Sheet

Міністерство освіти і науки України Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 1 з дисципліни «Моделювання систем»

«Перевірка генератора випадкових чисел на відповість закону розподілу»

Виконав: ІП-91 Газін Костянтин Андрійович

Перевірив: Дифучин Антон Юрійович

Оцінка _____

Київ 2021

Згенерувати 10000 випадкових чисел трьома вказаними нижче способами. **45 балів.**

- Згенерувати випадкове число за формулою $x_i = -\frac{1}{\lambda} ln(\xi_i)$, де ξ_i випадкове число, рівномірно розподілене в інтервалі (0;1). Числа ξ_i можна створювати за допомогою вбудованого в мову програмування генератора випадкових чисел. Перевірити на відповідність експоненційному закону розподілу $F(x) = 1 - e^{-\lambda x}$. Перевірку зробити при різних значеннях λ.
- Згенерувати випадкове число за формулами:

  $$x_i = \sigma \mu_i + a,$$
  $$\mu_i = \sum_{i=1}^{12} \xi_i - 6$$

  де ξ - випадкове число, рівномірно розподілене в інтервалі (0;1). Числа ξ_i можна створювати за допомогою убудованого в мову програмування генератора випадкових чисел. Перевірити на відповідність нормальному закону розподілу:

  $$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-a)^2}{2\sigma^2})$$

  Перевірку зробити при різних значеннях a і σ.
- Згенерувати випадкове число за формулою $z_{i+1} = az_i(mod\ c), x_{i+1} = z_{i+1}/c$, де $a = 5^{13}$, $\text{c} = 2^{31}$. Перевірити на відповідність рівномірному закону розподілу в інтервалі (0;1). Перевірку зробити при різних значеннях параметрів a і c.

Для кожного побудованого генератора випадкових чисел побудувати гістограму частот, знайти середнє і дисперсію цих випадкових чисел. По виду гістограми частот визначити вид закону розподілу. **20 балів**.

Відповідність заданому закону розподілу перевірити за допомогою критерію згоди χ^2. **30 балів**

Зробити висновки щодо запропонованих способів генерування випадкових величин. **5 балів**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm, expon, chi2, uniform
```

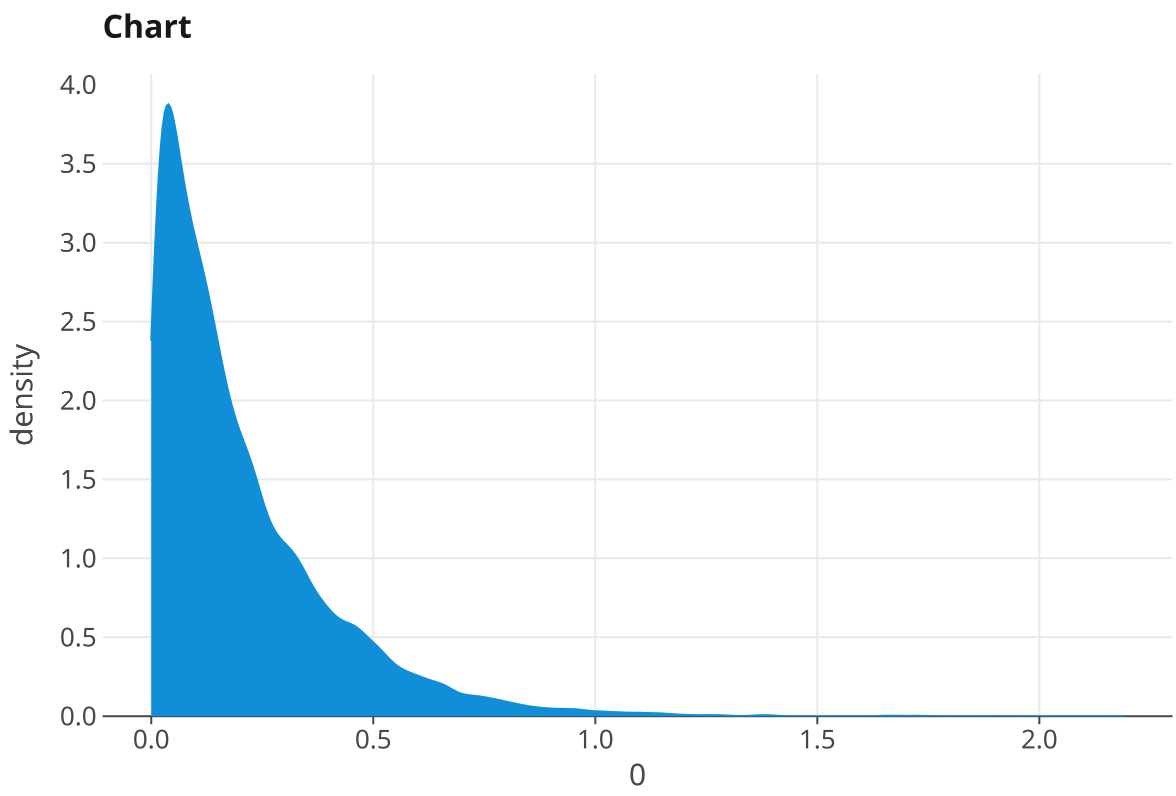
```
def optimal_intervals(n: int):
    return np.floor(1 + 3.332 * np.log10(n))

def partition(arr: np.ndarray):
    intervals: int = optimal_intervals(arr.size)
    step = (arr.max() - arr.min())/intervals
    cols = [arr.min() + i * step for i in np.arange(0, intervals)]
    res = np.empty(int(intervals))
    for elem in arr:
        for i in range(0, int(intervals)):
            if elem >= cols[i]:
                if i >= intervals - 1:
                    res[i] += 1
                elif elem < cols[i + 1]:
                    res[i] += 1
    return res, cols, step, intervals

def theoretical_exponential(l, partitions: np.ndarray, step):
    end_parts = partitions + step
    res = np.empty(partitions.size)
    for i in np.arange(0, partitions.size):
        res[i] = expon.cdf(end_parts[i], scale = 1 / l) - norm.cdf(partitions[i
    return res

def theoretical_norm(sigma, a, partitions: np.ndarray, step):
    end_parts = partitions + step
    res = np.empty(partitions.size)
    for i in np.arange(0, partitions.size):
        res[i] = norm.cdf(end_parts[i], loc = a, scale = sigma) - norm.cdf(part
    return res

def theoretical_uniform(start, length, partitions: np.ndarray, step):
    end_parts = partitions + step
    res = np.empty(partitions.size)
    for i in np.arange(0, partitions.size):
        res[i] = uniform.cdf(end_parts[i], loc = start, scale = length) - unifo
    return res


def get_xi2(probs, nteor):
    return (((probs - nteor)**2)/nteor).sum()

def normalize(arr: np.ndarray):
    return arr / arr.sum()
```

```
quantity = 10000
lambda_num = 5
rng = np.random.default_rng()
alpha = 1-.025
sigma = 5
a = 2
```

Exponential

```
def exponential_distribution(lambda_num, quantity):
    xi = np.array([-1/lambda_num*np.log(rng.random()) for x in np.arange(1, quan
    print('Mean for distribution is ' + str(xi.mean()))
    print('Variance for distribution is ' + str(xi.var()))
    return xi
exp = exponential_distribution(lambda_num, quantity)
expS = pd.DataFrame(exp)
```

```
Mean for distribution is 0.19973070909156557
Variance for distribution is 0.03959801088994999
```



```
ex_real, ex_cols, ex_step, ex_intervals = partition(exp)
ex_theor = theoretical_exponential(lambda_num, np.array(ex_cols), ex_step)

ex_real = normalize(ex_real)

chi2_real_ex = get_xi2(ex_real, ex_theor)
print("Chi squared: " + str(chi2_real_ex))
ex_freedom: int = int(ex_intervals) - 1 - 1
print("Degrees of freedom: " + str(ex_freedom))

chi2_crit_ex = chi2.ppf(alpha, df=ex_freedom)
print("Chi squared critical: " + str(chi2_crit_ex))


if chi2_crit_ex > np.abs(chi2_real_ex):
    print("This distribution is exponenial with alpha "
        + str(alpha) + " lamda " + str(lambda_num) + " and sample size " + str(
else:
    print("This distribution is NOT exponenial with alpha "
        + str(alpha) + " lamda " + str(lambda_num) + " and sample size " + str(
```

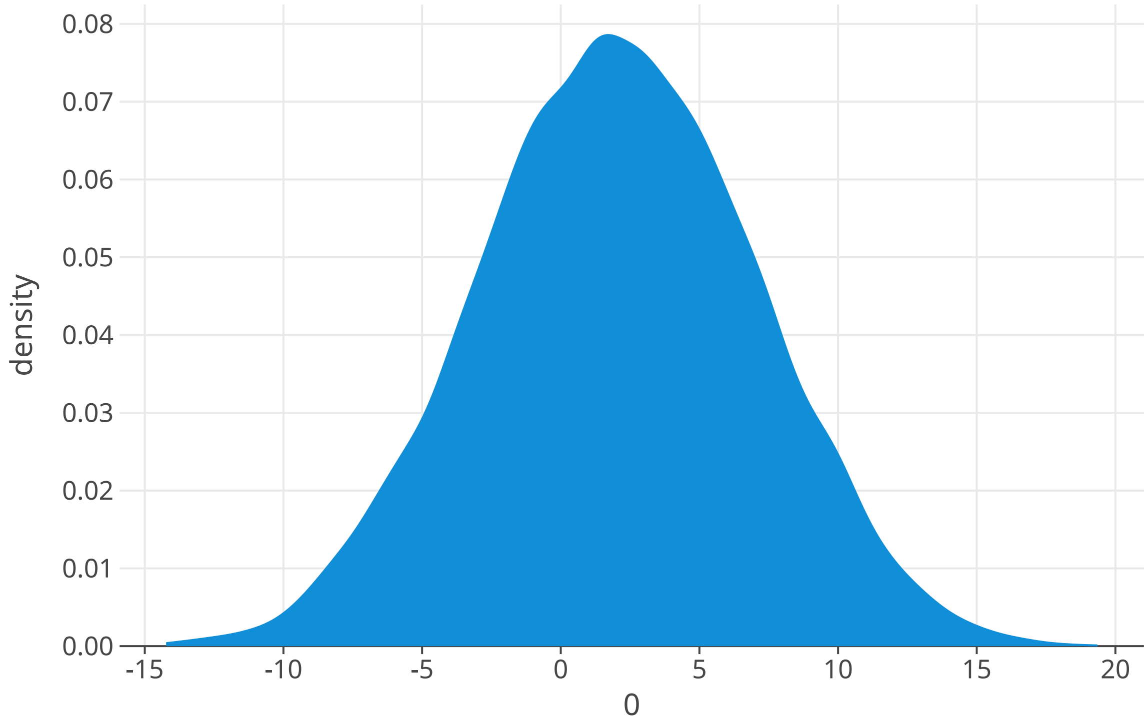
```
Chi squared: 11.24676086174214
Degrees of freedom: 12
Chi squared critical: 23.33666415864534
This distribution is exponenial with alpha 0.975 lamda 5 and sample size 1000
```

Normal

```
def normal_distribution(sigma, a, quantity):
    ar2 = []

    for i in np.arange(0, quantity):
        mu = np.array([rng.random() for y in np.arange(12)]).sum() - 6
        x2 = sigma * mu + a
        ar2.append(x2)
    xi2 = np.array(ar2)
    print('Mean for distribution is ' + str(xi2.mean()))
    print('Variance for distribution is ' + str(xi2.var()))
    return xi2

normallin = normal_distribution(sigma, a, quantity)
normallin_df = pd.DataFrame(normallin)
```

```
Mean for distribution is 2.0247525732242946
Variance for distribution is 24.745578437210863
```

Chart

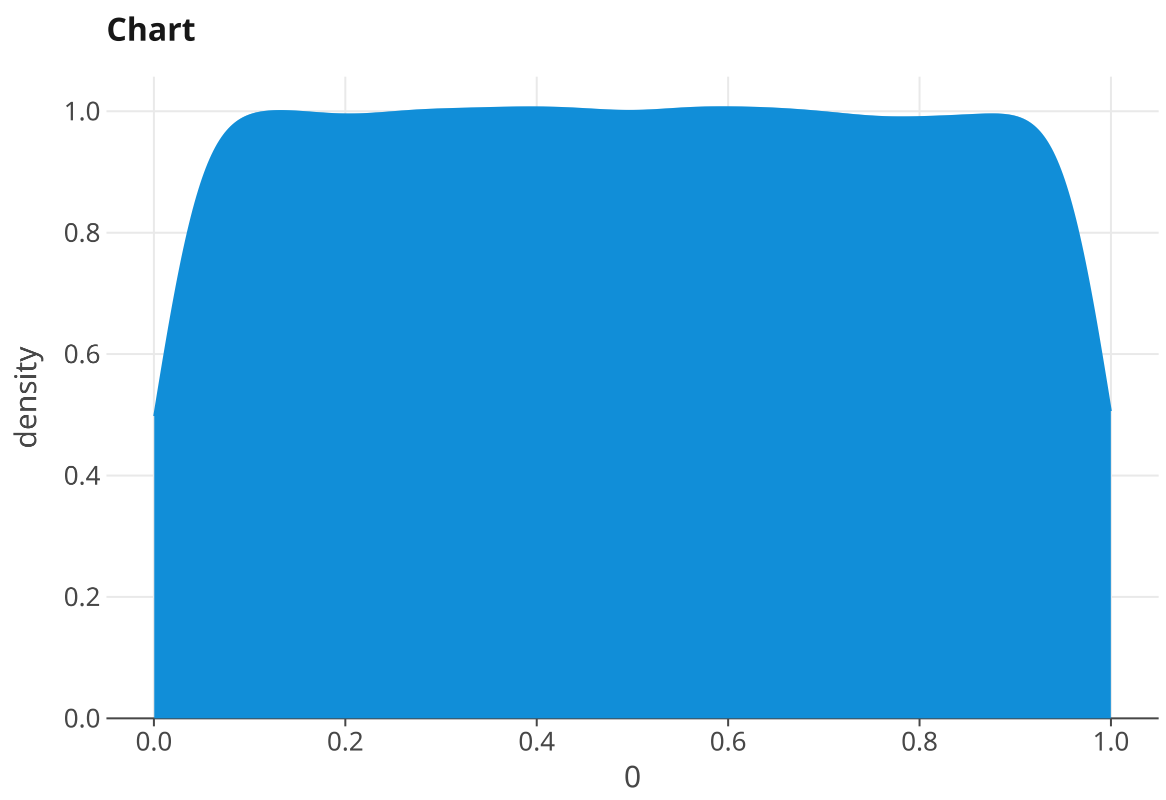



```
norm_real, norm_cols, norm_step, norm_intervals = partition(normallin)
norm_theor = theoretical_norm(sigma, a, np.array(norm_cols), norm_step)

norm_real = normalize(norm_real)

chi2_real_norm = get_xi2(norm_real, norm_theor)
print("Chi squared: " + str(chi2_real_norm))
norm_freedom: int = int(norm_intervals) - 2 - 1
print("Degrees of freedom: " + str(norm_freedom))

chi2_crit_norm = chi2.ppf(alpha, df=norm_freedom)
print("Chi squared critical: " + str(chi2_crit_norm))


if chi2_crit_norm > np.abs(chi2_real_norm):
    print("This distribution is normal with alpha "
        + str(alpha) + " sigma " + str(sigma) + " mean " + str(a) + " and sampl
else:
    print("This distribution is NOT normal with alpha "
        + str(alpha) + " sigma " + str(sigma) + " mean " + str(a) + " and sampl
```

```
Chi squared: 0.0012490780593698183
Degrees of freedom: 11
Chi squared critical: 21.9200492610212
This distribution is normal with alpha 0.975 sigma 5 mean 2 and sample size 1
```

Uniform

```python
a2 = np.power(5, 13)
c = np.power(2, 31)
z0 = rng.random() * c
def uniform_gen(a2, c, z0):
    ar3 = np.empty(quantity)
    ar3[0] = z0
    for i in np.arange(1, quantity):
        ar3[i] = a2 * ar3[i - 1] % c
    xi3 = ar3 / c
    return xi3

uni = uniform_gen(a2, c, z0)
uni_df = pd.DataFrame(uni)
```



```
uni_real, uni_cols, uni_step, uni_intervals = partition(uni)
uni_theor = theoretical_uniform(0, 1, np.array(uni_cols), uni_step)

uni_real = normalize(uni_real)

chi2_real_uni = get_xi2(uni_real, uni_theor)
print("Chi squared: " + str(chi2_real_uni))
uni_freedom: int = int(uni_intervals) - 2 - 1
print("Degrees of freedom: " + str(uni_freedom))

chi2_crit_uni = chi2.ppf(alpha, df=uni_freedom)
print("Chi squared critical: " + str(chi2_crit_uni))


if chi2_crit_uni > np.abs(chi2_real_uni):
    print("This distribution is uniform with alpha "
        + str(alpha) + " start " + str(0) + " end " + str(1) + " and sample siz
else:
    print("This distribution is NOT uniform with alpha "
        + str(alpha) + " start " + str(0) + " end " + str(1) + " and sample siz
```

```
Chi squared: 7.704252979966334e-05
Degrees of freedom: 11
Chi squared critical: 21.9200492610212
This distribution is uniform with alpha 0.975 start 0 end 1 and sample size 1
```

Висновок

У лабораторній роботі ми розглянули декілька способів генерування випадкових величин:

- По оберненій функції
- По рекурентній формулі
- Спец методами

Найпростіший метод це перший, але не всюди його можна застосувати. Коли його не можуть застосувати, то використовують два інші.