Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

**Отчёт**

**по практической работе №5**

**«Выборки и их характеристики»**

по дисциплине «Теория вероятностей»

вариант 15

Выполнил: Хоробрых Д.Е., группа P3216

Преподаватель: Селина Е.Г.

Санкт-Петербург

~ 2023 ~

1. **Текст задания**

Каждый студент получает выборку из 20 чисел. Необходимо определить следующие статистические характеристики: вариационный ряд, экстремальные значения и размах, оценки математического ожидания и среднеквадратического отклонения, эмпирическую функцию распределения и её график, гистограмму и полигон приведенных частот группированной выборки. Для расчета характеристик и построения графиков нужно написать программу на одном из языков программирования. Листинг программы и результаты работы должны быть представлены в отчете по практической работе.



1. **Код программы**

App.java

1. **import** com.github.sh0nk.matplotlib4j.PythonExecutionException;
3. **import** java.io.IOException;
4. **import** java.util.List;
6. **public** **class** App {
7. **public** **static** **void** main(String[] args) **throws** PythonExecutionException, IOException {
8. List<Double> elements = List.of(1.07, -1.02, -1.18, 1.34, 1.69, 0.31, 0.48, 0.11, 0.92, 0.04, 1.42, -1.59, -0.08, -0.21, 0.65, 0.55, 0.66, 1.22, 0.46, 0.82);
9. Task task = **new** Task(elements);
11. System.out.print("Вариационный ряд:**\t**");
12. System.out.println(task.getVariationSeries());
14. System.out.print("Экстремальные значения:**\t**");
15. **System.out.println(task.getExtremeValues());**
17. System.out.print("Размах выборки:**\t**");
18. System.out.printf("%.2f**\n**", task.getSampleRange());
20. **System.out.print("Оценка математического ожидания:\t");**
21. System.out.printf("%.2f**\n**", task.getExpectedValue());
23. System.out.print("Оценка дисперсии:**\t**");
24. System.out.printf("%.2f**\n**", task.getDispersion());
26. System.out.print("Оценка среднеквадратического отклонения:**\t**");
27. System.out.printf("%.2f**\n**", task.getStandardDeviation());
29. System.out.print("Мода ряда:**\t**");
30. **System.out.println(task.getModes());**
32. System.out.print("Медиана ряда:**\t**");
33. System.out.printf("%.2f**\n**", task.getMedian());
35. **System.out.println("Статистический ряд:\t");**
36. task.printStatisticSeries(task.getStatisticSeries());
38. System.out.println("Эмпирическая функция распределения:**\t**");
39. task.printEmpiricalDistributionFunction(task.getEmpiricalDistributionFunction());
40. **System.out.println();**
41. System.out.println();
42. System.out.println();
43. task.showFrequency();
44. }
45. **}**

Task.java

1. **import** com.github.sh0nk.matplotlib4j.Plot;
2. **import** com.github.sh0nk.matplotlib4j.PythonExecutionException;
3. **import** lombok.\*;
5. **import java.io.IOException;**
6. **import** java.util.\*;
8. **import** static java.lang.Math.pow;
9. **import** static java.lang.Math.sqrt;
10. **import static java.util.Collections.max;**
11. **import** static java.util.Collections.min;
13. @Getter
14. @Setter
15. **@RequiredArgsConstructor**
16. **public** **class** Task {
17. **public** **final** List<Double> elements;
19. **public** List<Double> getVariationSeries() {
20. ***// sorted collection***
21. **return** elements.stream().sorted().toList();
22. }
24. **public** List<Double> getExtremeValues() {
25. ***// [minimum, maximum]***
26. **return** List.of(min(elements), max(elements));
27. }
29. **public** Double getSampleRange() {
30. ***// maximum - minimum***
31. List<Double> extremeValues = getExtremeValues();
32. Double minimum = extremeValues.get(0);
33. Double maximum = extremeValues.get(1);
34. **return** maximum - minimum;
35. **}**
37. **public** Double getExpectedValue() {
38. *// MX = sum(x \* n\_x)*
39. **return** elements.stream()
40. **.mapToDouble(x -> elementFrequency(x) \* x)**
41. .sum();
42. }
44. **private** Double elementFrequency(Double element) {
45. ***// n = n\_i / n***
46. **return** ((**double**) elements.stream()
47. .filter(x -> Objects.equals(x, element))
48. .count()) / elements.size();
49. }
51. **public** Double getDispersion() {
52. *// DX = MX^2 - (MX)^2*
53. Double expectedSquareValue = elements.stream()
54. .mapToDouble(x -> elementFrequency(x) \* pow(x, 2))
55. **.sum();**
56. Double squaredExpectedValue = pow(getExpectedValue(), 2);
57. **return** expectedSquareValue - squaredExpectedValue;
58. }
60. **public Double getStandardDeviation() {**
61. *// oX = sqrt(DX)*
62. **return** sqrt(getDispersion());
63. }
65. **public Map<Double, Double> getEmpiricalDistributionFunction() {**
66. *// F\*\_n(x) = p\*{X < x}*
67. *// F\*\_n(x) = n\_x / n*
68. TreeMap<Double, Double> result = **new** TreeMap<>(); *// TreeMap для удобства расположения ключей в порядке возрастания*
70. **for (Double currentElement : elements) {**
71. result.put(currentElement, calculateFrequencyFuncForElement(currentElement));
72. }
73. **return** result;
74. }
76. **private** Double calculateFrequencyFuncForElement(Double element) {
77. **double** quantity = (**double**) elements.stream()
78. .filter(x -> x < element)
79. .count();
80. ***// считаем их частость (относительную частоту)***
81. **return** quantity / elements.size();
82. }
84. **private** Integer getElementQuantity(Double element) {
85. **return Math.toIntExact(elements.stream().filter(x -> x.equals(element)).count());**
86. }
88. **public** List<Double> getModes() {
89. *// Ищем максимальную частоту*
90. **Integer maxN = max(elements.stream().map(this::getElementQuantity).toList());**
91. *// Оставляем только те значения, которые имеют максимальную частоту*
92. **return** getVariationSeries().stream().filter(x -> getElementQuantity(x).equals(maxN)).toList();
93. }
95. **public Double getMedian() {**
96. **if** (elements.size() % 2 == 0) {
97. **return** (getVariationSeries().get(elements.size() / 2 - 1) + getVariationSeries().get(elements.size() / 2)) / 2;
98. } **else** {
99. **return** getVariationSeries().get(elements.size() / 2 - 1);
100. **}**
101. }
103. **public** Double calculateElementsInRange(Double x0, Double x1) {
104. **return** (**double**) elements.stream().filter(x -> x0 <= x && x < x1).count();
105. **}**
107. **public** List<List<Double>> getStatisticSeries() {
108. Double m = Math.ceil(1 + Math.log(elements.size()) / Math.log(2)); *// число интервалов*
109. **double** h = getSampleRange() / m;
110. **Double x0 = min(elements);**
111. List<List<Double>> result = **new** ArrayList<>();
112. **for** (**int** i = 0; i < m; i++) {
113. Double \_x0 = x0 + h \* i;
114. Double \_x1 = x0 + h \* (i + 1);
115. **result.add(List.of(\_x0, \_x1, calculateElementsInRange(\_x0, \_x1)));**
116. }
117. **return** result;
118. }
120. **public void printEmpiricalDistributionFunction(Map<Double, Double> func) {**
121. Double lastValue = **null**;
122. **for** (Double key : func.keySet()) {
123. **if** (lastValue == **null**) {
124. System.out.printf("**\t**-∞ < X <= %.2f**\t**||**\t**", key);
125. **} else {**
126. System.out.printf("%.2f < X <= %.2f**\t**||**\t**", lastValue, key);
127. }
128. lastValue = key;
129. }
130. **System.out.printf("%.2f < X < +∞\t||\t", lastValue);**
131. System.out.print("**\n\t**");
133. **for** (Double value : func.values()) {
134. System.out.printf("**\t**%.2f**\t\t**||**\t\t**", value);
135. **}**
136. System.out.printf("**\t**%.2f**\t**||**\t\t\t**", 1.0);
137. System.out.println();
138. }
140. **public void printStatisticSeries(List<List<Double>> statistic) {**
141. List<List<Double>> ranges = statistic.stream().map(x -> List.of(x.get(0), x.get(1))).toList();
142. List<Double> values = statistic.stream().map(x -> x.get(2)).toList();
144. **for** (List<Double> range : ranges) {
145. **Double start = range.get(0);**
146. Double end = range.get(1);
147. System.out.printf("%.2f <= X < %.2f**\t**||**\t**", start, end);
148. }
149. System.out.println();
150. **for (Double value : values) {**
151. System.out.printf("**\t**%.2f**\t\t\t**||**\t**", value);
152. }
153. System.out.println();

156. }
158. **public** **void** showFrequency() **throws** PythonExecutionException, IOException {
159. Plot plt = Plot.create();
160. **plt.subplot(2, 2, 1);**
161. List<Double> x1 = getStatisticSeries().stream().map(x -> (x.get(0) + x.get(1)) / 2).toList();
162. List<Double> y1 = getStatisticSeries().stream().map(x -> x.get(2)).toList();
163. plt.plot()
164. .add(x1, y1)
165. **.linestyle("-");**
166. plt.xlabel("x");
167. plt.ylabel("n\_i");
168. plt.title("Полигон частот");
170. **plt.subplot(2, 2, 2);**
171. List<Double> x2 = getStatisticSeries().stream().map(x -> (x.get(0) + x.get(1)) / 2).toList();
172. List<Double> y2 = getStatisticSeries().stream().map(x -> x.get(2) / elements.size()).toList();
173. plt.plot()
174. .add(x2, y2)
175. **.linestyle("-");**
176. plt.xlabel("x");
177. plt.ylabel("p\_i");
178. plt.title("Полигон частостей");
180. **plt.subplot(2, 2, 3);**
181. List<Double> x3 = getVariationSeries();
182. List<Double> y3 = getVariationSeries().stream().mapToDouble(**this**::calculateFrequencyFuncForElement).boxed().toList();
183. plt.plot().add(List.of(-100, min(x3)), List.of(0, 0), "r-");
184. plt.plot().add(List.of(max(x3), 1000000000), List.of(1, 1), "r-");
185. **plt.plot().add(List.of(max(x3)), List.of(1), "r<");**
186. plt.plot().add(List.of(min(x3)), List.of(0), "ro");
188. *// Рисуем основную функцию*
189. **for** (**int** i = 0; i < elements.size() - 1; i++) {
190. **Double \_x1 = x3.get(i);**
191. Double \_x2 = x3.get(i + 1);
192. Double \_y1 = y3.get(i + 1);
193. Double \_y2 = y3.get(i + 1);
194. plt.plot()
195. **.add(List.of(\_x1, \_x2), List.of(\_y1, \_y2), "r-");**
196. plt.plot()
197. .add(List.of(\_x1), List.of(\_y1), "r<");
198. plt.plot()
199. .add(List.of(\_x2), List.of(\_y2), "ro");
200. **}**
201. plt.xlabel("x");
202. plt.ylabel("y");
203. plt.xlim(min(getVariationSeries()) - 0.5, max(getVariationSeries()) + 0.5);
204. plt.ylim(-0.1, 1.1);
205. **plt.title("Эмпирическая функция распределения");**
207. plt.subplot(2, 2, 4);
208. List<List<Double>> data = getStatisticSeries();
209. List<Double> x4 = **new** ArrayList<>(data.stream().map(x -> x.get(0)).toList());
210. **x4.add(data.get(data.size() - 1).get(1));**
211. List<Double> y4 = data.stream().map(x -> x.get(2)).toList();
212. plt.hist()
213. .add(getVariationSeries()).bins(x4);
214. plt.title("Гистограмма частот статистического ряда");
215. **plt.show();**
216. }
217. }
219. **Вывод программы**

Вариационный ряд: [-1.59, -1.18, -1.02, -0.21, -0.08, 0.04, 0.11, 0.31, 0.46, 0.48, 0.55, 0.65, 0.66, 0.82, 0.92, 1.07, 1.22, 1.34, 1.42, 1.69]

Экстремальные значения: [-1.59, 1.69]

Размах выборки: 3,28

Оценка математического ожидания: 0,38

Оценка дисперсии: 0,73

Оценка среднеквадратического отклонения: 0,85

Мода ряда: [-1.59, -1.18, -1.02, -0.21, -0.08, 0.04, 0.11, 0.31, 0.46, 0.48, 0.55, 0.65, 0.66, 0.82, 0.92, 1.07, 1.22, 1.34, 1.42, 1.69]

Медиана ряда: 0,52

Статистический ряд:

-1,59 <= X < -1,04 || -1,04 <= X < -0,50 || -0,50 <= X < 0,05 || 0,05 <= X < 0,60 || 0,60 <= X < 1,14 || 1,14 <= X < 1,69 ||

2,00 || 1,00 || 3,00 || 5,00 || 5,00 || 4,00 ||

Эмпирическая функция распределения:

-∞ < X <= -1,59 || -1,59 < X <= -1,18 || -1,18 < X <= -1,02 || -1,02 < X <= -0,21 || -0,21 < X <= -0,08 || -0,08 < X <= 0,04 || 0,04 < X <= 0,11 || 0,11 < X <= 0,31 || 0,31 < X <= 0,46 || 0,46 < X <= 0,48 || 0,48 < X <= 0,55 || 0,55 < X <= 0,65 || 0,65 < X <= 0,66 || 0,66 < X <= 0,82 || 0,82 < X <= 0,92 || 0,92 < X <= 1,07 || 1,07 < X <= 1,22 || 1,22 < X <= 1,34 || 1,34 < X <= 1,42 || 1,42 < X <= 1,69 || 1,69 < X < +∞ ||

0,00 || 0,05 || 0,10 || 0,15 || 0,20 || 0,25 || 0,30 || 0,35 || 0,40 || 0,45 || 0,50 || 0,55 || 0,60 || 0,65 || 0,70 || 0,75 || 0,80 || 0,85 || 0,90 || 0,95 || 1,00 ||

Изображение выглядит как текст, диаграмма, линия, График

Автоматически созданное описание

1. **Вывод**

В ходе выполнения практической работы были рассмотрены основные статистические характеристики выборки, а также методы их нахождения. В результате был написан список методов, позволяющих вычислить ту или иную характеристику выборки, на языке Java, а также разработан код, выводящий графики полигона частот и частостей, эмпирической функции распределения и гистограммы частот статистического ряда в виде графиков.