P8

edgardjfc

April 2022

1 Introduction

For this assignment we observed the behaviour of an urn model where different groups of particles are formed, separated and regrouped back together in a repetitive cycle. With an initial amount of particles (n) forming an initial amount of groups (k), with the objective of finding the influence that both n and k influence the percentage of filtered particles in a system that purges groups of a certain size.

2 Code

The following code was designed from the base code given by our teacher E. Schaeffer [2] and leaning into the code written and developed by our classmate J. "FeroxDeitas" [1]

```
clusters = [500, 2500, 10000]
particles = [50000, 250000, 1000000]
filtersize = 100
runs = 100
dur = 50
```

Here we set the initial values for our number of clusters and the amount of particles that will be part of each permutation of the experiment, coming to a total of 9 different variations. Doing 100 runs per experiment with a duration of 50 each.

```
def rotura(x, c, d):
    return 1 / (1 + exp((c - x) / d))

def union(x, c):
    return exp(-x / c)

def romperse(tam, cuantos):
    if tam == 1:
        return [tam] * cuantos
```

```
    res = []
    for cumulo in range(cuantos):
        if random() < rotura(tam, c, d):
            primera = randint(1, tam - 1)
            segunda = tam - primera
            assert primera > 0
            assert segunda > 0
            assert primera + segunda == tam
            res += [primera, segunda]
        else:
            res.append(tam)
    assert sum(res) == tam * cuantos
    return res

def unirse(tam, cuantos):
    res = []
    for cumulo in range(cuantos):
        if random() < union(tam, c):
            res.append(-tam)
        else:
            res.append(tam)
    return res
```

Here we define the functions for combining and breaking apart the clusters for the experiment.

```
for k in clusters:
    resultados = pd.DataFrame()
    for n in particles:
        filtrados = []
        for r in range(runs):
            filtro = 0
            orig = np.random.normal(size = k)
            cumulos = orig - min(orig)
            cumulos += 1
            cumulos = cumulos / sum(cumulos)
            cumulos *= n
            cumulos = np.round(cumulos).astype(int)
            diferencia = n - sum(cumulos)
            cambio = 1 if diferencia > 0 else -1
            while diferencia != 0:
                p = randint(0, k - 1)
                if cambio > 0 or (cambio < 0 and cumulos[p] > 0):
                    cumulos[p] += cambio
                    diferencia -= cambio
            assert all(cumulos != 0)
            assert sum(cumulos) == n
```

In this part of the code the previous functions are called in order to produce the initial iterations for each run of the experiments.

```
        for paso in range(dur):
            assert sum(cumulos) == n
            assert all([c > 0 for c in cumulos])
            (tams, freqs) = np.unique(cumulos, return_counts = True)
            cumulos = []
            assert len(tams) == len(freqs)
            for i in range(len(tams)):
                cumulos += romperse(tams[i], freqs[i])
            assert sum(cumulos) == n
            assert all([c > 0 for c in cumulos])
            (tams, freqs) = np.unique(cumulos, return_counts = True)
            cumulos = []
            assert len(tams) == len(freqs)
            for i in range(len(tams)):
                cumulos += unirse(tams[i], freqs[i])
            cumulos = np.asarray(cumulos)
            neg = cumulos < 0
            a = len(cumulos)
            juntarse = -1 * np.extract(neg, cumulos)
            cumulos = np.extract(~neg, cumulos).tolist()
            assert a == len(juntarse) + len(cumulos)
            nt = len(juntarse)
            if nt > 1:
                shuffle(juntarse)
            j = juntarse.tolist()
            while len(j) > 1:
                cumulos.append(j.pop(0) + j.pop(0))
            if len(j) > 0:
                cumulos.append(j.pop(0))
            assert len(j) == 0
            assert sum(cumulos) == n
            assert all([c != 0 for c in cumulos])
```

Here, the different initial iterations are run for 50 different steps.

```
        for p in cumulos:
            if p >= filtersize:
                filtro += 1
        porcentaje = (filtro / sum(cumulos)) * 100
        filtrados.append(porcentaje)

    resultados['n: ' + str(n) + ', k: ' + str(k)] = pd.DataFrame(filtrados)
```

This last segment calculates the remaining particles and the filtered particles, obtaining the percentage for each of the different runs of each experiment.

3 Results

For figures 1-3 we have violin graphs where we see the percentage of the filtered clusters at the end of each run of experiments, observing the difference that each initial value of particles causes.

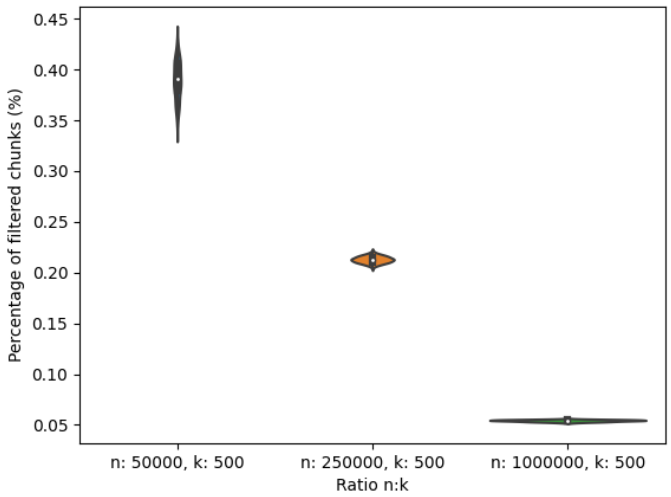

Figure 1: 500 initial clusters

4 Analysis

We performed a statistical analysis of variance which yielded positive results, indicating that each of the different permutations had significant statistical influence on each of the experiments.

5 Conclusion

For this assignment we observed that the different initial conditions had a significant statistical impact, and that it made a big difference how the initial values for k and n influenced the behavior of each experiment.

References

[1] J.FeroxDeitas. *GitHub,P8*. URL: `https://github.com/FeroxDeitas/Simulacion-Nano/tree/main/Tareas/P8`.

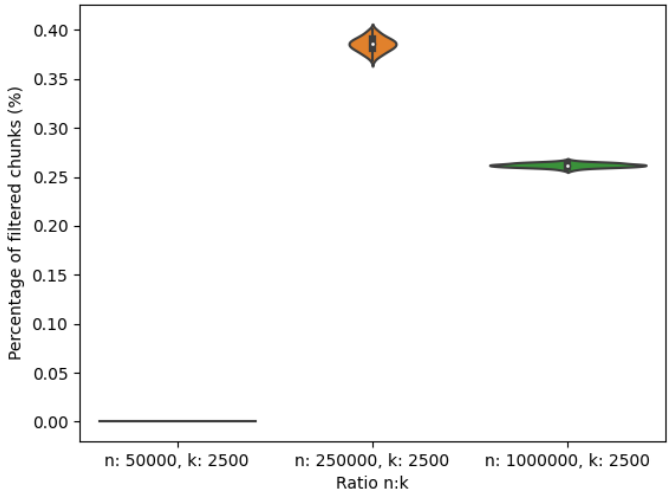

Figure 2: 2500 initial clusters

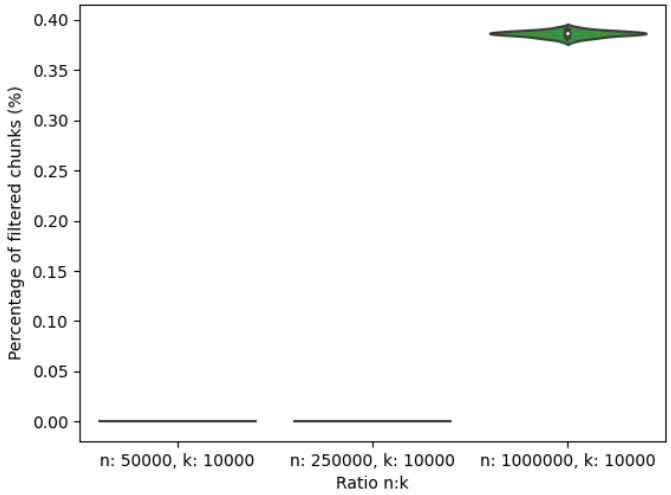

Figure 3: 10000 initial clusters

[2] E. Schaeffer. *GitHub, UrnModel*. URL: https://github.com/satuelisa/Simulation/tree/master/UrnModel.