МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра математического моделирования и анализа данных

АВСЯННИК
Елизавета Дмитриевна

Лабораторная работа №2

Имитационное и статистическое моделирование

Проверил:
В.П. Кирлица

Минск, 2021

0.0.1 Лабораторная работа 2: Задание 6.2, пункт 1), стр. 175

По 1000 реализаций S вычисленных по формуле (1), оценить числовые характеристики S : $S_{min}, S_{max}, E\{S\}, D\{S\}, P\{S_1 <= S <= S_2\}$, для следующих значений параметров, входящих в правую часть формулы (1):

i_3	8.6 %	8.7 %	8.8 %	9.2 %
P	0.1	0.3	0.5	0.1

i_4	8.7 %	8.9 %	9.4 %
P	0.1	0.6	0.3

Формула (1): $S = P(1 + n_1 i_1 + n_2 i_2 + ... + n_k i_k)$

[240]:
```
import numpy as np
import pandas as pd
from math import floor
from scipy.stats import chi2
import time

import typing as tp

from tabulate import tabulate
```

Мультипликативный конгруэнтный метод моделирования БСВ

[241]:
```
def generate_brv_congruential_sample(M: int=2**31, alpha_star0: int=65539, beta :
 ↪int=65539) -> float:
    alpha_t = alpha_star0
    while True:
        alpha_t = (alpha_t * beta) % M
        yield alpha_t / M

def generate_brv_congruential(n: int=100, M: int=2**31, alpha_star0: int=65539,␣
 ↪beta :int=65539) -> np.ndarray:
    alpha = np.array([])
    generator = generate_brv_congruential_sample(M=M, alpha_star0=alpha_star0,␣
 ↪beta=beta)
    for i in range(n):
        alpha = np.append(alpha, next(generator))
    return alpha
```

Метод Макларена-Марсальи моделирования БСВ для моделирования равномерного распределения на отрезке

[242]:
```
def generate_brv_mm_sample(k: int=128, a:float=0, b:float=1) -> float:
    v = np.array([])
    brv_cong_b = generate_brv_congruential_sample()
    t = time.perf_counter()
    alpha_star0 = beta =  int(10**9*float((t-int(t))))
    brv_cong_c = generate_brv_congruential_sample(alpha_star0=alpha_star0,␣
 ↪beta=beta)
    for i in range(k):
            v = np.append(v, next(brv_cong_b))
    while True:
        index = floor(next(brv_cong_c) * k)
        alpha_t = v[index]
        v[index] = next(brv_cong_b)
        yield alpha_t * (b - a) + a

def generate_brv_mm(n: int=100, k: int=128,  a:float=0, b:float=1) -> np.ndarray:
    alpha = np.array([])
    generator = generate_brv_mm_sample(k=k, a=a, b=b)
    for i in range(n):
        alpha = np.append(alpha, next(generator))
    return alpha
```

Алгоритм моделирования ДСВ (для выбора i_3 и i_4)

[243]:
```
def random_choice(array: list=None, size: int=1, probas: list=None) -> list:
    probas_cum = []
    last_element = 0
    for i in range(len(probas)):
        probas_cum.append(last_element + probas[i])
        last_element = probas_cum[i]

    generator = generate_brv_mm_sample()

    result = []
    for i in range(size):
            rand_uniform = next(generator)
            for j in range(len(probas_cum)):
                if(rand_uniform < probas_cum[j]):
                    result.append(array[j])
                    break
    return result if size > 1 else result[0]
```

0.0.2 Основной цикл программы

[244]:
```
k = 4
n = 0.25
P = 1230
S_interval = [1335, 1340]
R = {"i_1": (8.2, 9), "i_2" : (8.4, 9.1)}
probas = {"i_3": {8.6: 0.1, 8.7: 0.3, 8.8: 0.5, 9.2: 0.1}, "i_4": {8.7: 0.1, 8.9:
 ↪ 0.6, 9.3: 0.3}}

N = 1000
S = np.ones(N)

i_param_full = []

for j in range(N):
    i_arr = []
    for i in range(k):
        if i == 0 or i == 1:
            generator = generate_brv_mm_sample(a=R[f"i_{i+1}"][0],␣
 ↪b=R[f"i_{i+1}"][1])
            i_arr.append(next(generator))
        else:
            i_arr.append(random_choice(list(probas[f"i_{i+1}"].keys()), \
                                       size=1, probas=list(probas[f"i_{i+1}"].
 ↪values())))
        S[j] += n * i_arr[i] / 100
    i_param_full.append(i_arr)
    S[j] *= P
```

0.0.3 Получены 1000 реализаций вектора $i = (i_1, i_2, i_3, i_4)$

Показаны первые 10

[245]:
```
print(tabulate(i_param_full[:10], headers=['i_1', 'i_2', 'i_3', 'i_4'],␣
 ↪floatfmt=".4f"))
```

```
   i_1     i_2     i_3     i_4
------  ------  ------  ------
8.6559  8.4867  8.8000  9.3000
8.2870  8.8540  9.2000  9.3000
8.4259  8.9719  8.8000  8.9000
8.4613  8.6932  8.7000  8.9000
8.4468  8.4761  8.8000  8.9000
8.8611  9.0606  8.8000  8.9000
8.2571  8.9191  8.8000  8.9000
8.4208  9.0494  8.8000  8.9000
8.8516  8.4749  9.2000  9.3000
```

```
8.3332  8.5971  8.8000  8.7000
```

0.0.4 Получены 1000 реализаций величины S

Показаны первые 10

[246]:
```
print(S[:10])
```

```
[1338.37111041 1339.59589117 1337.92569371 1336.87026313 1336.46548211
 1339.5366481  1337.24436071 1338.14816734 1340.16648849 1335.87305739]
```

0.0.5 S_{min}

[247]:
```
S.min()
```

[247]:
```
1334.2610075343855
```

0.0.6 S_{max}

[248]:
```
S.max()
```

[248]:
```
1341.9461214276182
```

0.0.7 $E\{S\}$

[249]:
```
S.mean()
```

[249]:
```
1338.4074306560335
```

0.0.8 $D\{S\}$

[250]:
```
S.var()
```

[250]:
```
1.78976247023972
```

0.0.9 $P\{S_1 <= S <= S_2\}$

[251]:
```
len([s for s in S if s <= S_interval[1] and s >= S_interval[0]]) / len(S)
```

[251]:
```
0.869
```

[ ]:
```

```