Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский Государственный Университет Телекоммуникаций и Информатики»

Кафедра вычислительных систем

Лабораторная работа
по дисциплине «Моделирование»

Выполнили: студенты 4 курса группы ИС-941

Патрушев Н.В.

Сизов М.А.

Проверил: старший преподаватель кафедры ВС

Петухова Я. В.

Новосибирск, 2023

Оглавление

Постановка задачи

Сгенерировать набор чисел и проанализировать распределение при помощи критерия Пирсона и автокорреляции.

Краткая теория

Критерий Пирсона

Проверка гипотезы по критерию Пирсона.

1. Для проверки гипотезы вычисляют теоретические частоты и находят $\chi^2_{\text{набл}} = \sum \frac{(n_i - n_i')^2}{n_i'}$.
2. По таблице критических точек распределения χ^2 по заданному уровню значимости α и числу степеней свободы k находят $\chi^2_{\text{крит}}(\alpha, k)$.
3. Если $\chi^2_{\text{набл}} < \chi^2_{\text{крит}}$ то нет оснований отвергать гипотезу, если не выполняется данное условие - то отвергают.

Функция плотности равномерного распределения величины X имеет вид $f(x) = \frac{1}{b-a} x \in [a, b]$.

Для того, чтобы при уровне значимости α проверить гипотезу о том, что непрерывная случайная величина распределена по равномерному закону, требуется:

1. Найти по заданному эмпирическому распределению выборочное среднее

   $\overline{x_b}$ и $\sigma_b = \sqrt{D_b}$ и принять в качестве оценки параметров a и b величины $a = \overline{x_b} - \sqrt{3} * \sigma_b, b = \overline{x_b} + \sqrt{3} * \sigma_b$
2. Найти вероятность попадания случайной величины X в частичные интервалы (x_i, x_{i+1}) по формуле:

   $P_i = P(x_i < X < x_{i+1})$

   $P_i = F(x_{i+1}) - F(x_i) = \frac{x_{i+1}}{b-a} - \frac{x_i}{b-a}$
3. Найти теоретические (выравнивающие) частоты по формуле: $n_i' = n * p_i$.
4. Приняв число степеней свободы $k = S - 3$ и уровень значимости α=0,01 по таблицам χ^2 найдём $\chi^2_{\text{крит}}$ по заданным α и k, $\chi^2_{\text{крит}}(\alpha, k)$.
5. По формуле $\chi^2_{\text{набл}} = \sum \frac{(n_i - n_i')^2}{n_i'}$, где n_i - эмпирические частоты, находим наблюдаемое значение $\chi^2_{\text{набл}}$.
6. Если $\chi^2_{\text{набл}} < \chi^2_{\text{крит}}$ – нет оснований, отвергать гипотезу.

Автокорреляция

Пусть τ – смещение.

Коэффициент автокорреляции:

$$a(\tau) = \frac{\sum_{t=1}^{n-\tau} (x_t - \overline{x}) * (x_{t+\tau} - \overline{x})}{\varsigma^2 (n - \tau)}$$

, где $\varsigma^2 = \frac{1}{n}\sum_{t=1}^{n}(x_t - \overline{x})^2$, $\overline{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$

Ход работы

Генерация 100000 чисел и замер частот выпадения:

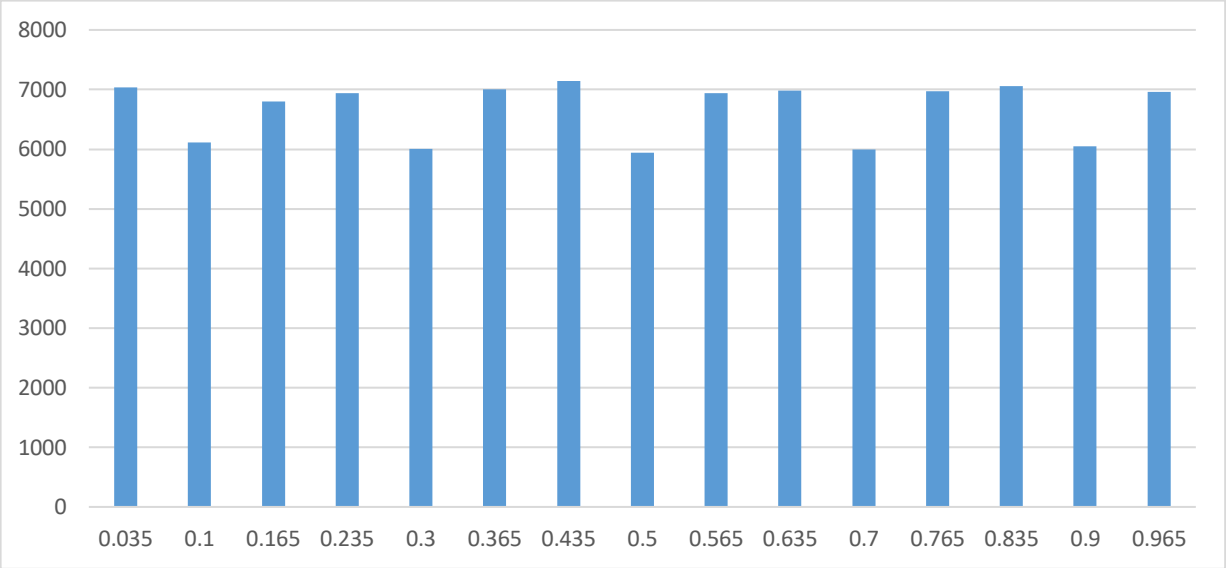

Рис 1. Гистограмма частот выпадения СВ

Критерий согласия Пирсона

Проверка гипотезы о равномерном распределении.

Критерий α=0.01. Набор состоит из 120 чисел.

Расчёт k.

k = 8-2-1 и h = 0,125.

xi	ni
0.065	17
0.19	14
0.315	13
0.44	12
0.565	13
0.69	16
0.815	20
0.94	15

Рис 2. Интервалы и количество чисел, попавших в них.

$\chi^2_{\text{набл}}$:

```
Хи-квадрат: 2.847222222222222
```

Рис 3. Результат выполнения программы.

Отвергать гипотезу нет оснований, т.к $\chi^2_{\text{набл}} < \chi^2_{\text{крит}}$ ($\chi^2_{\text{крит}} = 15{,}086$).

Автокорреляция

Смещение τ = 2. Коэффициент автокорреляции:

Рис 4. Коэффициент автокорреляции при смещении τ = 2.

Коэффициент *r* от -1 до 1. Можно считать, что автокорреляция отсутствует, т. к. выборочный коэффициент автокорреляции незначимо отличается от 0.

Вывод

Произведена проверка гипотезы о равномерном распределении.

Значение критерия Пирсона $\chi^2_{\text{набл}} = 2.84722$ меньше чем $\chi^2_{\text{крит}} = 15{,}086$ при уровне значимости α=0.01 и степенях свободы равных 5. Коэффициент автокорреляции находится в диапазоне от -1 до 1 и равен -0.27777.

На основании полученных данных отвергать гипотезу нет оснований.

Приложение

Index.ts

```
const N = 120;
const map: Map<Array<number>, Array<number>> = new Map();

const calcStep = (): number => {
  let k = Math.round(Math.log10(N) * 3.322 + 1);

  k = k < 5 ? 5 : k;
  k = k > 15 ? 15 : k;
  return 1 / k;
};

const generateIntervals = (intervals: Map<Array<number>, number>) => {
  const k = calcStep();
  for (let i = 0; i <= 1 - k; i += k) {
    intervals.set([+i.toFixed(2), +(i + k).toFixed(2)], 0);
  }
};

const calcHits = () => {
  const intervals: Map<Array<number>, number> = new Map();

  generateIntervals(intervals);

  for (let i = 0; i < N; i++) {
    const number: number = +Math.random().toFixed(3);

    for (const interval of intervals.keys()) {
      if (interval[0] < number && number <= interval[1]) {
        const intervalCountNumber = intervals.get(interval);

        if (intervalCountNumber !== undefined) {
          intervals.set(interval, intervalCountNumber + 1);
        }
      }
    }
  }

  for (const [interval, count] of intervals.entries()) {
    map.set(interval, [count]);
  }
};

const calcPi = () => {
  for (const [interval, props] of map.entries()) {
    const pi = interval[1] - interval[0];
    props.push(pi);

    map.set(interval, props);
  }
};

const calcNih = () => {
  for (const [interval, props] of map.entries()) {
```

```
    const pi = props.at(-1);

    if (pi) {
      props.push(pi * N);
      map.set(interval, props);
    }
  }
};

const calcDiff = () => {
  for (const [interval, props] of map.entries()) {
    const ni = props.at(0);

    if (ni) {
      const ni_h = props.at(-1);

      if (ni_h) {
        const diff = ni - ni_h;
        props.push(diff);
        map.set(interval, props);
      }
    }
  }
};

const calcSqrt = () => {
  for (const [interval, props] of map.entries()) {
    const diff = props.at(-1);

    if (diff) {
      const diff2 = diff ** 2;
      props.push(diff2);

      map.set(interval, props);
    }
  }
};

const calcSqrtDivNih = () => {
  for (const [interval, props] of map.entries()) {
    const ni_h = props.at(-3);
    if (ni_h) {
      const s = props.at(-1);

      if (s) {
        const sqrtDivNiH = s / ni_h;

        props.push(sqrtDivNiH);
        map.set(interval, props);
      }
    }
  }
};

const calcHi = () => {
  let hiSqrt = 0;

  for (const props of map.values()) {
```

```
    const hi = props.at(-1);
    if (hi) hiSqrt += hi;
  }

  return hiSqrt;
};

const calcMathWait = () => {
  let mathWait = 0;
  for (const [interval, props] of map.entries()) {
    const ni = props[0];
    mathWait += ni;
  }

  return mathWait / Array.from(map.values()).length;
};

const calcDisp = (mathWait: number) => {
  let disp = 0;
  for (const [interval, props] of map.entries()) {
    disp += Math.pow(props[0] - mathWait, 2);
  }

  return disp / Array.from(map.values()).length;
};

const autocorrelation = () => {
  const t = 2;
  const mathWait = calcMathWait();
  const disp = calcDisp(mathWait);

  let correlation = 0;
  const nis = Array.from(map.values());
  for (let i = 0; i < nis.length - t; i++) {
    correlation +=
      ((nis[i][0] - mathWait) * (nis[i + t][0] - mathWait)) /
      (disp * (nis.length - t));
  }

  return correlation;
};

const hiSqrt = () => {
  calcHits();

  calcPi();
  calcNih();
  calcDiff();
  calcSqrt();
  calcSqrtDivNih();

  const hi = calcHi();

  return hi;
};

const bootstrap = () => {
  const hi = hiSqrt();
```

```
170   const corr = autocorrelation();
171
172   console.log(map);
173   console.log("\nХи-квадрат:", hi, "\nАвтокорреляция: ", corr);
174 };
175
176 bootstrap();
```