МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра компьютерных технологий и систем

ИМИТАЦИОННОЕ И СТАТИСТИЧЕСКОЕ МОДЕЛИРОВАНИЕ

ОТЧЁТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

Савицкой Елизаветы Дмитриевны

студентки 4 курса, 4 группы

Минск, 2021

Задание №7

1) **№2, стр. 52**

Моделирование непрерывных случайных величин (НСВ)

Основными методами построения моделирующих алгоритмов для указанных законов распределения являются:

- метод обратной функции;
- метод исключения;
- метод функциональных преобразований,

и другие методы, основанные на учете свойств распределений.

Универсальными методами проверки точности моделирования НСВ являются критерии согласия (χ^2 Пирсона, Колмогорова и др.), а также критерии серий, реализованные в пакете. Графические методы анализа точности моделирования НСВ включают:

- анализ гистограммы частот распределения;
- анализ эмпирической функции распределения

Одномерное нормальное распределение

НСВ $\xi \in R^1$ с плотностью распределения

$$p_\xi(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp\{-\frac{(x-\mu)^2}{2\sigma^2}\}$$

имеет одномерное нормальное (гауссово) распределение с параметрами: средним значением $\mu \in R^1$ и дисперсией $\sigma^2 > 0$, (обозначается $N_1(\mu,\ \sigma^2)$).
Функция распределения (0,1) N1 обозначается Φ(x) и имеет вид:

$$\Phi(x) = F_\eta(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{t^2}{2}}dt$$

и называется функцией Лапласа. Случайные величины ξ и η связаны соотношением:

$$\xi = \eta + \sigma \cdot \eta$$

где σ - среднее квадратическое (стандартное) отклонение. Таким образом, задача моделирования $\xi \sim N_1(\mu,\ \sigma^2)$) сводится к моделированию стандартной гаусовскойСВ η и применению формулы $\xi = \eta + \sigma \cdot \eta$.

Задание

Используя случайные выборки реализаций объема $n = 1000$, сравнить по точности и быстродействию методы моделирования СВ $\xi \sim N_1(\mu,\ \sigma^2)$. Положить: $\mu_1 = 0$, $\sigma^2 = 1.2,:\ \mu_1 = 1,\ \sigma^2 = 0.1$. Получить последовательность реализаций СВ ξ с «усеченным» нормальным распределением. Оценить долю пропущенных реализаций СВ η из $n = 1000$ смоделированных. Положить: $\sigma^2 = 9, \mu = 3{,}6{,}9$.

$$\xi = \begin{cases} x,\ если\quad x > 0\ (где\ x - реализация\quad СВ\ \eta \sim N_1(\mu\sigma^2), \\ \text{пропускается в противном случае} \end{cases}$$

Алгоритмы моделирования для нормального распределения

Первый алгоритм реализуем методом суммирования, основанном на центральной предельной теореме: если $a_1, a_2, a_3, \dots, a_N$ - независимые БСВ, то при $N \to \infty$ случайная величина $\zeta = \sqrt{\frac{12}{N}}\left(\sum_{i=1}^{N} a_i - \frac{N}{2}\right)$ распределена асимптотически нормально, так что $F_\zeta(x) \to \Phi(x), x \in R^1$. На практике приемлемая точность аппроксимации стандартной гаусовской СВ достигается при $N = 12$.

```python
#  алгоритм реализуемый методом суммирования, основанном на центральной предельной теореме
def sampling_using_the_central_limit_theorem(sample_of_realizations_, location, scale, n_):
    start_time = timeit.default_timer()
    for _ in range(n_):
        sample_of_realizations_.append(location + scale * (sum([random() for _ in range(12)]) - 6))
    end_time = timeit.default_timer()
    return end_time - start_time
```

Второй алгоритм основан на методе функционального преобразования БСВ. Известно, что если a_1, a_2 - независимые БСВ, то случайные величины

$$\eta_1 = \sqrt{-2 * ln a_1} * \sin(2\pi a_2),\ \eta_2 = \sqrt{-2 * ln a_1} * \cos(2\pi a_2)$$

Являются независимыми стандартными гаусовскими. Таким образом, алгоритм моделирования η ~ (0,1) N1 на основе данного метода позволяет получить из двух реализаций a_1, a_2 БСВ две независимые реализации СВ η с помощью преобразований.

```python
# алгоритм основанный на методе функционального преобразования БСВ
def sampling_using_method_of_functional_transformations(sample_of_realizations_, location, scale, n_):
    start_time = timeit.default_timer()
    for counter in range(int(n_ / 2)):
        a1 = random()
        a2 = random()
        n1 = math.sqrt(-2 * math.log(a1)) * math.sin(2 * math.pi * a2)
        n2 = math.sqrt(-2 * math.log(a1)) * math.cos(2 * math.pi * a2)
        sample_of_realizations_.append(location + n1 * scale)
        sample_of_realizations_.append(location + n2 * scale)
    end_time = timeit.default_timer()
    return end_time - start_time
```

Критерий серий

Критерий серий предназначен для проверки гипотезы о случайности выборки $\{x_1, x_2, \dots, x_n\}$. Критерий основан на исследовании знаковой последовательности разностей:

$x_i - x_{med} (i = \overline{1, n})$, где x_{med} - медиана выборки $\{x_i\}$. Знаковая последовательность состоит из знаков "$+$", "$-$", соответствующих разностям и характеризуется:

- $\gamma(K)$ - общим числом серий;
- $T(K)$ - протяжённостью самой длинной серии, где K $(K \le N)$ - число элементов знаковой последовательности.

Под "серией" понимается последовательность подряд идущих одинаковых знаков. Очевидно, если $\{x_i\}$ - случайная выборка, то знаковая последовательность не должна содержать слишком длинных серий, а общее число серий не должно быть слишком малым. Если одно из следующих неравенств $\gamma(K) > [0.5(K+1-1.96\sqrt{K-1})]; T(K) < [3.3\log_{10}(K+1)]$, отвергается, то гипотеза о случайности выборки $\{x_i\}$ отвергается.

```python
# первая часть задания - сравнение точности и быстродействия
#  алгоритм реализуемый методом суммирования, основанном на центральной предельной теореме
time_using_the_central_limit_theorem_01 = sampling_using_the_central_limit_theorem(
    sample_using_the_central_limit_theorem_01, location_01_01, scale_01_01, n)

time_using_the_central_limit_theorem_02 = sampling_using_the_central_limit_theorem(
    sample_using_the_central_limit_theorem_02, location_01_02, scale_01_02, n)

# алгоритм основанный на методе функционального преобразования БСВ
time_using_method_of_functional_transformations_01 = sampling_using_method_of_functional_transformations(
    sample_using_functional_transformations_01, location_01_01, scale_01_01, n)

time_using_method_of_functional_transformations_02 = sampling_using_method_of_functional_transformations(
    sample_using_functional_transformations_02, location_01_02, scale_01_02, n)
# сравнение быстродействия
print('-------------------------------------- Compare Time --------------------------------------')
print('N(', location_01_01, ', ', variance_01_01, ') - using the central limit theorem - time : ',
      time_using_the_central_limit_theorem_01, sep='')
print('N(', location_01_02, ', ', variance_01_02, ') - using the central limit theorem - time : ',
      time_using_the_central_limit_theorem_02, sep='')
print('------------------------------------------------------------------------------------------')
print('N(', location_01_01, ', ', variance_01_01,
      ') - using the method of functional transformations - time : ',
      time_using_method_of_functional_transformations_01, sep='')
print('N(', location_01_02, ', ', variance_01_02,
      ') - using the method of functional transformations - time : ',
      time_using_method_of_functional_transformations_02, sep='')

pd.DataFrame({'using the central limit theorem N(0, 1.2) ': sample_using_the_central_limit_theorem_01}).plot(
    kind='hist', bins=100, alpha=0.4)
pd.DataFrame({'using the central limit theorem N(1, 0.1) ': sample_using_the_central_limit_theorem_02}).plot(
    kind='hist', bins=100, alpha=0.4)
pd.DataFrame({'using the method of functional transformations N(0, 1.2) ': sample_using_functional_transformations_01}).plot(
    kind='hist', bins=100, alpha=0.4)
pd.DataFrame({'using the method of functional transformations N(1, 0.1) ': sample_using_functional_transformations_02}).plot(
    kind='hist', bins=100, alpha=0.4)
plt.show()
```

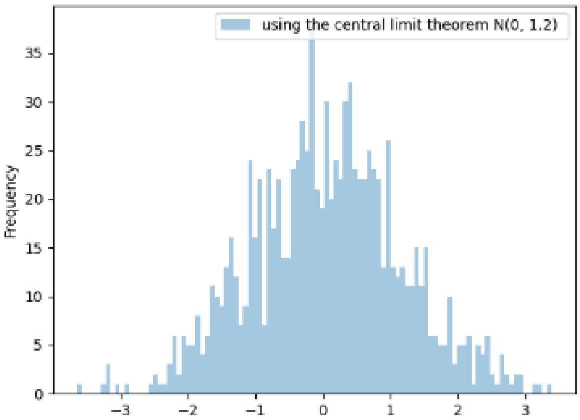
```python
# критерий серий
def series_criterion(sample_of_realizations_):
    sequence_of_signed_differences = []
    # находим медиану
    median = 0  # медиана выборки
    if len(sample_of_realizations_) % 2 != 0:
        median = sorted(sample_of_realizations_)[int(len(sample_of_realizations_) / 2)]
    else:
        median = (sorted(sample_of_realizations_)[int(len(sample_of_realizations_) / 2 - 1)] +
                  sorted(sample_of_realizations_)[int(len(sample_of_realizations_) / 2)]) / 2
    for counter in range(len(sample_of_realizations_)):
        if sample_of_realizations_[counter] != median:
            if sample_of_realizations_[counter] > median:
                sequence_of_signed_differences.append(1)
            else:
                sequence_of_signed_differences.append(0)
    print(sequence_of_signed_differences)
    series = []
    counter = 0
    len_of_series = 0
    while counter < len(sequence_of_signed_differences) - 1:
        len_of_series = 0
        counter_in_series = counter
        while sequence_of_signed_differences[counter] == sequence_of_signed_differences[counter_in_series]:
            len_of_series += 1
            counter_in_series += 1
            if counter_in_series == len(sequence_of_signed_differences) - 1:
                if sequence_of_signed_differences[counter_in_series - 1] == \
                        sequence_of_signed_differences[counter_in_series]:
```

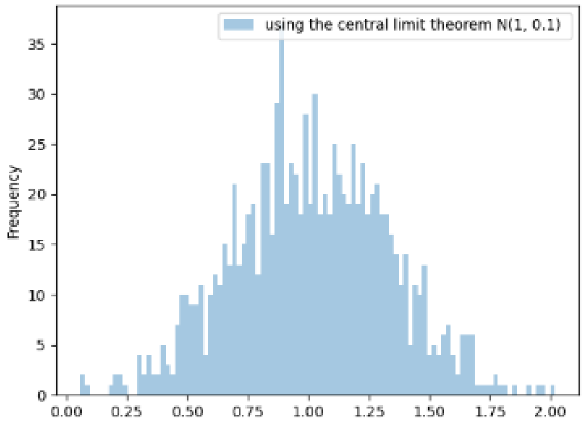
```python
            len_of_series += 1
            counter_in_series += 1
            if counter_in_series == len(sequence_of_signed_differences) - 1:
                if sequence_of_signed_differences[counter_in_series - 1] == \
                        sequence_of_signed_differences[counter_in_series]:
                    len_of_series += 1
                else:
                    series.append(1)
                break
        series.append(len_of_series)
        counter = counter_in_series
    print(series)
    total_number_of_series = len(series)  # общие числом серий
    length_of_the_longest_series = max(series)  # протяжённостью самой длинной серии
    number_of_elements_of_signed_sequence = len(sequence_of_signed_differences)  # число элементов знаковой
    # последовательности
    if total_number_of_series > 0.5 * (number_of_elements_of_signed_sequence + 1
                                       - 1.96 * math.sqrt(number_of_elements_of_signed_sequence - 1)) or \
            length_of_the_longest_series < 3.3 * math.log10(number_of_elements_of_signed_sequence + 1):
        print("Test passed !")
```

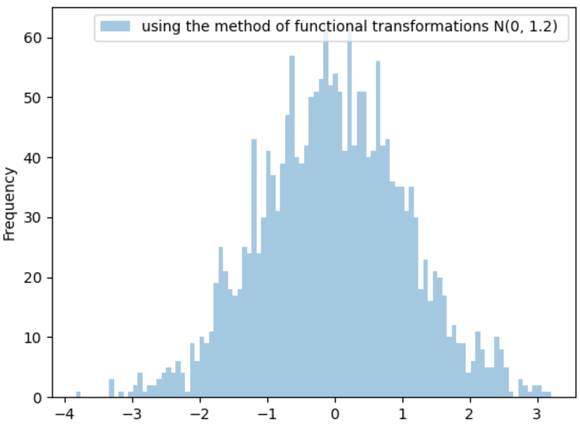
```python
#  проверка точности моделирования с помощью критерия серий
series_criterion(sample_using_the_central_limit_theorem_01)
```

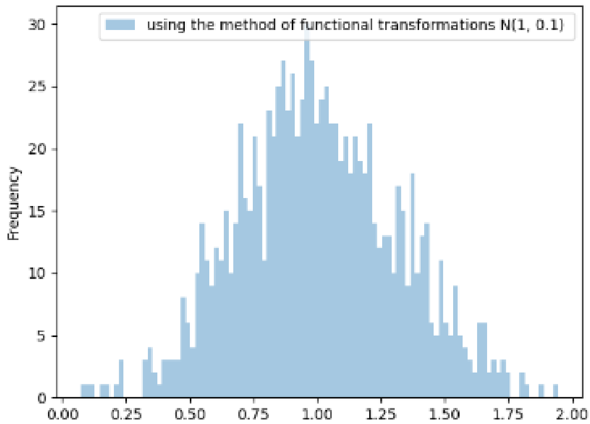
```python
# так как в предыдущем задании лучшие результаты были при использовании алгоритма основанного на методе
# функционального преобразования БСВ, будем использовать его для генерации выборок во втором задании
print('------------------------------------ "Truncated" Normal Distribution ------------------------------------')

sampling_using_method_of_functional_transformations(
    sample_of_truncated_normal_distribution_01, location_02_01, scale_02, n)
sample_of_truncated_normal_distribution_01 = \
    create_truncated_normal_distribution(sample_of_truncated_normal_distribution_01)
print('N(', location_02_01, ', ', variance_02, ') - доля пропущенных реализаций : ',
      1 - len(sample_of_truncated_normal_distribution_01)/n, sep='')

sampling_using_method_of_functional_transformations(
    sample_of_truncated_normal_distribution_02, location_02_02, scale_02, n)
sample_of_truncated_normal_distribution_02 = \
    create_truncated_normal_distribution(sample_of_truncated_normal_distribution_02)
print('N(', location_02_02, ', ', variance_02, ') - доля пропущенных реализаций : ',
      1 - len(sample_of_truncated_normal_distribution_02)/n, sep='')

sampling_using_method_of_functional_transformations(
    sample_of_truncated_normal_distribution_03, location_02_03, scale_02, n)
sample_of_truncated_normal_distribution_03 = \
    create_truncated_normal_distribution(sample_of_truncated_normal_distribution_03)
print('N(', location_02_03, ', ', variance_02, ') - доля пропущенных реализаций : ',
      1 - len(sample_of_truncated_normal_distribution_03)/n, sep='')
```

Результаты










```
-------------------------------------- Compare Time --------------------------------------
N(0, 1.2) - using the central limit theorem - time : 0.0038269990000117105
N(1, 0.1) - using the central limit theorem - time : 0.003765888999623712
------------------------------------------------------------------------------------------
N(0, 1.2) - using the method of functional transformations - time : 0.0007871759989939164
N(1, 0.1) - using the method of functional transformations - time : 0.0007771890013827942
------------------------------------------------------------------------------------------
Test 1 - the central limit theorem - N(0, 1.2)
Test passed !
Test 2 - the central limit theorem - N(1, 0.1)
Test passed !
Test 3 - the method of functional transformations - N(0, 1.2)
Test passed !
Test 4 - the method of functional transformations - N(1, 0.1)
Test passed !
----------------------------- "Truncated" Normal Distribution ----------------------------
N(3, 9) - доля пропущенных реализаций : 0.15700000000000003
N(6, 9) - доля пропущенных реализаций : 0.022000000000000002
N(9, 9) - доля пропущенных реализаций : 0.0020000000000000018
```

Выводы

В первой части задания алгоритм основанный на методе функционального преобразования БСВ дал более быстрый результат (примерно в 4 раз быстрее), оба алгоритма прошли проверку точности моделирования с помощью критерия серий, но анализ гистограммы частот распределения показал, что более точный результат дал алгоритм, основанный на методе функционального преобразования БСВ. Во второй части с ростом коэффициента сдвига, доля пропущенных реализаций СВ уменьшалась (от 0.157 при $\mu = 3$ до 0.002 при $\mu = 9$).