Mohammed Al-rowad
201420697

Simulation and modeling project.

15th Jan 2019

OVERVIEW

- This document was sent with a link to the code (in github).

A shell application that generates 10 random numbers based on the user's input (seed), validate them, and improve them.

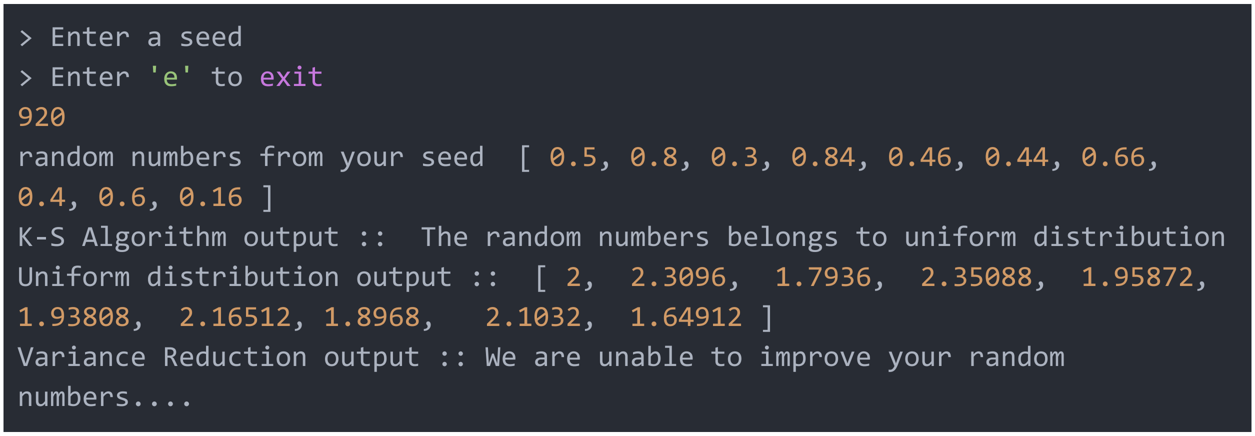
It was built using JavaScript in NodeJs environment.

- this document starts with showing some test cases then showing the code.

Test cases :

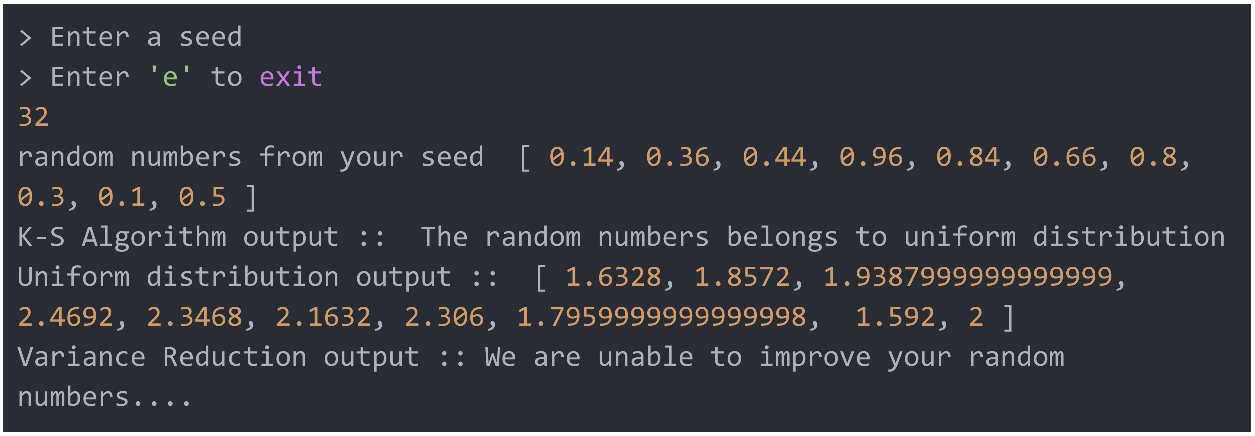
Four test cases and each one of them include all four parts.

Test number one:



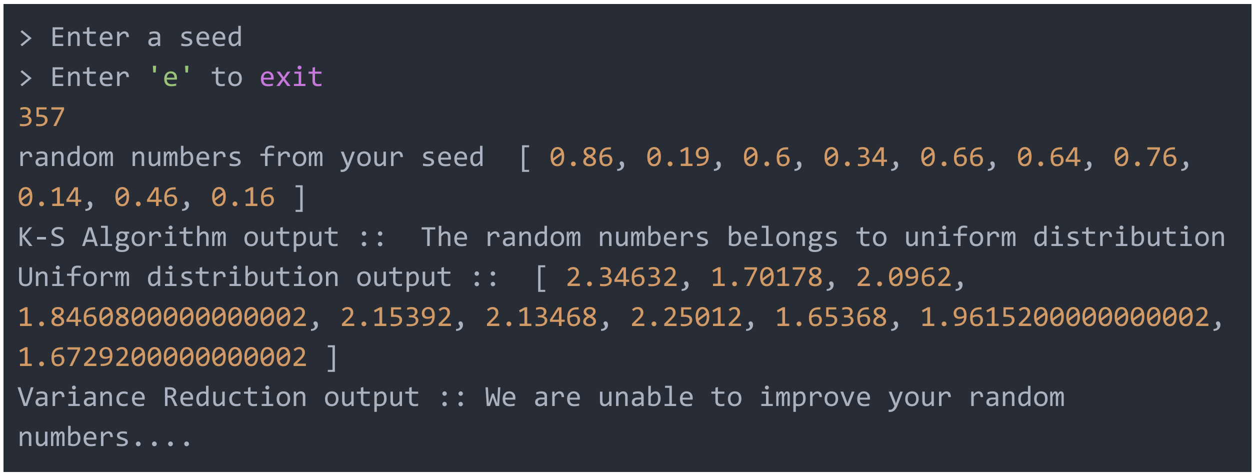
```
> Enter a seed
> Enter 'e' to exit
920
random numbers from your seed  [ 0.5, 0.8, 0.3, 0.84, 0.46, 0.44, 0.66,
0.4, 0.6, 0.16 ]
K-S Algorithm output ::  The random numbers belongs to uniform distribution
Uniform distribution output ::  [ 2,  2.3096,  1.7936,  2.35088,  1.95872,
1.93808,  2.16512, 1.8968,   2.1032,  1.64912 ]
Variance Reduction output :: We are unable to improve your random
numbers....
```

Test number two:



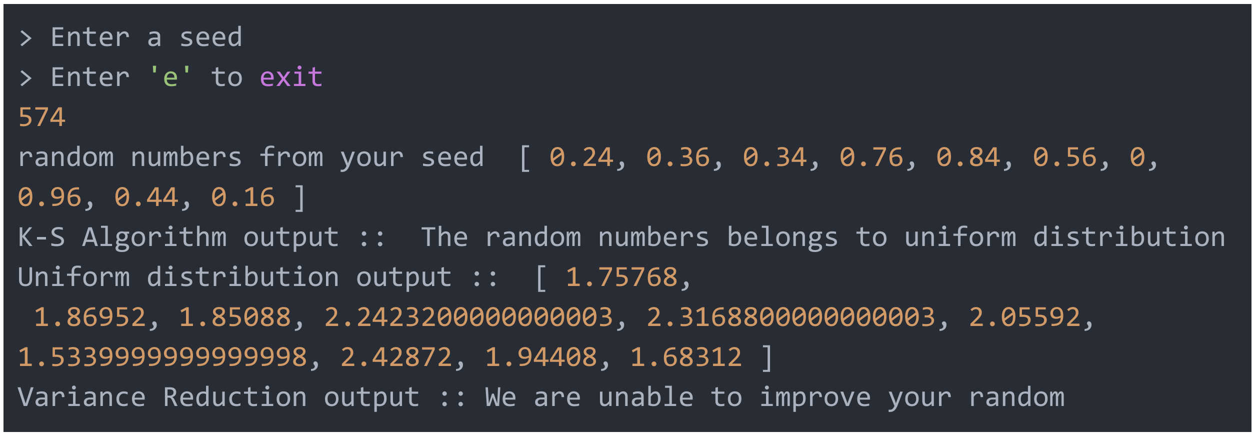
```
> Enter a seed
> Enter 'e' to exit
32
random numbers from your seed  [ 0.14, 0.36, 0.44, 0.96, 0.84, 0.66, 0.8,
0.3, 0.1, 0.5 ]
K-S Algorithm output ::  The random numbers belongs to uniform distribution
Uniform distribution output ::  [ 1.6328, 1.8572, 1.9387999999999999,
2.4692, 2.3468, 2.1632, 2.306, 1.7959999999999998,  1.592, 2 ]
Variance Reduction output :: We are unable to improve your random
numbers....
```

Test number three:



```
> Enter a seed
> Enter 'e' to exit
357
random numbers from your seed  [ 0.86, 0.19, 0.6, 0.34, 0.66, 0.64, 0.76,
0.14, 0.46, 0.16 ]
K-S Algorithm output ::  The random numbers belongs to uniform distribution
Uniform distribution output ::  [ 2.34632, 1.70178, 2.0962,
1.8460800000000002, 2.15392, 2.13468, 2.25012, 1.65368, 1.9615200000000002,
1.6729200000000002 ]
Variance Reduction output :: We are unable to improve your random
numbers....
```

Test number four:



```
> Enter a seed
> Enter 'e' to exit
574
random numbers from your seed  [ 0.24, 0.36, 0.34, 0.76, 0.84, 0.56, 0,
0.96, 0.44, 0.16 ]
K-S Algorithm output ::  The random numbers belongs to uniform distribution
Uniform distribution output ::  [ 1.75768,
 1.86952, 1.85088, 2.2423200000000003, 2.3168800000000003, 2.05592,
1.5339999999999998, 2.42872, 1.94408, 1.68312 ]
Variance Reduction output :: We are unable to improve your random
```

```
numbers....
```

Assignment one

Algorithm

The numbers was generated based on the following *pseudo code* :

```
• Get the user's seed, then each iteration do :
1. Generate a number called val
    a. Shift the binary representation of the seed by its length ( k ), to the left.
    b. Then square it.
2. Get the val [iteration number] or first number ( if there is no value in it ) and the first number of val, store them as result.
3. Store the result in a Set called ( random ).
4. Map the random values
    a. by dividing each random number in the random set over 10 power the length of current number.
5. Change the value of k to be the set length plus the current iteration number.
```

Code :

The below code is the code that generate the random number, the parts of the code that formats the output and add colors are excluded.

```
console.log('> Enter a seed', '\n> Enter \'e\' to exit')
const stdin = process.openStdin()
stdin.addListener('data', d => {
    let random = new Set()
    let resArr = []
    let tenNumbers = 10
    let seed = d.toString().trim()
    if (seed === 'e') process.exit()
    let k = seed.length
    for (let i = 0; i < tenNumbers; i++) {
        const val = Math.pow(seed << k, 2).toString()
        const result = +`${val[i] || val[0]}${val[val.length - 1]}`
        if (random.has(result)){
           seed++
           tenNumbers++
        }
        random.add(result)
        k = random.values.length + i
    }
    random.forEach(r => resArr.push(r / Math.pow(10, r.toString().length)))
    console.log(resArr)
})
```

Assignment two

Simple implementation of **K-S algorithm**

Code :

```
// assugment #2 ~ K-S Algorithm
const uniDist = require('../assignment #3/main').uniformDis;
// step ~1
exports.KSAlgorithm = randomNumbers => {
 const freq = new Map();
 // step ~2
 randomNumbers.forEach(rn =>
   freq.has(rn) ? freq.set(rn, freq.get(rn) + 1) : freq.set(rn, 1)
 );
 let acc = [];
 let len = -1;
 // step ~3
 freq.forEach((value, key) => acc.push(value + (acc.length > 0 ? acc[++len] : 0)))
 // step ~4
 let facc = [];
 acc.forEach(num => facc.push(num / acc[acc.length - 1]))
 // step ~5
 let findex = [];
 freq.forEach((value, key) =>
   findex.push((key / freq.size)));
 // step ~6
 let errors = [];
 for (let i = 0; i < findex.length; i++)
   errors.push(Math.abs(findex[i] - facc[i]));
 // step ~7
 const ktheo = errors.reduce((prv, curr) => prv > curr ? prv : curr);
 // step ~8
 const kexp = 16.92;
 if (ktheo < kexp) console.log('K-S Algorithm output :: ', 'The random numbers belongs to uniform distribution')
 else console.log('K-S Algorithm output :: ', 'The random numbers DO NOT belong to uniform distribution')
 uniDist(randomNumbers);
}
```

Assignment three

Simple implementation the **Uniform distribution rule.**

Code :

```
const VRed = require('../assignment #4/main').VRed;
exports.uniformDis = randomNumbers => {
 const mean = randomNumbers.reduce((prv, curr) => prv + curr) / randomNumbers.length;
 const a = Math.abs(mean - 2);
 const b = mean + 2
 let newrv = [];
 randomNumbers.forEach((rn, i) =>
   newrv.push(a + (b - a) * rn));

 console.log('Uniform distribution output :: ', newrv)
 VRed(newrv);
}
```

Assignment Four

Simple implementation the **Variance reduction algorithm.**

Code :

```
exports.VRed = randomNumbers => {
 let newrv = [];
 // inter arr = 1
 randomNumbers.forEach(rn =>
   newrv.push(-Math.log(rn))
 );
 let mean = newrv.reduce((prv, curr) => prv + curr) / newrv.length;
 const inter = setInterval(() => {
   if ((Math.abs(mean - 1) <= 10e-5)) {
     console.log('Variance Reduction output ::  improved random numbers ',
newrv);
     process.exit(1);
   }
   newrv.forEach(rn =>
     newrv.push(Math.abs(1 - rn))
   );
   newrv.slice(10);
   mean = newrv.reduce((prv, curr) => prv + curr) / newrv.length;
 }, 0);

 setTimeout(() => {
   clearInterval(inter);
   console.log('Variance Reduction output :: We are unable to improve your
random numbers....')
 }, 1000);
}
```