MATH301

Fall 2020

Mini Project

Instructor
Bülent Yılmaz

Maker
Nazmi Yılmaz

Introduction

These simulations are made with code samples written in javascript. All data sets are automatically converted into .xlsx files and .json files. The data sets are not included in this report for simpleness of the document. You can visit the following repository to get all data sets as excel files and json files.

GitHub > https://github.com/nazmi-yilmaz/math301-mini-project

Action 1

- **Javascript code:**

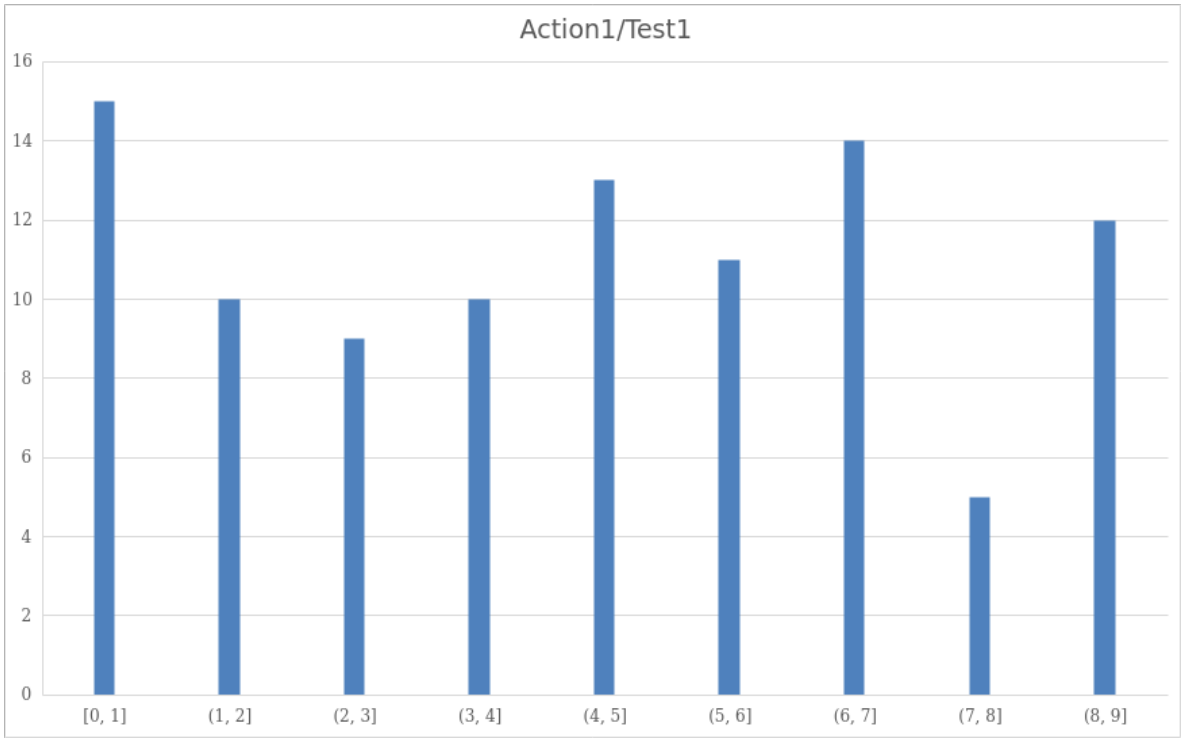
```
const fs = require('fs')
const XLSX = require('xlsx')

// Generates excel file from the dataset
function saveRecords(dest, cols, meta) {
if (!fs.existsSync(dest)) {
fs.mkdirSync(dest, { recursive: true })
}
// Write meta data
fs.writeFileSync(`${dest}/meta.json`, JSON.stringify(meta))
// Write .xlsx file
let sheet = XLSX.utils.json_to_sheet(cols)
let wb = XLSX.utils.book_new()
XLSX.utils.book_append_sheet(wb, sheet, 'Default')
XLSX.writeFile(wb, `${dest}/data.xlsx`)
}

// Generates an array of random integers with the specified size
function randomIntegerArray(lowerBound, upperBound, size) {
return [...Array(size - 1)].map((i) => {
let f = Math.random()
return Math.floor(f * (upperBound - lowerBound + 1) + lowerBound)
})
}

// Calculates the mean
function getMean(dataset) {
return dataset.reduce((a, b) => a + b) / dataset.length
}

// Calculates the variance
function getVariance(dataset) {
let sum = 0
let mean = getMean(dataset)
for (i of dataset) {
let val = i - mean
sum = sum + val * val
}
return sum / (dataset.length - 1)
}
```

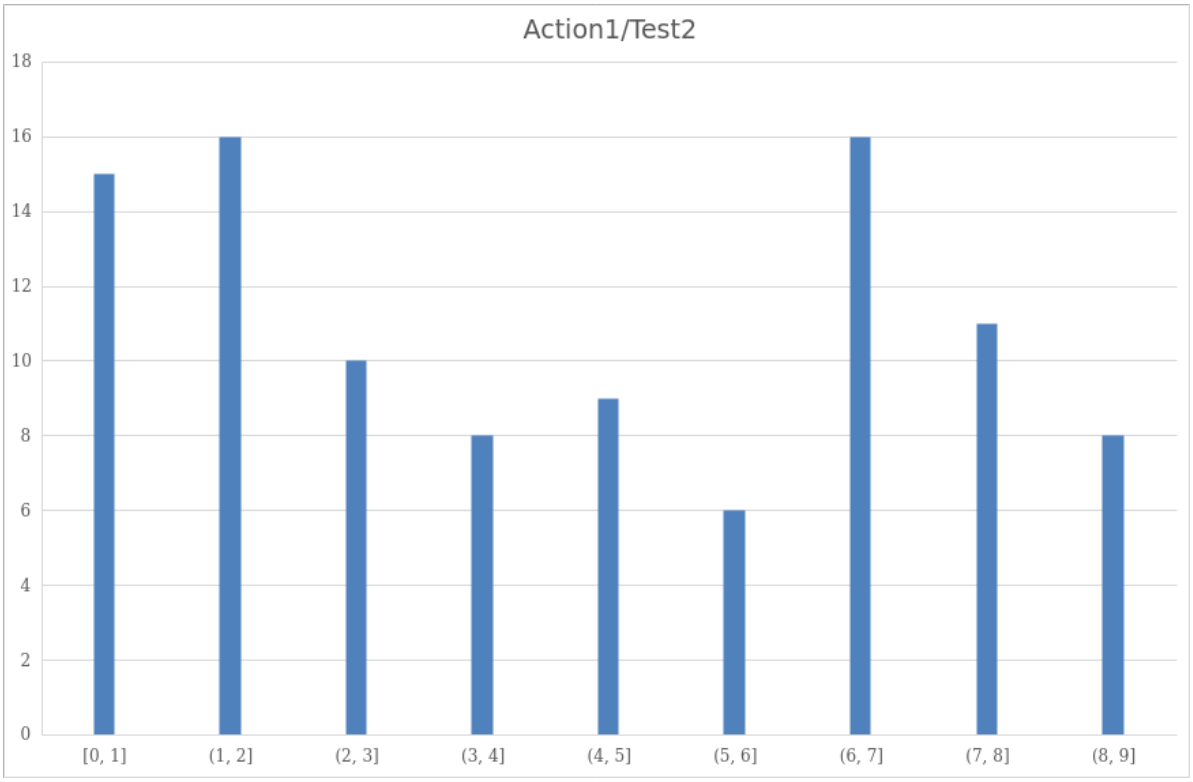
```
// Calculates the standard deviation
function getStandardDeviation(dataset) {
return Math.sqrt(getVariance(dataset))
}

// Function for single test
function test(number) {
let nums = randomIntegerArray(1, 9, 100)
let meta = {
mean: getMean(nums),
variance: getVariance(nums),
standard_deviation: getStandardDeviation(nums),
}
console.log(meta)

let rows = [...nums].map((n) => {
return { n: n }
})
saveRecords(`./data/action1/test${number}`, rows, meta)
}

// Simulates 5 test cases
function simulate() {
for (let i = 1; i <= 5; i++) {
test(i)
}
}

simulate();
```

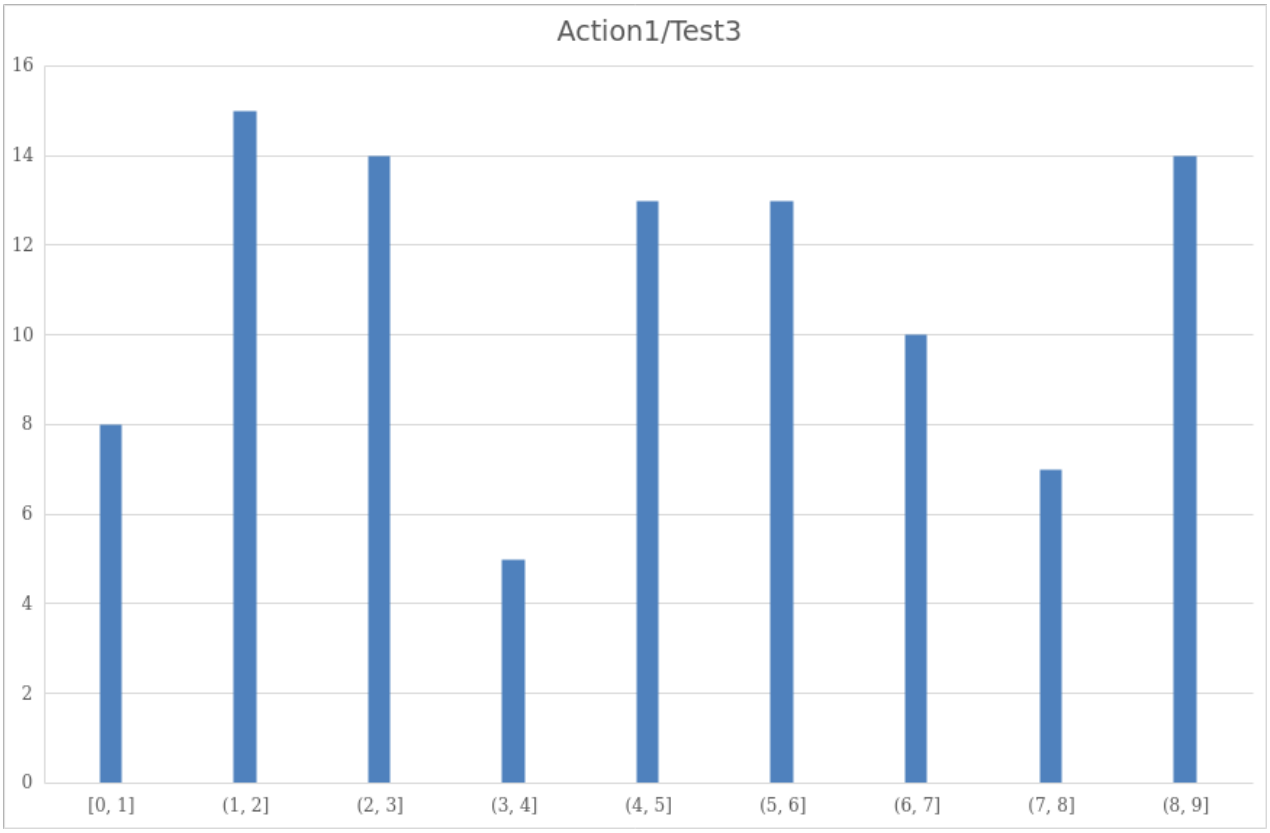
- **Table for Results**

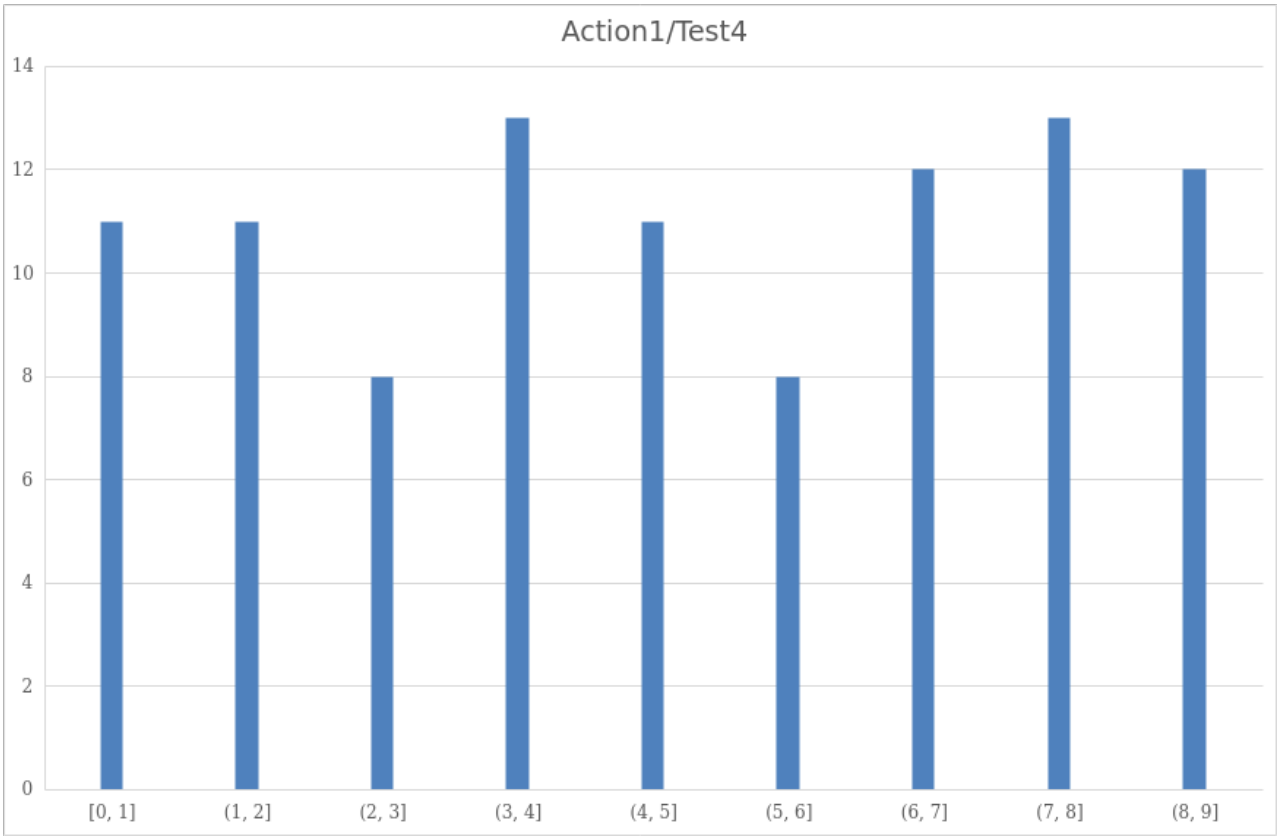
Test/Prop	Mean	Standard Deviation	Variance
1	4.838	2.629	6.912
2	4.666	2.706	7.326
3	5.000	2.602	6.775
4	5.13	2.640	6.972
5	4.585	2.539	6.449

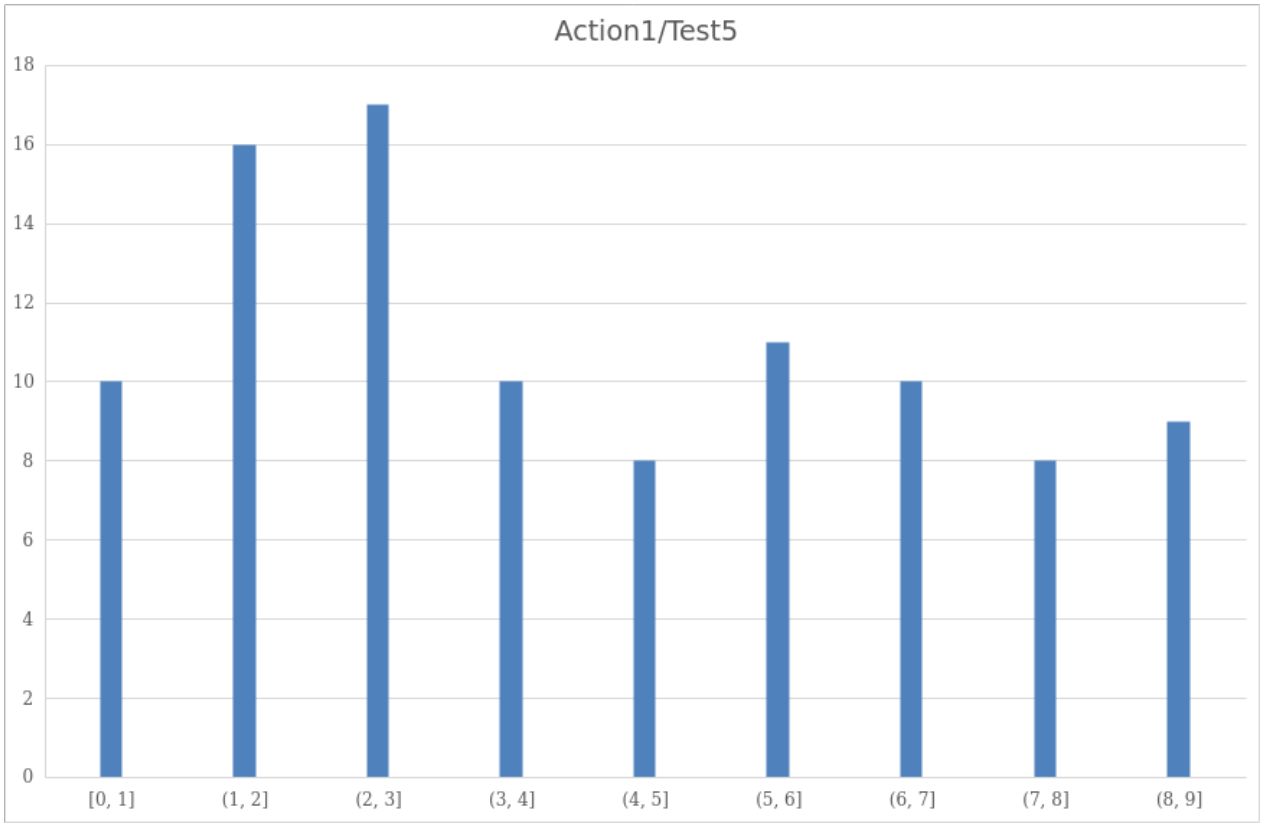
- **Histograms for Test Cases**






Action1/Test3
16
14
12
10
8
6
4
2
0
[0, 1]
(1, 2]
(2, 3]
(3, 4]
(4, 5]
(5, 6]
(6, 7]
(7, 8]
(8, 9]


Action1/Test4
14
12
10
8
6
4
2
0
[0, 1]
(1, 2]
(2, 3]
(3, 4]
(4, 5]
(5, 6]
(6, 7]
(7, 8]
(8, 9]


Action1/Test5
18
16
14
12
10
8
6
4
2
0
[0, 1]
(1, 2]
(2, 3]
(3, 4]
(4, 5]
(5, 6]
(6, 7]
(7, 8]
(8, 9]

Action 2

- **Javascript code:**

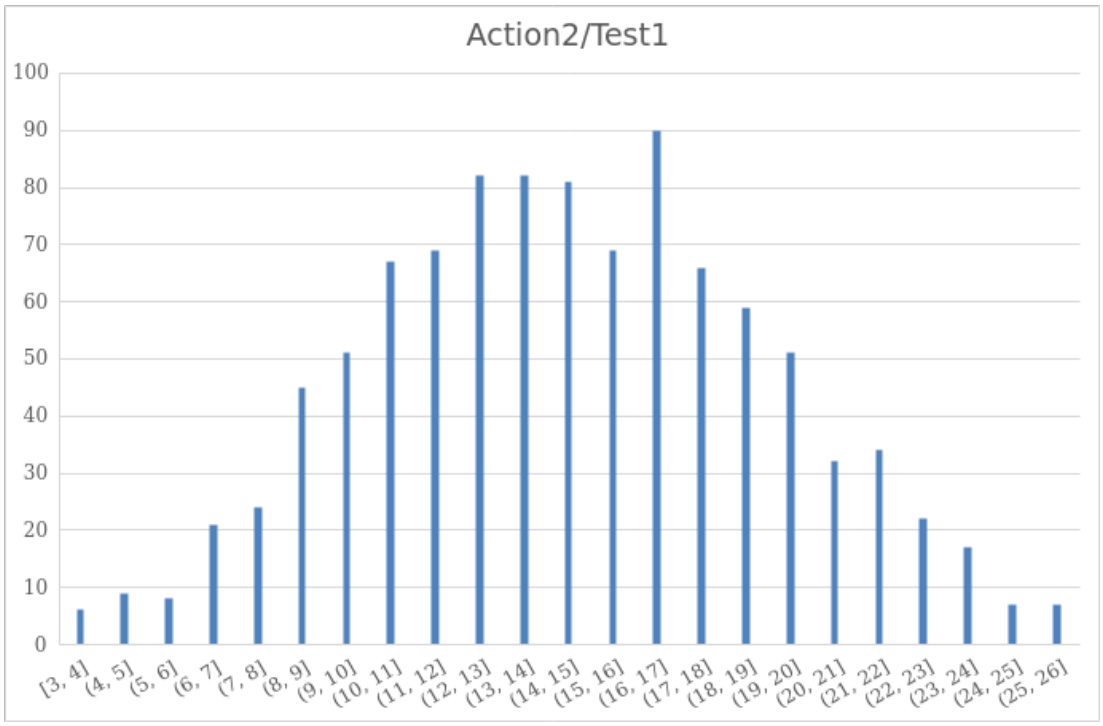
```
const fs = require('fs')
const XLSX = require('xlsx')

// Generates excel file from the dataset
function saveRecords(dest, cols, meta) {
      if (!fs.existsSync(dest)) {fs.mkdirSync(dest, { recursive: true })}
      // Write meta data
      fs.writeFileSync(`${dest}/meta.json`, JSON.stringify(meta))
      // Write .xlsx file
      let sheet = XLSX.utils.json_to_sheet(cols)
      let wb = XLSX.utils.book_new()
      XLSX.utils.book_append_sheet(wb, sheet, 'Default')
      XLSX.writeFile(wb, `${dest}/data.xlsx`)
}

// Generates an array of random integers with the specified size
function randomIntegerArray(lowerBound, upperBound, size) {
      return [...Array(size - 1)].map((i) => {
      let f = Math.random()
      return Math.floor(f * (upperBound - lowerBound + 1) + lowerBound)
})
}
// Calculates the mean
function getMean(dataset) {
      return dataset.reduce((a, b) => a + b) / dataset.length
}
// Calculates the variance
function getVariance(dataset) {
      let sum = 0
      let mean = getMean(dataset)
      for (i of dataset) {
      let val = i - mean
      sum = sum + val * val
      }
      return sum / (dataset.length - 1)
}
// Calculates the standard deviation
function getStandardDeviation(dataset) {
      return Math.sqrt(getVariance(dataset))
}
```

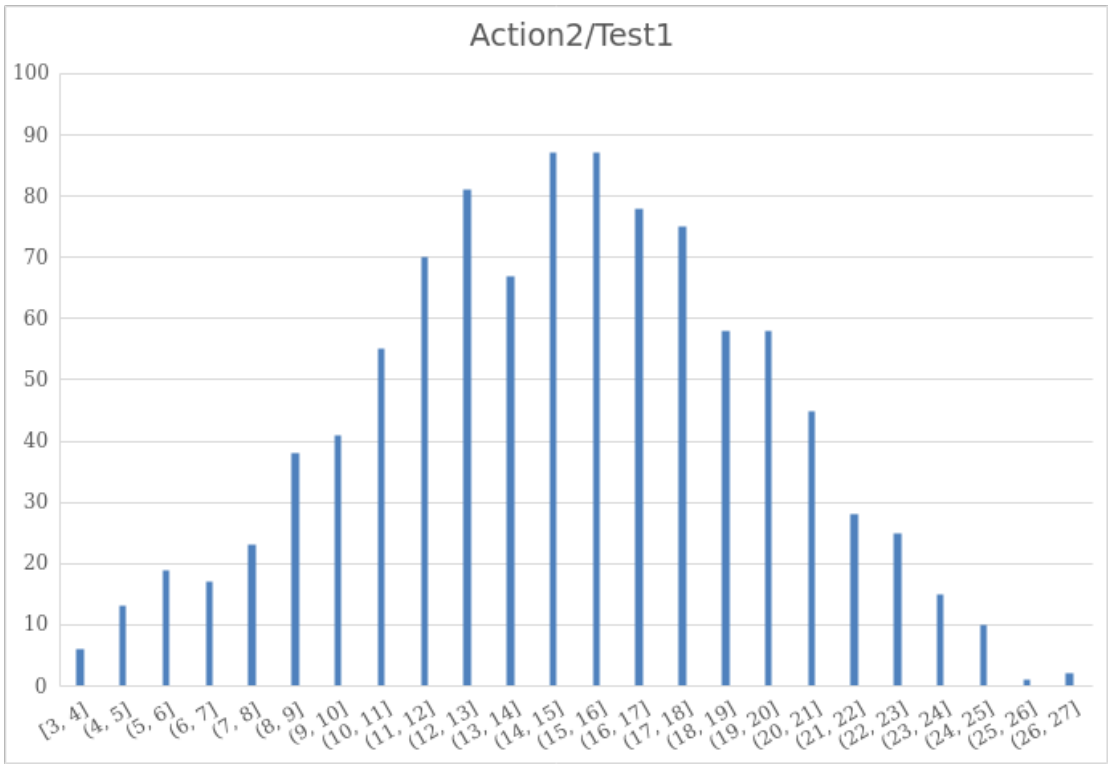
```
// Function for single test
function test(number) {
      let n1 = randomIntegerArray(1, 9, 1000)
      let n2 = randomIntegerArray(1, 9, 1000)
      let n3 = randomIntegerArray(1, 9, 1000)
      let result = [...Array(999)].map((v, i) => {
      return {
      n1: n1[i],
      n2: n2[i],
      n3: n3[i],
      sum: n1[i] + n2[i] + n3[i],
      }
})

let sum = result.map((i) => i.sum)
let meta = {
mean: getMean(sum),
variance: getVariance(sum),
standard_deviation: getStandardDeviation(sum),
}
      console.log(meta)
saveRecords(`./data/action2/test${number}`, result, meta)
}
// Simulates 5 test cases
function simulate() {
for (let i = 1; i <= 5; i++) {
test(i)
}
}
simulate();
```

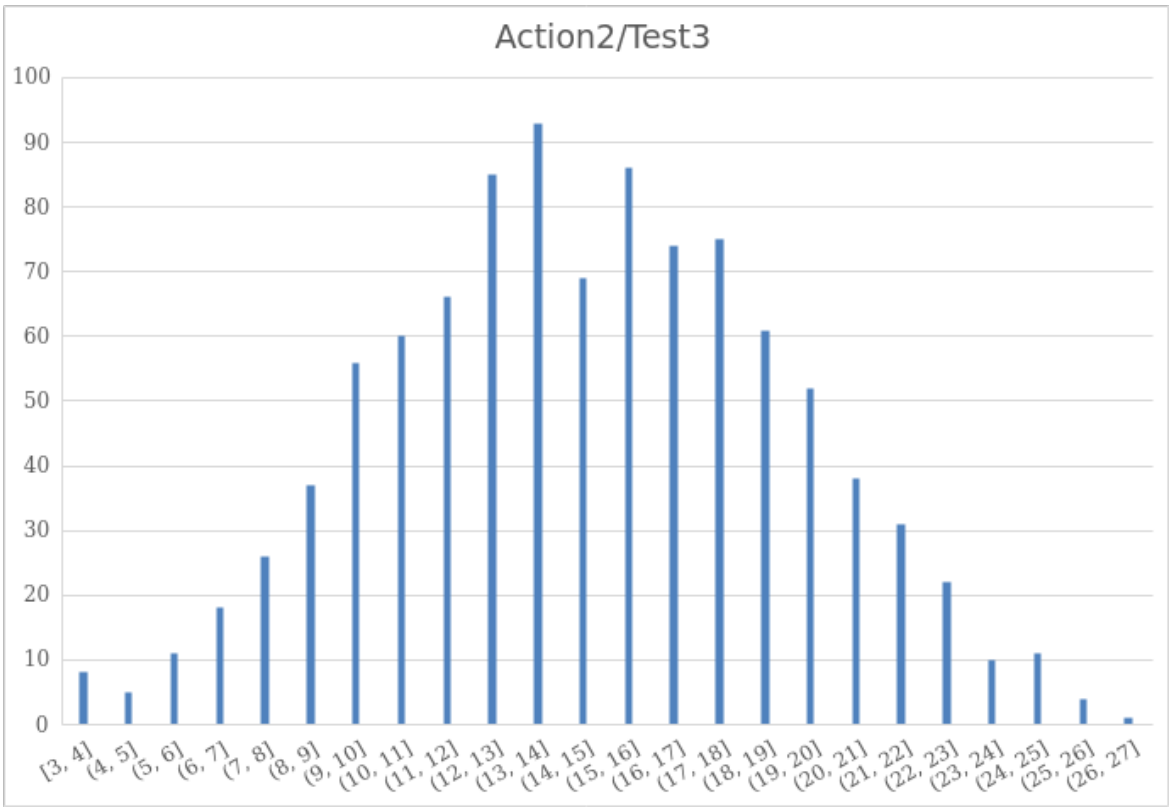
- **Table for Results**

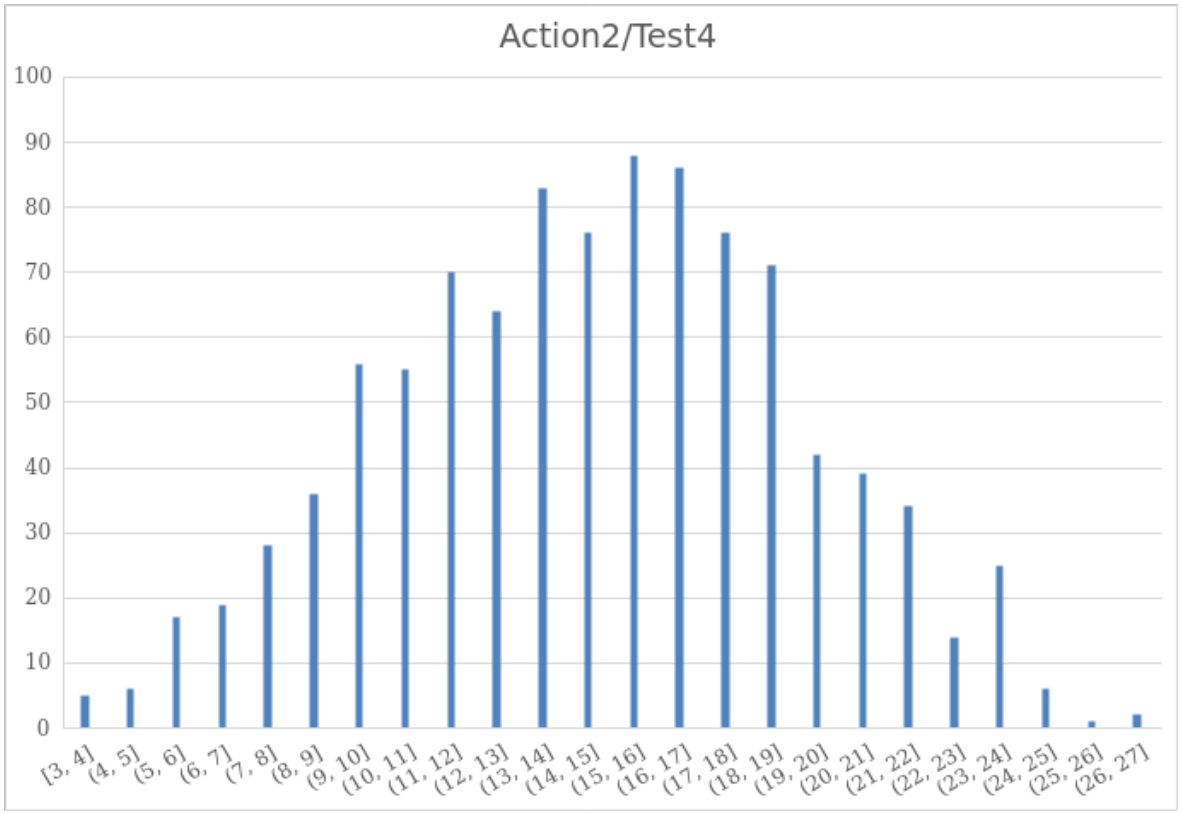
Test/Prop	Mean	Standard Deviation	Variance
1	15.033	4.499	20.246
2	15.169	4.558	20.777
3	15.070	4.449	19.796
4	15.131	4.481	20.085
5	14.858	4.380	19.185

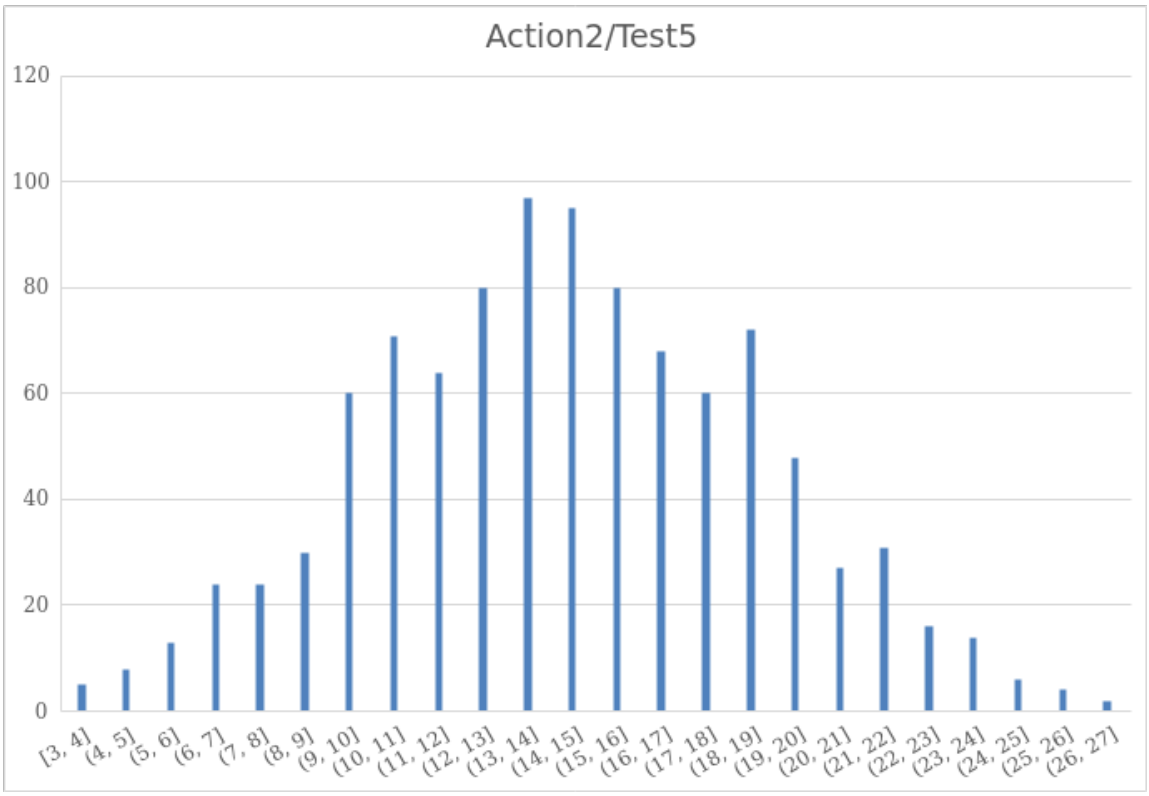
- **Histograms for Test Cases**






Action2/Test3
100
90
80
70
60
50
40
30
20
10
0
[3, 4]
(4, 5]
(5, 6]
(6, 7]
(7, 8]
(8, 9]
(9, 10]
(10, 11]
(11, 12]
(12, 13]
(13, 14]
(14, 15]
(15, 16]
(16, 17]
(17, 18]
(18, 19]
(19, 20]
(20, 21]
(21, 22]
(22, 23]
(23, 24]
(24, 25]
(25, 26]
(26, 27]


Action2/Test4
100
90
80
70
60
50
40
30
20
10
0
[3, 4]
(4, 5]
(5, 6]
(6, 7]
(7, 8]
(8, 9]
(9, 10]
(10, 11]
(11, 12]
(12, 13]
(13, 14]
(14, 15]
(15, 16]
(16, 17]
(17, 18]
(18, 19]
(19, 20]
(20, 21]
(21, 22]
(22, 23]
(23, 24]
(24, 25]
(25, 26]
(26, 27]


Action2/Test5
120
100
80
60
40
20
0
[3, 4]
(4, 5]
(5, 6]
(6, 7]
(7, 8]
(8, 9]
(9, 10]
(10, 11]
(11, 12]
(12, 13]
(13, 14]
(14, 15]
(15, 16]
(16, 17]
(17, 18]
(18, 19]
(19, 20]
(20, 21]
(21, 22]
(22, 23]
(23, 24]
(24, 25]
(25, 26]
(26, 27]

Action 3

- **Simulation Strategy:**

  The simulation strategy as follows:
  - We have p values : 0.1 0.2 0.3 0.4 0.5
  - For each p, we have n : 100 1000 10000
  - So, simulation consists of 15 parts

- **Javascript code:**

```
const fs = require('fs')
const XLSX = require('xlsx')
// Generates excel file from the dataset
function saveRecords(dest, cols, meta) {
      if (!fs.existsSync(dest)) {
      fs.mkdirSync(dest, { recursive: true })
      }
      // Write meta data
      fs.writeFileSync(`${dest}/meta.json`, JSON.stringify(meta))
      // Write .xlsx file
      let sheet = XLSX.utils.json_to_sheet(cols)
      let wb = XLSX.utils.book_new()
      XLSX.utils.book_append_sheet(wb, sheet, 'Default')
      XLSX.writeFile(wb, `${dest}/data.xlsx`)
}
// Selects a child with P(boy) = p
function pick(p) {
      let r = Math.random()
      if (r <= p) {
      return 'boy'
      }
      return 'girl'
}
// Selects N children
function test(p, n) {
      const children = []
      for (let i = 0; i < n; i++) {children.push(pick(p))}
      return {
            children,
            x: children.filter((c) => c == 'boy').length,
      }
}
```

```
// Simulates system with some ps and ns
function simulate() {
      let ps = [0.1, 0.2, 0.3 0.4, 0.5]
      let ns = [100, 1000, 10000]
      ps.forEach(function (p, i) {
            ns.forEach(function (n, i2) {
                  const { children, x } = test(p, n)
                  const meta = {p,n,x,}
                  const rows = children.map((c) => {
                        return {
                              child: c,
                        }
                  })
                  saveRecords(`./data/action3/test${i + 1}/${i2 + 1}`,
rows, meta)
            })

      })
}

simulate()
```

- **Table for Results**

p/n	100	1000	10000
0.1	11	110	990
0.2	17	201	2016
0.3	36	339	2956
0.4	47	394	3925
0.5	56	458	5010

- **Evaluation**

  When the sample size increases, the x/n gets closer to the p

Action 4

- **Javascript code:**

```
const fs = require('fs')
// Generates json file with result data
function saveRecords(dest, meta) {
      if (!fs.existsSync(dest)) {
      fs.mkdirSync(dest, { recursive: true })
      }
      // Write meta data
      fs.writeFileSync(`${dest}/meta.json`, JSON.stringify(meta))
      }
// Calculates the mean
function getMean(dataset) {
      return dataset.reduce((a, b) => a + b) / dataset.length
}
// Calculates the variance
function getVariance(dataset) {
      let sum = 0
      let mean = getMean(dataset)
      for (i of dataset) {
      let val = i - mean
      sum = sum + val * val
      }
      return sum / (dataset.length - 1)
}
// Enter student : p => P(female)
function enter(p) {
      let r = Math.random()
      if (r <= p) {
      return 'female'
      }
      return 'male'
}
// Tests for a single p
function test(p) {
      let counter = 0
      while (true) {
            let student = enter(p)
            if (student == 'female') {
                  return counter
            }
      counter++
      }
}
```

```
// Simulates system with some ps and ns
function simulate() {
      let ps = [0.5, 0.4, 0.3, 0.2, 0.1]
      let xs = []

      ps.forEach(function (p, i) {
            let x = test(p)
            xs.push(x)
            saveRecords(`./data/action4/test${i + 1}`, { p, x })
      })

      let variance = getVariance(xs)
      saveRecords(`./data/action4`, { variance })
}

simulate()
```

- **Table for Results**

p	x
0.1	12
0.2	8
0.3	3
0.4	2
0.5	0

x values' variance = 24

- **Evaluation**

  As p increases x is decreasing.

Action 5

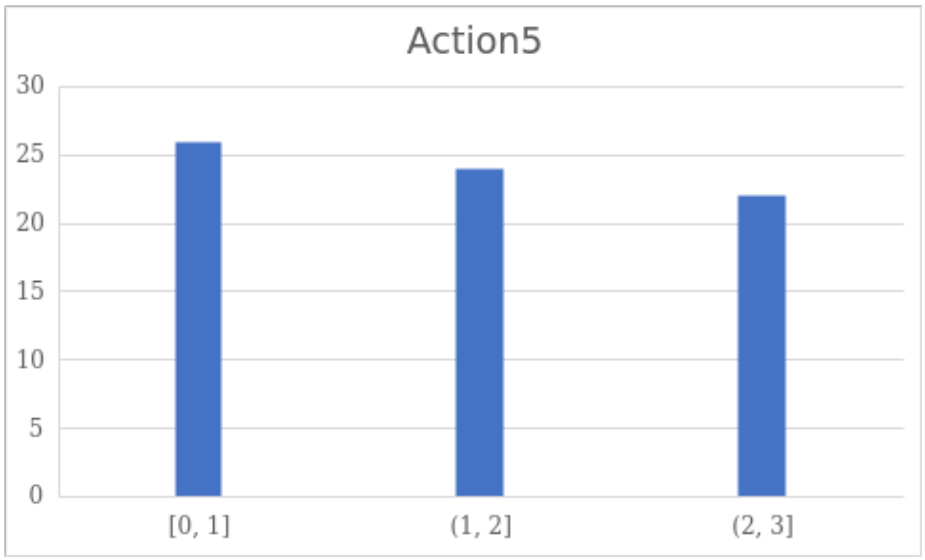
- **Simulation Strategy:**
  - M = 3
  - N = 23
  - For making photon number random, a fluctuation function is used
  - We have T=7 for fluctuation range
  - Our photon number comes between 16-30 (average 23)

- **Javascript code:**

```
const M = 3 // 110510283
const N = 23 // September 23
const T = 7 // fluctuation range

const fs = require('fs')

// Generates json file with result data
function saveRecords(dest, meta) {
      if (!fs.existsSync(dest)) {
            fs.mkdirSync(dest, { recursive: true })
      }
      // Write meta data
      fs.writeFileSync(`${dest}/meta.json`, JSON.stringify(meta))
}

// Calculates the mean
function getMean(dataset) {
      return dataset.reduce((a, b) => a + b) / dataset.length
}

// Calculates the variance
function getVariance(dataset) {
      let sum = 0
      let mean = getMean(dataset)
      for (i of dataset) {
            let val = i - mean
            sum = sum + val * val
      }
      return sum / (dataset.length - 1)
}
```

```
function photons() {
      let r1 = Math.random()
      let r2 = Math.random()
      let nfluc = -1 * r1 * T
      let pfluc = r2 * T
      return Math.floor(N + nfluc + pfluc)
}

function test() {
      let data = []
      for (let i = 0; i < M; i++) {
      let count = photons()
      data.push(count)
      }
      return {
            data: data,
            sum: data.reduce((a, b) => a + b),
            variance: getVariance(data),
      }
}

function simulate() {
      let result = test()
      saveRecords(`./data/action5/test1`, result)
}

simulate()
```

- **Table for Results**




```
sum of photons = 71
x values’ variance = 6.33
```