (rsum - k / 2)*

*return x*

*def proverka\_1():*

*gran = [i for i in range(10, 36)]*

*p = [0.01 for i in range(10)]+[0.04 for i in range(5)]+[0.07 for i in range(10)]*

*k = 25*

*n = 100000*

*d = 1*

*m = [0 for i in range(25)]*

*f = [0 for i in range(25)]*

*for i in range(n):*

*x = grafic\_raspr()*

*for j in range(k):*

*if (x > gran[j]) and x <= gran[j + 1]:*

*m[j] += 1*

*print("\nПроверка 1")*

*for i in range(k):*

*f[i] = m[i] / (n \* d)*

*print("diap", gran[i], "-", gran[i+1])*

*print(f"m[{i}]=", m[i])*

*print(f"f[{i}]=", f[i])*

*# подсчет переменной хи-квадрат*

*hi\_kv = 0*

*for i in range(k):*

*hi\_kv += (m[i] ^ 2) / (n \* p[i])*

*print("\nхи-квадрат", hi\_kv)*

*build\_histogram(f, [str(i + 10) + " - " + str(i + 11) for i in range(25)], 'First Check')*

*def proverka\_2():*

*gran = [90, 100, 110, 120, 130, 140, 150]*

*p = [0.0214, 0.136, 0.3413, 0.3413, 0.136, 0.0214]*

*k = 6*

*n = 5000*

*d = 10*

*m = [0 for i in range(6)]*

*f = [0 for i in range(6)]*

*for i in range(n):*

*x = gauss(120, 10)*

*for j in range(k):*

*if (x > gran[j]) and x <= gran[j + 1]:*

*m[j] += 1*

*print("\nПроверка 2")*

*for i in range(k):*

*f[i] = m[i] / (n \* d)*

*print("diap", gran[i], "-", gran[i+1])*

*print(f"m[{i}]=", m[i])*

*print(f"f[{i}]=", f[i])*

*# подсчет переменной хи-квадрат*

*hi\_kv = 0*

*for i in range(k):*

*hi\_kv += pow(m[i], 2 / (n \* p[i]))*

*# hi\_kv -= n*

*print("\nхи-квадрат", hi\_kv)*

*build\_histogram(f, [str(i) + " - " + str(i + 10) for i in range(90, 150, 10)], 'Second Check')*

*def proverka\_3():*

*gran = [40, 50, 60, 70, 80, 90, 100]*

*p = [0.0214, 0.136, 0.3413, 0.3413, 0.136, 0.0214]*

*k = 6*

*n = 5000*

*d = 10*

*m = [0 for i in range(6)]*

*f = [0 for i in range(6)]*

*for i in range(n):*

*x = gauss(70, 10)*

*for j in range(k):*

*if (x > gran[j]) and x <= gran[j + 1]:*

*m[j] += 1*

*print("\nПроверка 3")*

*for i in range(k):*

*f[i] = m[i] / (n \* d)*

*print("diap", gran[i], "-", gran[i+1])*

*print(f"m[{i}]=", m[i])*

*print(f"f[{i}]=", f[i])*

*# подсчет переменной хи-квадрат*

*hi\_kv = 0*

*for i in range(k):*

*hi\_kv += (m[i] ^ 2) / (n \* p[i])*

*print("\nхи-квадрат", hi\_kv)*

*build\_histogram(f, [str(i) + " - " + str(i + 10) for i in range(40, 100, 10)], 'Third Check')*

*def raschet():*

*razmer\_partii = 5000*

*m\_och = 120*

*m\_per = 70*

*sigma = 10*

*n = 2000*

*kol = 0*

*t00 = 0*

*tp0 = 0*

*for i in range(n):*

*sumr1 = 0*

*sumr2 = 0*

*for j in range(6):*

*r1 = random.random()*

*r2 = random.random()*

*sumr1 += r1*

*sumr2 += r2*

*o = m\_och + sigma \* sqrt(2) \* (sumr1 - 3)*

*p = m\_per + sigma \* sqrt(2) \* (sumr2 - 3)*

*# Доля примесей S-%*

*r = random.random()*

*if r < 0.1:*

*S = (r + 0.1) / 0.01*

*elif r < 0.3:*

*S = (r + 0.7) / 0.04*

*else:*

*S = (r + 1.45) / 0.07*

*t0 = razmer\_partii / o*

*ver = random.random()*

*if (S > 20) and (ver < 0.7):*

*kol += 1*

*t0 += razmer\_partii / o*

*tp = (razmer\_partii - S \* razmer\_partii / 100) / p*

*# Общее время очистки*

*t00 += t0*

*# Общее время переработки*

*tp0 += tp*

*# Среднее время обработки одной партии*

*t = tp0 / n + t00 / n*

*print("\nСреднее время обработки одной партии", t, "часов")*

*vrst = kol / n*

*print("Вероятность того, что для партии потребуется повторная очистка =", vrst \* 100, "%")*

*proverka\_1()*

*proverka\_2()*

*proverka\_3()*

*raschet()*

По результатам выполнения программы получены следующие результаты (рисунки 2-7).

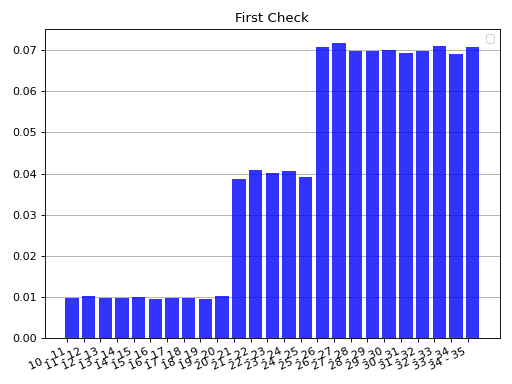


Рисунок 2 – Первая проверка

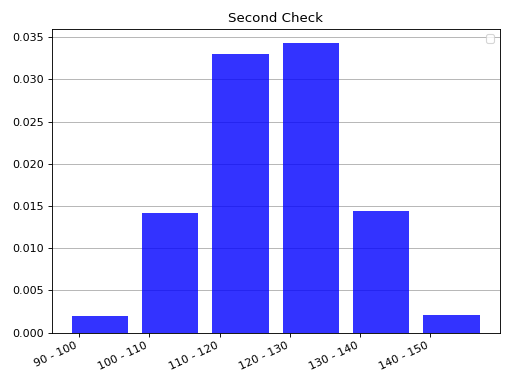


Рисунок 3 – Вторая проверка

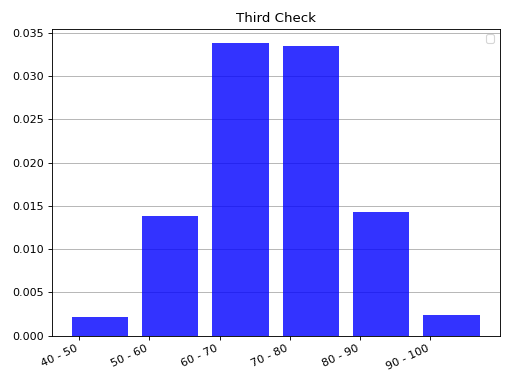


Рисунок 4 – Третья проверка

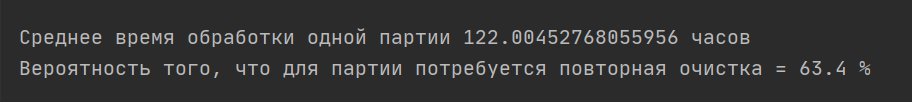


Рисунок 5 – Результаты первого запуска программы

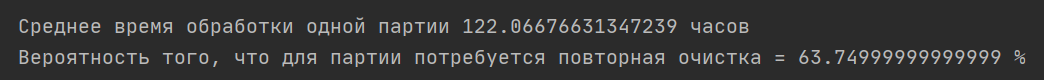


Рисунок 6 – Результаты второго запуска программы

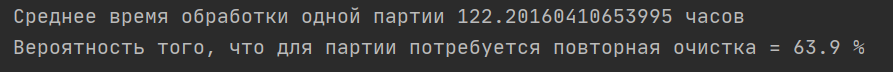


Рисунок 7 – Результаты третьего запуска программы

# ВЫВОДЫ

В результате выполнения данной лабораторной работы освоена теория для решения задач моделирования с использованием имитации случайных величин на основе метода Монте-Карло. Также разработали и реализовали алгоритм для решения задачи на языке *Python*. Выполнили три проверки разработанного алгоритма.