Reporte 8:
Modelo de Urnas

Jorge Torres

4 de abril de 2022

1. Objetivo

En esta práctica se realiza la simulación de un sistema de coalescencia y fragmentación de partículas, en donde una cantidad n de partículas forma inicialmente una cantidad k de cúmulos de diversos tamaños. En cada paso de la iteración, un cúmulo puede unirse a otro o puede fragmentarse en dos cúmulos más pequeños, no necesariamente del mismo tamaño. El objetivo consiste en estudiar estadísticamente el efecto que tiene la tasa n/k en el porcentaje de partículas que se lograrían filtrar si se tuviese un filtro de un tamaño determinado. En esta ocasión, se utilizan tres cantidades de partículas en combinación con tres cantidades de cúmulos iniciales, dando un total de 9 combinaciones para la tasa n/k.

2. Desarrollo

El desarrollo de la presente práctica está basado en el código implementado por E. Schaeffer [1], en donde realiza una simulación de unión y fragmentación de partículas. El desarrollo completo puede encontrarse en el repositorio en GitHub de J. Torres [2].

En primera instancia, se defninen los parámetros iniciales con los que opera la simulación. Éstos consisten de una lista con las tres cantidades de cúmulos iniciales, `clusters`; una lista con las tres cantidades de partículas, `particles`; el tamaño del filtro, en términos de la cantidad mínima de partículas que debe tener un cúmulo para que se quede alojado en el mismo, `filtersize`; la cantidad de iteraciones que se realizan para cada combinación de partículas y cúmulos, `runs`; y la cantidad de pasos que dura cada iteración para cada combinación, `dur`. Los parámetros se pueden ver implementados en el código 1.

Código 1: Parámetros de Operación

```
clusters = [2500, 5000, 10000]
particles = [250000, 500000, 1000000]
filtersize = 100
runs = 100
dur = 50
```

En el código 2 se definen las probabilidades de que los cúmulos se rompan o se unan con las funciones `rotura()` y `union()`, respectivamente. La probabilidad de rotura depende de manera sigmoidal de un tamaño crítico de cúmulo, c, y de un factor arbitrario para suavizar la curva, d. El tamaño crítico está dado por la media de tamaños de los cúmulos en cada iteración, mientras que el factor de suavización se determina con la desviación estándar de los tamaños de cúmulos. Por otro lado, la probabilidad de unión depende únicamente del tamaño crítico de manera exponencial negativa. De esta forma, los cúmlos más pequeños tienden a unirse, mientras que los más grandes tienden a fracturarse.

Código 2: Probabilidades de Rotura y Unión

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

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

c = np.median(cumulos)
d = np.std(cumulos) / 4
```

En seguida, en el código 3 se definen las funciones `romperse()` y `unirse()`. La función `romperse()` decide qué cúmulos se romperán en el siguiente paso y qué tamaños tendrán los dos cúmulos resultantes de la rotura, mientras que la función `unirse()` crea dos grupos de cúmulos para posteriormente decidir cuáles se unirán con cuáles en el próximo paso de la iteración. Estas funciones dependen respectivamente de las probabilidades de rotura y unión descritas en el código 2.

Código 3: Acciones de Rotura y Unión de Cúmulos

```
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
```

El siguiente paso, observado en el código 4, es comenzar la iteración entre las cantidades iniciales de cúmulos y las cantidades de partículas. También se comienzan las 100 repeticiones de cada combinación. Para cada repetición se crea nuevamente una cantidad k de cúmulos, de tal manera que la distribución de tamaños siguen una distribución normal, no existen cúmulos vacíos, en total suman a la cantidad n de partículas y todos los tamaños son números enteros.

Código 4: Inicio de Iteraciones y Creación de Cúmulos

```
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
```

En el código 5, se inician los 50 pasos de la iteración, en donde los cúmulos son agrupados por tamaños y la cantidad de cúmulos que existen con esos tamaños. Es aquí donde se comienzan a fracturar y unir los cúmulos con las funciones definidas en el código 3. Para cada paso de la iteración, primero se decide qué cúmulos van a romperse y se rompen; después, se decide qué cumulos van a unirse y se unen de manera aleatoria unos con otros. Estos pasos crean una nueva distribución de cantidades y tamaños de cúmulos para cada paso de la iteración.

Código 5: Fractura y Unión de Cúmulos en la Iteración

```
            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])
```

Por último, con el código 6, al final de cada iteración se decide la cantidad de cúmulos que son lo suficientemente grandes para quedar en el filtro y se calcula un porcentaje de partículas filtradas. Estos porcentajes se guardan en archivos que se utilizan para su posterior análisis.

Código 6: Cálculo de Porcentajes de Partículas Filtradas

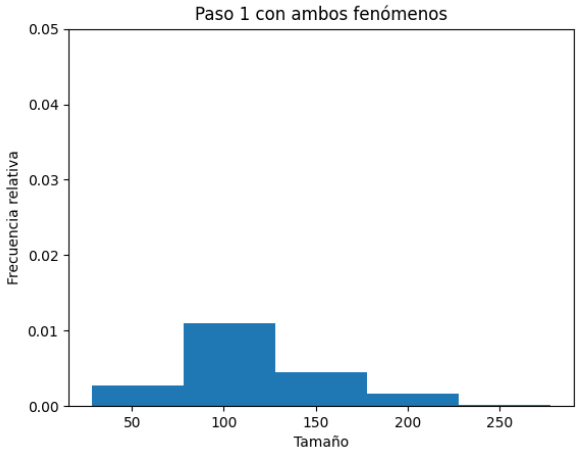
```
            for p in cumulos:
                if p >= filtersize:
                    filtro += 1
            porcentaje = (filtro / sum(cumulos)) * 100
            filtrados.append(porcentaje)

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

    resultados.to_csv('urnas_{:d}.csv'.format(k))
```

3. Resultados

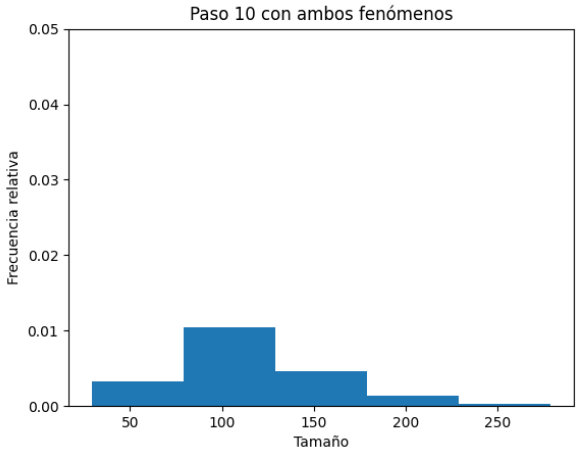
A manera de resultados, primero se muestra una visualización de viñetas donde se observa el comportamiento a través del tiempo de una iteración de fracturas y uniones de cúmulos por medio de histogramas. En segunda instancia, se muestran diagramas tipo violín donde se relacionan la tasa de cantidad de partículas a cantidad de cúmulos iniciales, con el porcentaje de partículas filtradas. Además, se explica el análisis estadístico realizado para determinar si existe una diferencia estadísticamente significativa al variar la cantidad de cúmulos iniciales.

3.1. Comportamiento del Sistema a Través del Tiempo

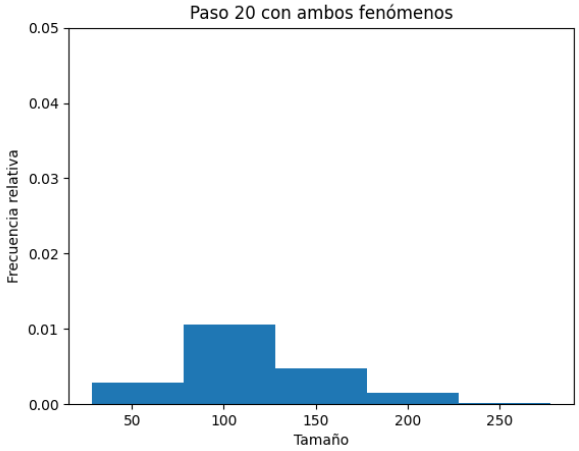
La figura 1 muestra el comportamiento que tiene una iteración de la simulación a través del tiempo. Los parámetros de dicha iteración son una cantidad de partículas, $n = 250000$ y una cantidad inicial de cúmulos, $k = 2500$. Una visualización animada de la misma iteración se puede encontrar en el archivo GIF del repositorio de J. Torres. Como se puede observar, el sistema tiende tener más cúmulos de tamaños cercanos al tamaño crítico, lo cual es de esperarse, ya que estarían oscilando entre unirse y fracturarse.



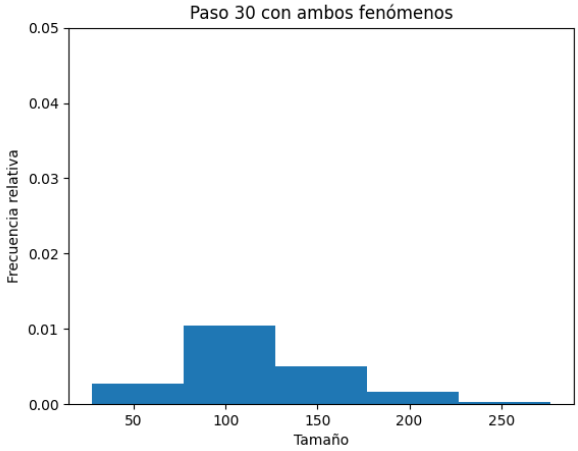
(a)



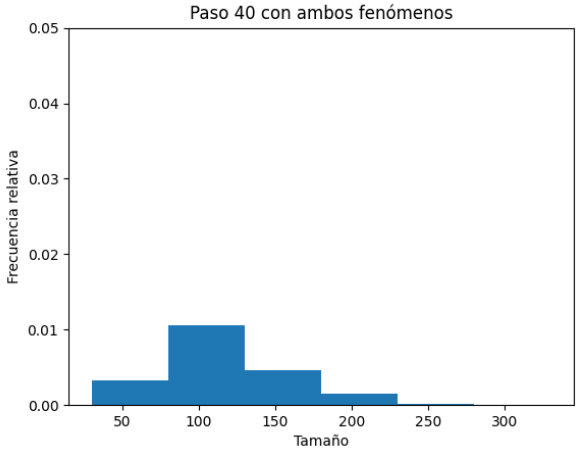
(b)



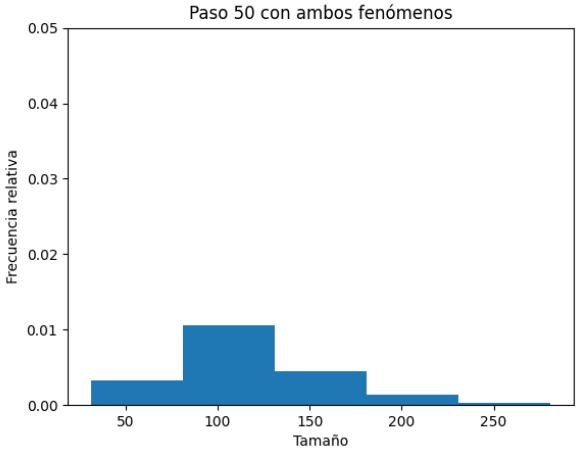
(c)



(d)

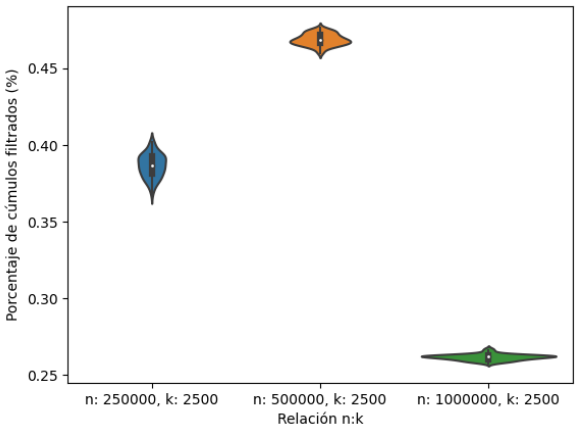


(e)

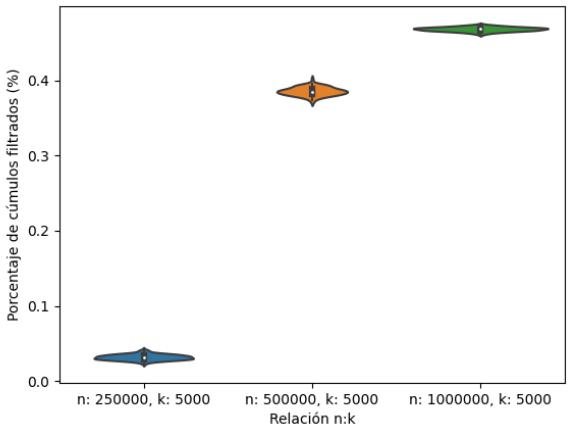


(f)

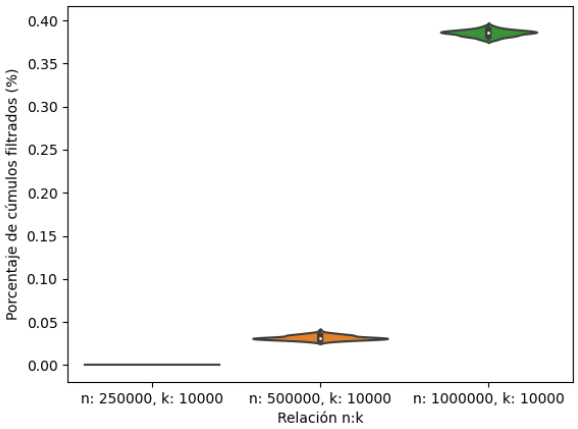
Figura 1: Evolución del sistema a través del tiempo



(a) Cantidad inicial de cúmulos, $k = 2500$



(b) Cantidad inicial de cúmulos, $k = 5000$



(c) Cantidad inicial de cúmulos, $k = 10000$

Figura 2: Distribución de porcentajes de partículas filtradas para cada tasa n/k.

3.2. Análisis Estadístico

La figura 2 muestra los diagramas tipo violín de las nueve combinaciones entre cantidad de partículas y cantidad inicial de cúmulos, donde se observan las distribuciones del porcentaje de partículas filtradas para las 100 iteraciones de cada combinación. En la figura 2a se observan los porcentajes para la variación de cantidad de partículas con una $k = 2500$, para la figura 2b la cantidad inicial de cúmulos es $k = 5000$, y para la figura 2c es de $k = 10000$.

Además de los diagramas tipo violín, se realizó un análisis de varianza para determinar si existe una diferencia significativa al variar la cantidad de partículas con las que trabaja el sistema. Del análisis se encontró que los porcentajes de partículas filtradas no pertenecen a la misma distribución al variar la cantidad de partículas, ya que los valores p son mucho menores al valor de 0,05 que es necesario para deducir lo contrario. Cabe mencionar que no se realizó un análisis para demostrar si los porcentajes pertenecen a la misma distribución al variar la cantidad inicial de cúmulos formados. Sin embargo, al observar los diagramas se puede apreciar una diferencia considerable en las distribuciones, lo cual podría indicar que también existe una relación entre el porcentaje de parttículas filtradas y la cantidad inicial de cúmulos.

4. Conclusiones

En la práctica se realizó la variación de la tasa entre cantidad de partículas y cantidad inicial de cúmulos, n/k, para determinar si existe una relación entre ella y el porcentaje de partículas filtradas. Del análisis estadístico se

observa que hay una diferencia significativa al variar la cantidad de partículas, por lo que es muy probable que exista una relación con la misma. Aunque no se realizó el análisis para determinar si también hay relación con la cantidad inicial de cúmulos, es fácil deducir que sí la hay al observar la gran diferencia de distribuciones de los diagramas tipo violín.

Referencias

[1] E. Schaeffer. Urnmodel, 2019. URL https://github.com/satuelisa/Simulation/tree/master/UrnModel.

[2] J. Torres. P8, 2022. URL https://github.com/FeroxDeitas/Simulacion-Nano/tree/main/Tareas/P8.