БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

**Лабораторная работа №2**

Выполнил: Данилов Валентин Юрьвевич

4 курс 4 группа

Преподаватель: Кирлица В.П.

Минск 2019

**Задание 14.**

1. **Постановка задачи**

Осуществить моделирование n=1000 реализаций CB ξ~t(m) Исследовать точность и быстродействие алгоритма моделирования. Положить m=1.

1. **Формулы и краткие пояснения к ним**

**Генерация стандартного нормального распределения**

Для генерации использовался алгоритм, основанный на ЦПТ, то есть, если имеется N независимо распределенных БСВ , то при стремлении N к бесконечности следующая СВ будет распределена асимптотически нормально:

Приемлемая на практике точность достигается при N=12.

**Генерация распределения Стьюдента (t-распределения)**

Для моделирования распределения Стьюдента можно воспользоваться следующим его свойством:

,

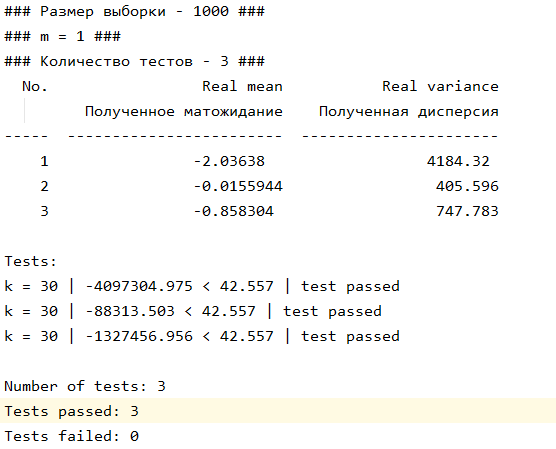
где – искомая СВ, имеющая распределение Стьюдента, (0,1) – СВ со стандартным нормальным распределением, а – СВ с распределением Хи-квадрат.

Моделирования Хи-квадрат распределения производится с использованием следующей формулы:

*,*

где – независимые СВ с стандартным распределением.

1. **Рузультаты**

****

1. **Исходный код**

**LAB2.PY**

**import** numpy **as** np  
**from** test **import** pearson  
**from** utils **import** format\_test\_result, report\_student  
**from** distribution **import** student, student\_distribution, next\_gauss\_standard  
  
TEST\_NUMBER = 3  
N = 1000  
m = 1  
  
indices = []  
test\_results = []  
theory\_means = []  
theory\_variances = []  
real\_means = []  
real\_variances = []  
  
**for** i **in** range(TEST\_NUMBER):  
 \_student\_selection = np.random.standard\_t(m, N)  
  
 indices.append(i + 1)  
 real\_mean = np.mean(\_student\_selection)  
 real\_means.append(real\_mean)  
 real\_variances.append(np.var(\_student\_selection))  
  
 pearson\_test\_result = pearson(sorted(\_student\_selection), distr\_f=student\_distribution)  
 test\_result = format\_test\_result(\*pearson\_test\_result)  
 test\_results.append(test\_result)  
  
headers = [**"No."**, **"Real mean\nПолученное матожидание"**, **"Real variance\nПолученная дисперсия"**,  
 **"Theory mean\nТеоретическое матожидание"**, **"Theory variance\nТеоретическая дисперсия"**]  
rows = np.c\_[indices, real\_means, real\_variances]  
report\_student(**"lab2\_report.txt"**, headers, rows, test\_results, N, m)

**TEST.PY**

**import** numpy **as** np  
**from** bisect **import** bisect\_right  
  
epsilon = 0.05  
MAX\_K = 30  
  
DELTA = {  
 2: 3.841,  
 3: 5.991,  
 4: 7.815,  
 5: 9.488,  
 6: 11.070,  
 7: 12.592,  
 8: 14.067,  
 9: 15.507,  
 10: 16.919,  
 11: 18.307,  
 12: 19.675,  
 13: 21.026,  
 14: 22.362,  
 15: 23.685,  
 16: 24.996,  
 17: 26.296,  
 18: 27.587,  
 19: 28.869,  
 20: 30.144,  
 21: 31.410,  
 22: 32.671,  
 23: 33.924,  
 24: 35.172,  
 25: 36.415,  
 26: 37.652,  
 27: 38.885,  
 28: 40.113,  
 29: 41.337,  
 30: 42.557  
}  
  
  
**def** get\_frequency(sorted\_seq, k):  
 min\_el = sorted\_seq[0]  
 max\_el = sorted\_seq[-1]  
 step = float((max\_el - min\_el) / (k + 1))  
 segments = np.arange(min\_el, max\_el, step)  
 v = [0] \* k  
 last\_position = 0  
 **for** i **in** range(k):  
 position = bisect\_right(sorted\_seq, segments[i + 1])  
 v[i] = position - last\_position  
 last\_position = position  
 **return** v, segments  
  
  
**def** get\_probabilities(segments, f):  
 k = len(segments)  
 p = [0] \* k  
 **for** i **in** range(k - 1):  
 p[i] = f(1, segments[i + 1]) - f(1, segments[i])  
 **return** p  
  
  
**def** pearson(sorted\_seq, distr\_f):  
 n = len(sorted\_seq)  
 k = MAX\_K  
 v, segments = get\_frequency(sorted\_seq, k)  
 p = get\_probabilities(segments, distr\_f)  
 delta = DELTA[k]  
 value = sum([(v[i] - n \* p[i]) \*\* 2 / (n \* p[i]) **for** i **in** range(k)])  
 **return** value, delta, k

**UTILS.PY**

**from** tabulate **import** tabulate  
**import** io  
  
  
**def** multiply\_list(list):  
 *# Multiply elements one by one* result = 1  
 **for** x **in** list:  
 result = result \* x  
 **return** result  
  
  
TEST\_TEMPLATE = **"{value:.3f} {sign} {delta:.3f} | test {result}"**RESULT\_TEMPLATE = **"\nNumber of tests: {}\nTests passed: {}\nTests failed: {}"**HEADING\_TEMPLATE\_STUDENT = **"### Размер выборки - {} ###\n### m = {} ###\n### Количество тестов - {} ###\n"  
  
  
def** get\_test\_result(value, delta):  
 **return** [**'<'**, **'passed'**] **if** value < delta **else** [**'>='**, **'failed'**]  
  
  
**def** format\_test\_result(value, delta, k=**None**):  
 template = TEST\_TEMPLATE  
 **if** k:  
 template = **'k = {k} | '** + TEST\_TEMPLATE  
 sign, result = get\_test\_result(value, delta)  
 **return** template.format(k=k, value=value, sign=sign, delta=delta, result=result)  
  
  
**def** report\_student(filename, headers, values, test\_results, selection\_size, m, mode=**"w+"**):  
 failed = 0  
 passed = 0  
 **for** test\_result **in** test\_results:  
 **if "failed" in** test\_result:  
 failed = failed + 1  
 **else**:  
 passed = passed + 1  
  
 **with** io.open(filename, mode, encoding=**"utf-8"**) **as** file:  
 file.write(HEADING\_TEMPLATE\_STUDENT.format(selection\_size, m, len(test\_results)))  
 file.write(  
 tabulate(  
 values, headers  
 )  
 )  
 file.write(**"\n\nTests: \n"**)  
 **for** test\_result **in** test\_results:  
 file.write(test\_result)  
 file.write(**"\n"**)  
 file.write(RESULT\_TEMPLATE.format(len(test\_results), passed, failed))