"P2" Teoría de Colas

NESTOR

Marzo 2022

1. Objetivo

El objetivo de la práctica consiste en analizar los tiempos de ejecución de ("n") algoritmos de diferentes que se encuentran en un número entero dado que es un número primo. Ya que un número primo es un número natural mayor que ("1") que tiene únicamente dos divisores positivos distintos. Ya que que se trata de un número positivo que es imposible de expresar como producto de otros dos números enteros igualmente positivos pero menores que él o en su defecto como un producto de dos números enteros que poseen varias formas.

2. Desarrollo

De acuerdo con el desarrollo en el código implementado por E. Schaeffer [4], primero se definen los parámetros de ejecución de los algoritmos, donde se crea una lista de números desde 1000 hasta 7919. A continuación se muestra los siguientes comandos:

```
from math import ceil, sqrt
from random import shuffle
import multiprocessing
from time import time
from scipy.stats import f_oneway
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

d = 1000
h = 7919
replicas = 30
original = [x for x in range(d, h + 1)]
invertido = original[::-1]
aleatorio = original.copy()
shuffle(aleatorio)
cores = multiprocessing.cpu_count()

def primo_1(n):
if n < 3:
    return True
for i in range(2, n):
    if n % i == 0:
    return False
    return True
```

```
def primo_2(n):
if n < 4:
    return True
if n % 2 == 0:
    return False
for i in range(3, n - 1, 2):
    if n % i == 0:
    return False
    return True

def primo_3(n):
if n < 4:
    return True
if n % 2 == 0:
    return False
for i in range(3, int(ceil(sqrt(n))), 2):
    if n % i == 0:
    return False
    return True

def primo_4(n):
if n < 4:
    return True
if n % 2 == 0:
    return False
for i in range(4, int(ceil(sqrt(n))), 3):
    if n % i == 0:
    return False
    return True

if __name__ == "__main__":
resultados_1 = {'Prueba 1': [], 'Prueba 2': [], 'Prueba 3': [], 'Prueba 4': []}
resultados_2 = {'Resultado 1': [], 'Resultado 2': [], 'Resultado 3': [], 'Resultado 4': []}
with multiprocessing.Pool(processes = cores-1) as pool:
  pool.map(primo_1, original)
  for r in range(replicas):
    t = (time()*1000)
    pool.map(primo_1, original)
    resultados_1['Prueba 1'].append((time()*1000)-t)
    t = (time()*1000)
    pool.map(primo_2, original)
    resultados_1['Prueba 2'].append((time()*1000)-t)
    t = (time()*1000)
    pool.map(primo_3, original)
    resultados_1['Prueba 3'].append((time()*1000)-t)
    t = (time()*1000)
    pool.map(primo_4, original)
    resultados_1['Prueba 4'].append((time()*1000)-t)
    t = (time()*1000)
    pool.map(primo_3, original)
    resultados_2['Resultado 1'].append((time()*1000) - t)
    t = (time()*1000)
    pool.map(primo_3, invertido)
    resultados_2['Resultado 2'].append((time()*1000) - t)
```

```
    t = (time()*1000)
    pool.map(primo_3, aleatorio)
    resultados_2['Resultado 3'].append((time()*1000) - t)
    t = (time()*1000)
    pool.map(primo_3, aleatorio)
    resultados_2['Resultado 4'].append((time()*1000) - t)
df1 = pd.DataFrame(data = resultados_1)
df2 = pd.DataFrame(data = resultados_2)
print(df1, '\n', df2)
stat1, p1 = f_oneway(resultados_1['Prueba 1'],
    resultados_1['Prueba 2'],
    resultados_1['Prueba 3'],
    resultados_1['Prueba 4'])
print('Variando algoritmo\n', 'stat=%.3f, p=%.3f' % (stat1, p1))
if p1 > 0.05: print('Estad sticamente no significativa\n')
else:
print('Estad sticamente significativa\n')
    stat2, p2 = f_oneway(resultados_2['Resultado 1'],
    resultados_2['Resultado 2'],
    resultados_2['Resultado 3'],
    resultados_2['Resultado 4'])
print('Variando orden de numeros\n', 'stat=%.3f, p=%.3f' % (stat2, p2))
if p2 > 0.05:
 print('Estad sticamente no significativa\n')
else:
 print('Estad sticamente significativa\n')

    sns.violinplot(data = df1, scale='count')
    plt.savefig('Prueba.png')
    plt.show()
    sns.violinplot(data = df2, scale='count')
    plt.savefig('Resultado.png')
    plt.show()
```

3. Resultados

Se muestra en las gráficas para visualizar la distribución de los datos y su densidad de probabilidad. Este gráfico es una combinación de un diagrama de cajas y bigotes y un diagrama de densidad girado y colocado a cada lado, para mostrar la forma de distribución de los datos. La barra negra gruesa en el centro representa el intervalo intercuartil, la barra negra fina que se extiende desde ella, representa el 95 por ciento de los intervalos de confianza, y el punto blanco es la mediana. A continuación se muestra las siguiemtes gráficas:

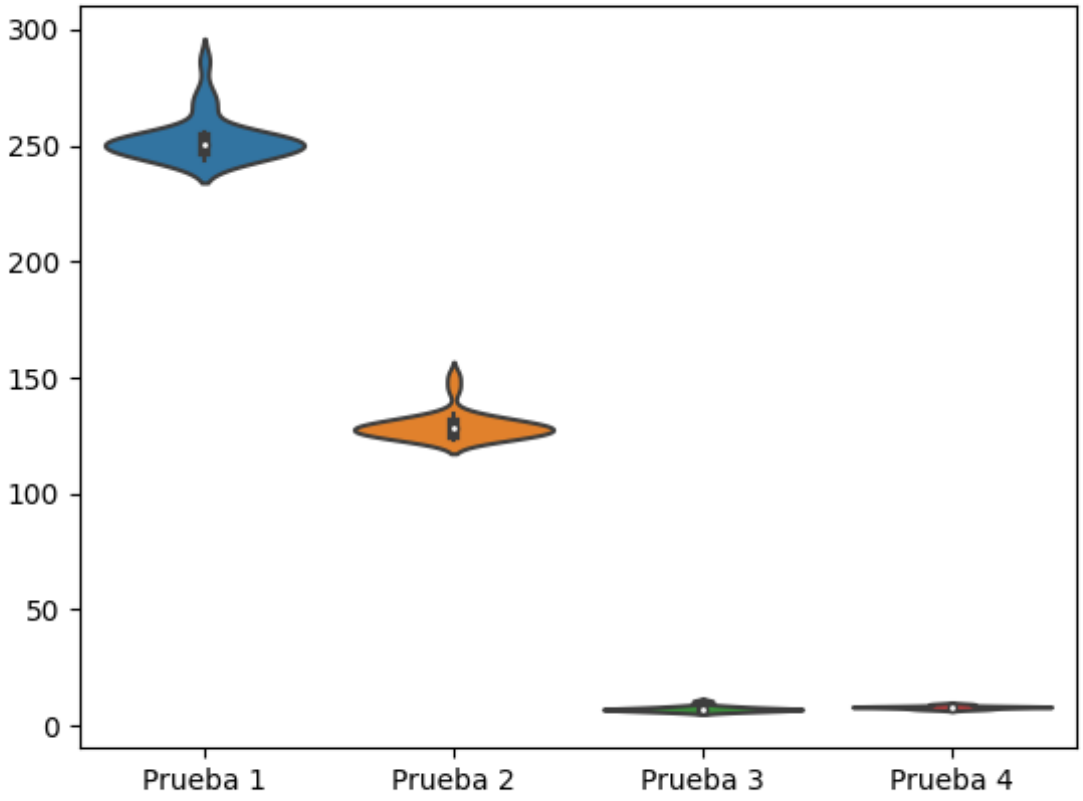

Figura 1: Prueba de Algoritmo

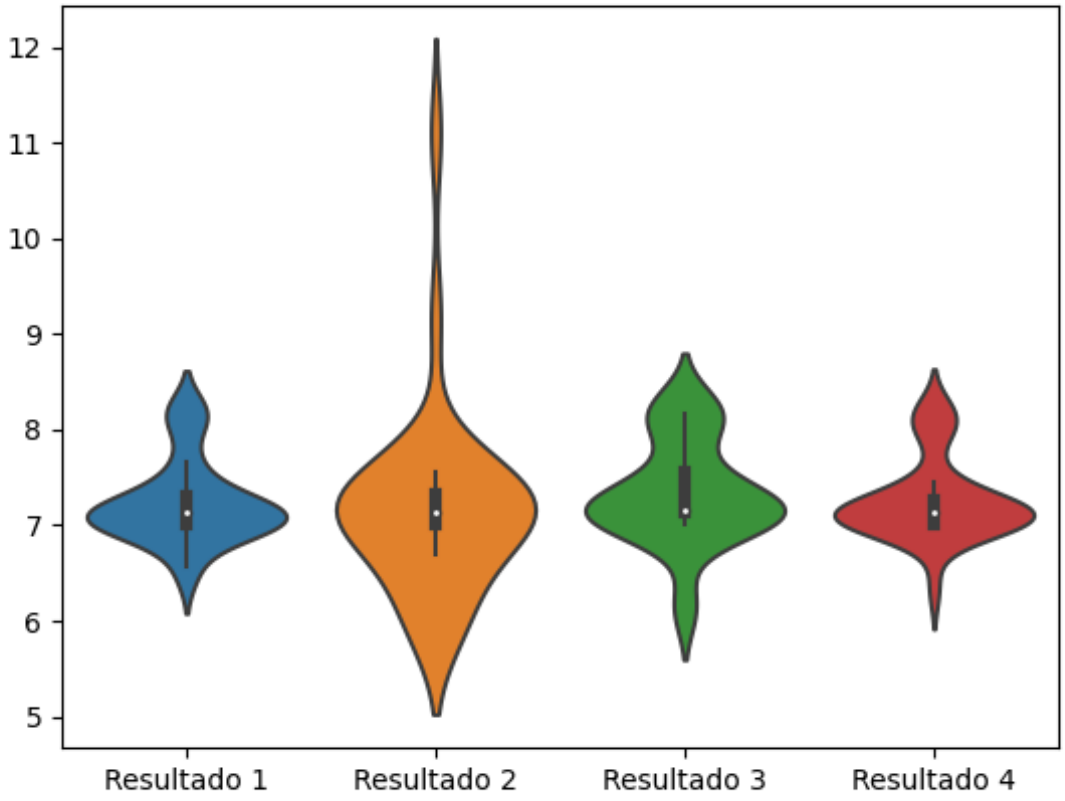

Figura 2: Resultado de Variación

4. Conclusiones

Como puede verse en la tabla, tenemos una densidad más alta entre 100 y 300 para la figura 1 y para la figura 2 es de 5 y 10. Eso es muy significativo porque, como en la descripción nos da un valor medio.

Referencias

[1] J. Hunter. Lines, bars and markers. *Repositorio, GitHub*, 2021.

[2] D. Leyva. Teoría de colas. *Repositorio, GitHub*, 2021.

[3] N. Pérez. Teoría de colas. *Repositorio, GitHub*, 2021.

[4] E. Schaeffer. Queuingtheory. *Repositorio, GitHub*, 2022.

[5] J. Torres. Teoría de colas. *Repositorio, GitHub*, 2022.

[5] [4] [3] [2] [1]