UFRJ - Universidade Federal do Rio de Janeiro
Professor: Daniel Sadoc
Alunos:
Claudio Netto - 113166858
Gabriel Martins Machado Christo - 117217732
Yago Alves da Costa - 115212477
Yuri Medeiros da Silva - 117061898

Primeiro trabalho de AD: Fila M/M/1 e Ruína do apostador

1 Introdução

O objetivo do trabalho é a implementação de um simulador orientado a eventos discretos
em uma fila seguindo as políticas de atendimento First Come First Served (FCFS) com tempo de simulação virtual. Visamos compreender a fila M/M/1 e entender a relação entre a fila M/M/1 e ruína do apostador.
As simulações presentes foram executadas nas máquinas do google colab, com uma T4-GPU no colab. Os casos utilizados foram analisados com 3200 rodadas e 100 clientes e com 10000 rodadas e 1000 clientes, exceções serão especificadas no relatório

2. Filas M/M/1

Nessa primeira parte do trabalho focaremos no desenvolvimento e estudo da fila M/M/1 visando entender

- Como se comporta em regime transiente
- Como se comporta em regime estacionário

Nos 4 cenários a seguir

Fila M/M/1	ρ	λ	μ
Cenário 1	0.5	1	2
Cenário 2	0.5	2	4
Cenário 3	1.05	1.05	1
Cenário 4	1.10	1.10	1

2.1 A implementação:

2.1.1 Intervalo de confiança

Para estimar os intervalos de confiança utilizamos o teste-t

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

fórmula

Para obtermos o intervalo de confiança, na linha 22, somamos e subtraímos com o valor obtido pelo teste t.

```
1 class Estimator:
2   def __init__(self):
3     self.samples_list = [] # amostras
4     self.samples_sum = 0.0 # Soma das amostras
5     self.squares_sum = 0.0 # Soma dos quadrados das amostras
6     self.n = 0 # Numero de amostras
7   def add_sample(self, sample):
8     self.samples_list.append(sample)
9     self.samples_sum += sample
10    self.squares_sum += (sample**2)
11    self.n += 1
12  def get_samples_list(self):
13    return self.samples_list
14  def mean(self):
15    return self.samples_sum / self.n
16  def variance(self):
17    term1 = self.squares_sum / (self.n - 1)
18    term2 = (self.samples_sum**2) / (self.n * (self.n - 1))
19    return term1 - term2
20  def tstudent_ci(self):
21    term = T_PERCENTILE * (math.sqrt(self.variance())/math.sqrt(N_RODADAS))
22    return self.mean() - term, self.mean() + term
23  def clear(self):
24    self.samples_sum = 0.0
25    self.samples_list = []
26    self.squares_sum = 0.0
27    self.n = 0
28
```

2.1.2 Implementação de Fila

Exemplo de uma depuração feita com os prints, indicando as chegadas de clientes e saídas.

Nesses primeiros prints, temos a chegada do primeiro cliente, onde marcamos o tempo de simulação e tipo desse evento. Logo em seguida, esse primeiro cliente é servido, deixando a fila vazia.

```
COMEÇANDO RODADA: 0

New Round
0 de  5
Eventos na fila de espera: 0
tempo de simulação 0.16910308517481865
Evento atual:
event_type: ARRIVAL
t: 0.16910308517481865
client_id: 0

New Round
0 de  5
Eventos na fila de espera: 0
tempo de simulação 0.30766910996553587
Evento atual:
event_type: END_OF_SERVICE
t: 0.30766910996553587
client_id: 0

[] Fila de espera vazia
```

Depois, temos dois eventos de chegadas em seguida. Do cliente 1 e do cliente 2.

```
New Round
1 de  5
Eventos na fila de espera: 0
tempo de simulação 1.0352436713337052
Evento atual:
event_type: ARRIVAL
t: 1.0352436713337052
client_id: 1


New Round
1 de  5
Eventos na fila de espera: 0
tempo de simulação 1.7060918893726598
Evento atual:
event_type: ARRIVAL
t: 1.7060918893726598
client_id: 2
```

Em seguida, temos os eventos dos clientes 1 e 2 sendo servidos. Onde sempre estimamos o tempo no sistema que o cliente ficou na fila de espera e o tempo que ficou sendo servido.

```
New Round
1 de  5
Eventos na fila de espera: 1
tempo de simulação 1.7108680195295887
Evento atual:
event_type: END_OF_SERVICE
t: 1.7108680195295887
client_id: 1


New Round
2 de  5
Eventos na fila de espera: 0
tempo de simulação 1.8326696162185865
Evento atual:
event_type: END_OF_SERVICE
t: 1.8326696162185865
client_id: 2

[] Fila de espera vazia
```

A implementação pode ser vista no colab.
https://colab.research.google.com/drive/1mIQ8VV4tkDDTjp49PZUJ3vR6sT1SKdWq?usp=sharing

2.2 Cenários:

1º Set de Valores de Entrada

- Número de rodadas = 3.200
- mínimo de 100 clientes atendidos

2º Set de Valores de Entrada

- Número de rodadas = 10.000
- mínimo de 1000 clientes atendidos

> Retirado do enunciado do trabalho : *Em aula, nós argumentamos que o número médio de clientes nos sistemas 1 e 2 será o mesmo, mas que o tempo médio de espera no sistema 2 será metade daquele experimentado no sistema 1. Um dos propósitos do trabalho é você verificar que isso de fato ocorre na prática. Ou seja, você deverá verificar, na prática, que o sistema 1 se comporta como o sistema 2, mas em câmera lenta.*

- Observamos que realmente acontece e poderá ser visto nas tabela a seguir

- Outra observação importante a ser ressaltada é que, nos gráficos de ***tempo de serviço x densidade****,* de **amostras de E(W)** e de **amostras de E(Nq)**, conforme aumentamos a quantidade de rodadas o gráfico tende a se aproximar de uma distribuição normal, devido ao Teorema Central do Limite, fica evidente quando comparamos com o segundo set de dados
- Também vemos que a CDF assume o formato de uma distribuição exponencial.
- A Média de vezes que o sistema atinge o estado zero por rodada também é menor nos casos 3 e 4, como o $\rho > 1$ nesses cenários, então o sistema tende a estar mais vezes ocupado e não zerar.

2.2.1 Cenário 1

Fila M/M/1	ρ	λ	μ
Cenário 1	0.5	1	2

Teste de corretude

```
[CORRETUDE] Calculando a Lei de little com base nos estimadores de
[CORRETUDE] media de tempo na fila/ media de tempo total  =  lambda
[CORRETUDE] Littles Law: 1.0190316643963804
```

Calculando o lambda pelas médias de tempo obtidas na simulação, encontramos justamente o lambda proposto.

```
print(f"[CORRETUDE] Rho: {estimador_tempo_servindo.mean()/estimador_tempo_total.mean()}\n")
#lambda = 1
```

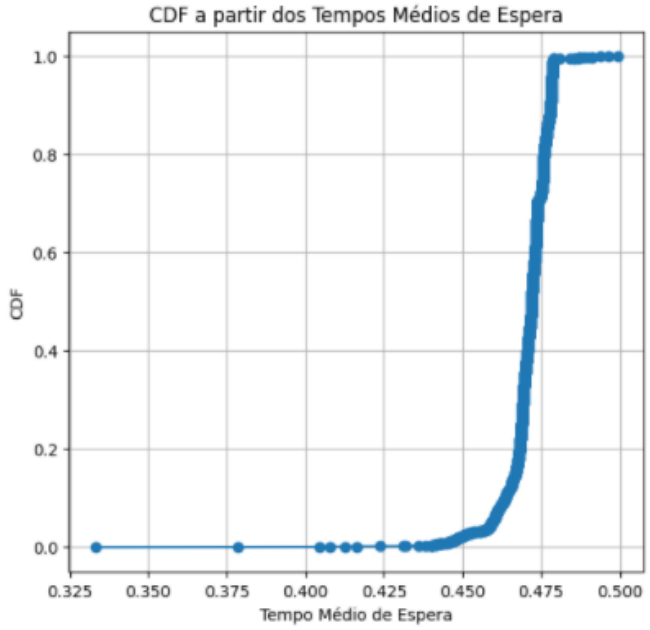
```
[CORRETUDE] media do tempo de servico: 49.71868776216045 - 50.061902341576165
[CORRETUDE] media do tempo total no sistema (tempo ocioso + tempo de servico): 100.7530834619649 - 101.45329794334808
[CORRETUDE] Rho: 0.49345915499932325
```

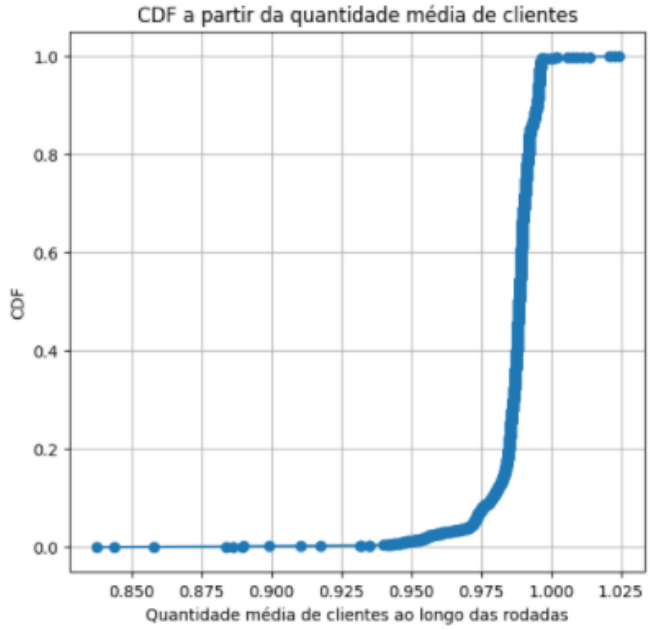
Mesma coisa para o cálculo do Rho

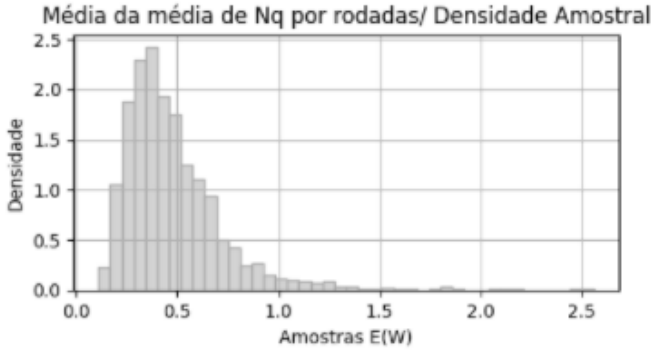
Tabela de Valores Obtidos para 1º Set (3200 rodadas e 100 clientes)

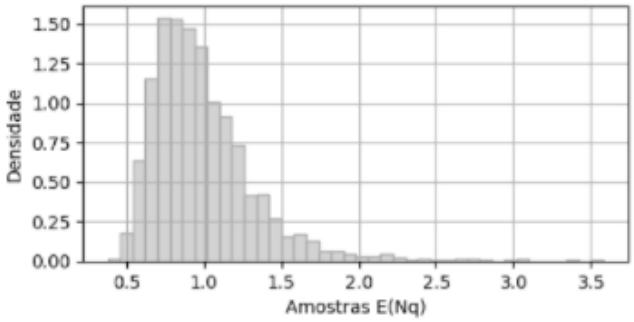
Medida	**Intervalo de Confiança**
média Nq - número de clientes na fila	0.984 - 1.008
variância Nq	1.108 - 1.191
Média W - Tempo de espera na fila	0.470 - 0.487
Variância W	0.624 - 0.669
Média de vezes que o sistema atinge o estado zero por rodada	50.201 - 50.780
média do tempo total de simulação	100.75 - 101.45
média do tempo de serviço da simulação	49.71 - 50.06
média do tempo total dos clientes no sistema	0.943 - 1.011

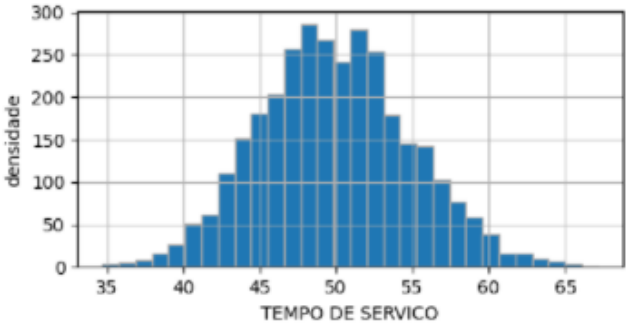
1º Set de Valores de Entrada

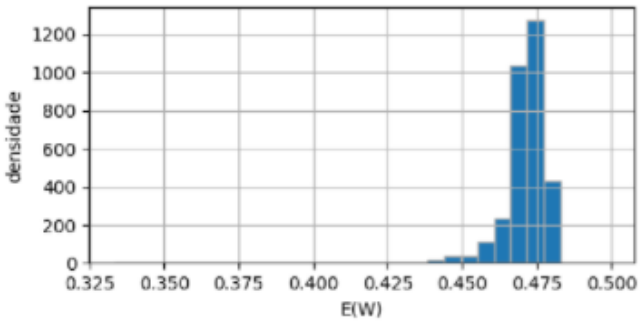


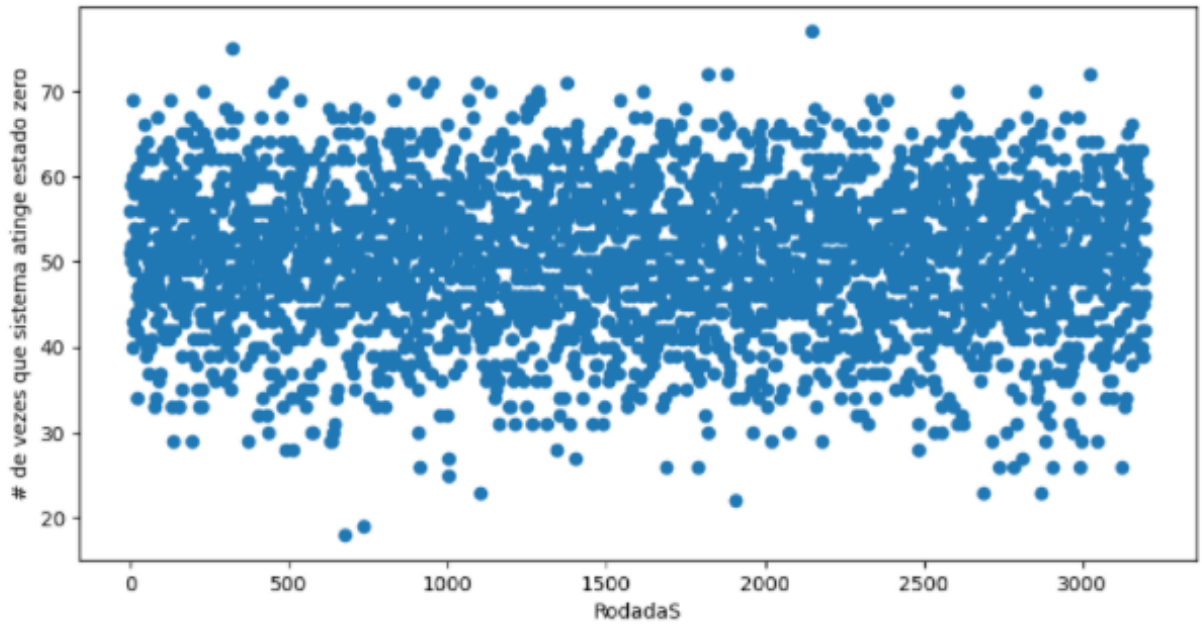


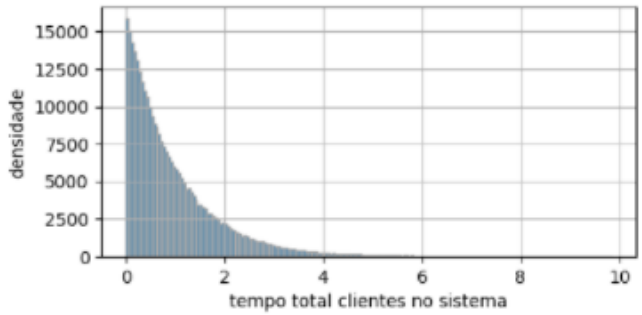









de vezes que sistema atinge estado zero
RodadaS


densidade
tempo total clientes no sistema


densidade
tempo de serviço clientes no sistema


Gráfico da transiente - E[Nq]
E[Nq]
Rodadas


Gráfico da transiente - E[W]
E[W]
Rodadas

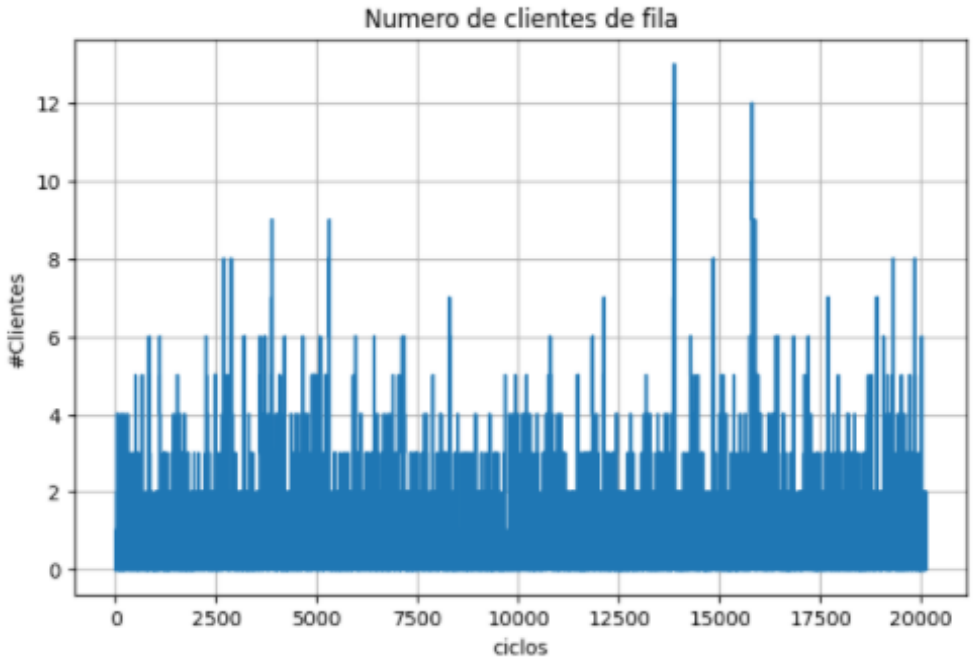
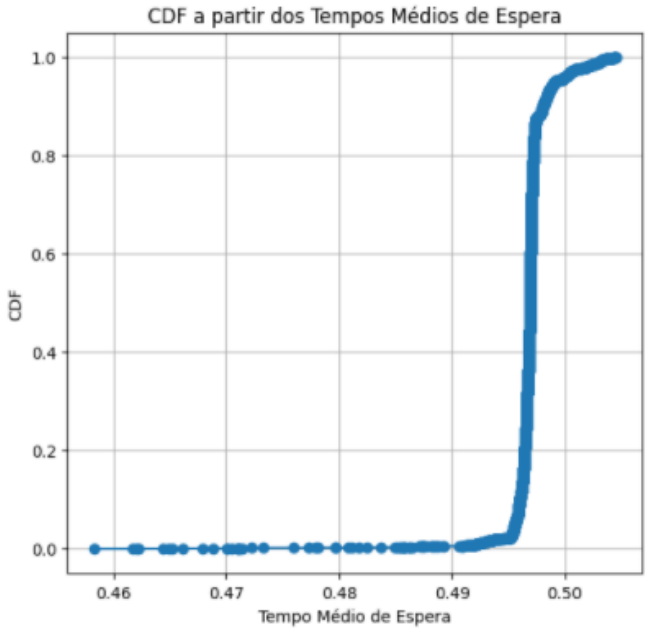
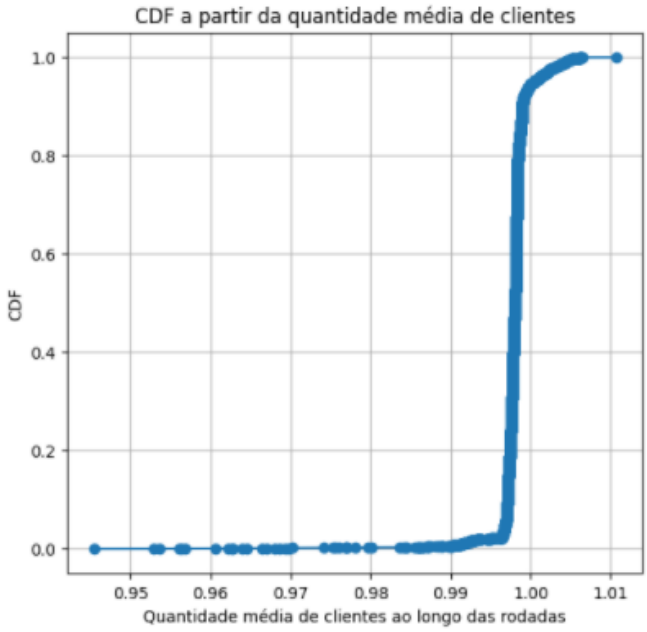

Numero de clientes de fila
#Clientes
ciclos
0
2
4
6
8
10
12
0
2500
5000
7500
10000
12500
15000
17500
20000

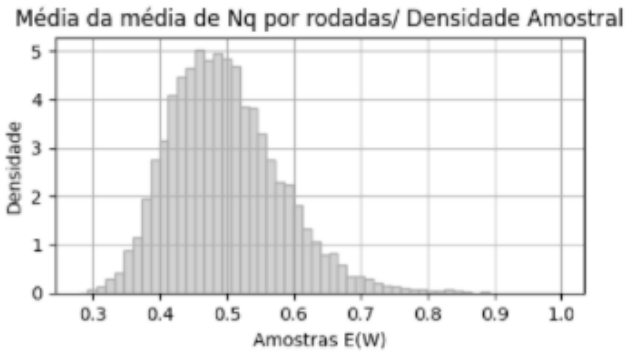
Tabela de Valores Obtidos para 2º Set (10000 rodadas e 1000 clientes)

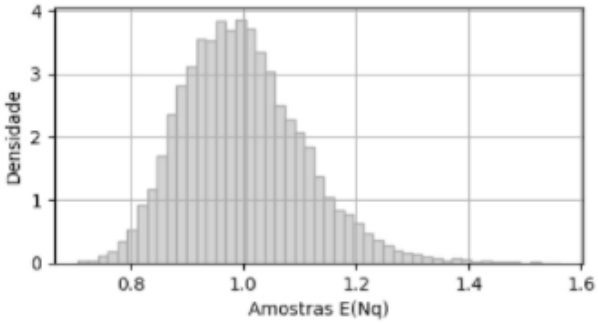
Medida	**Intervalo de Confiança**
média Nq - número de clientes na fila	0.997 - 1.001
variância Nq	1.227 - 1.245
Média W - Tempo de espera na fila	0.496 - 0.499
Variância W	0.732 - 0.743
Média de vezes que o sistema atinge o estado zero por rodada	500.073 - 501.140
média do tempo total de simulação	1000.67 - 1001.90
média do tempo de serviço da simulação	499.67 - 500.29
média do tempo total dos clientes no sistema	0.978 - 1.017

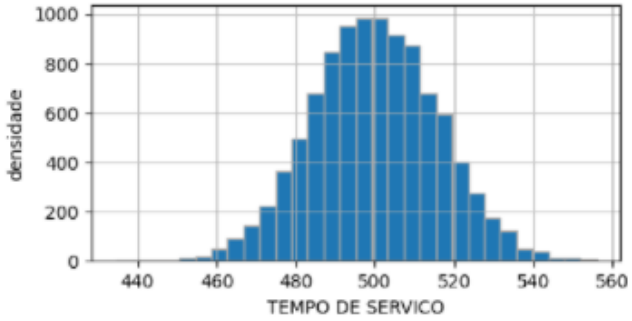
2º Set de Valores de Entrada

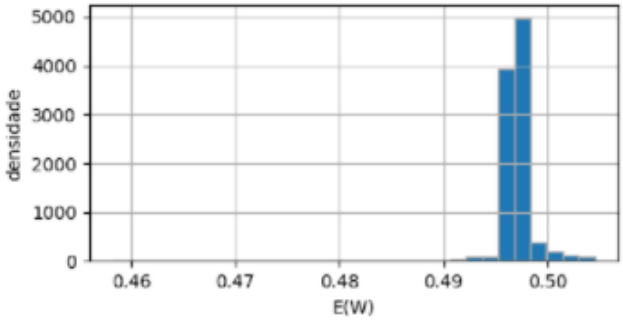


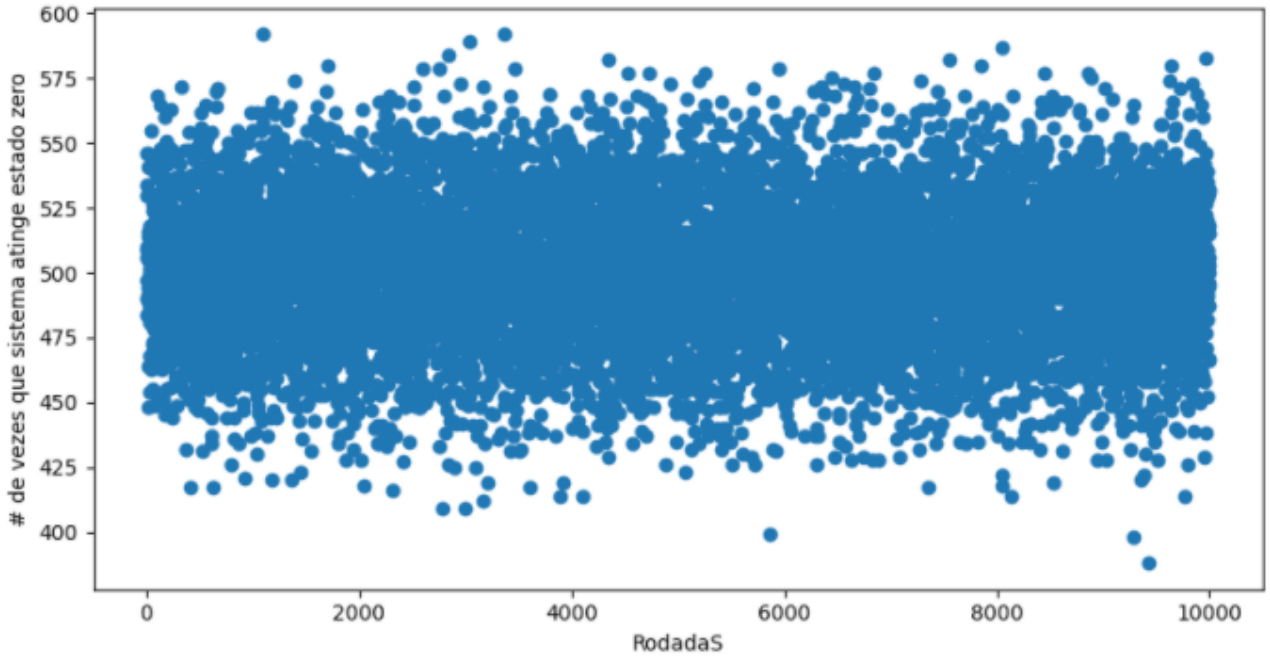



Média da média de Nq por rodadas/ Densidade Amostral
Densidade
Amostras E(W)


Densidade
Amostras E(Nq)


densidade
TEMPO DE SERVICO


densidade
E(W)

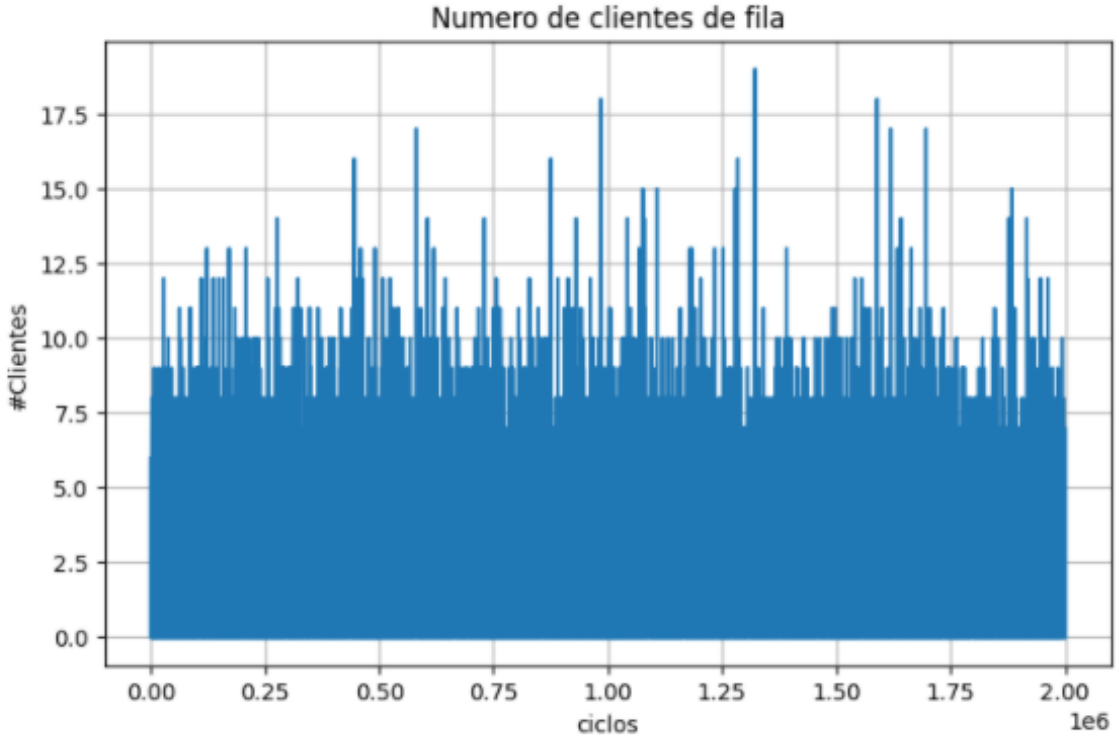

de vezes que sistema atinge estado zero
RodadaS


densidade
tempo total clientes no sistema


densidade
tempo de serviço clientes no sistema


Gráfico da transiente - E[Nq]
E[Nq]
Rodadas


Gráfico da transiente - E[W]
E[W]
Rodadas


Numero de clientes de fila
#Clientes
ciclos
1e6

2.2.2 Cenário 2

Fila M/M/1	ρ	λ	μ
Cenário 2	0.5	2	4

Teste de corretude

```
[CORRETUDE] Calculando a Lei de little com base nos estimadores de
[CORRETUDE] media de tempo na fila/ media de tempo total  =  lambda
[CORRETUDE] Littles Law: 2.0380633287927608
```

Calculando o lambda pelas médias de tempo obtidas na simulação, encontramos justamente o lambda proposto.

```
print(f"[CORRETUDE] Rho: {estimador_tempo_servindo.mean()/estimador_tempo_total.mean()}\n")
#lambda = 1
```

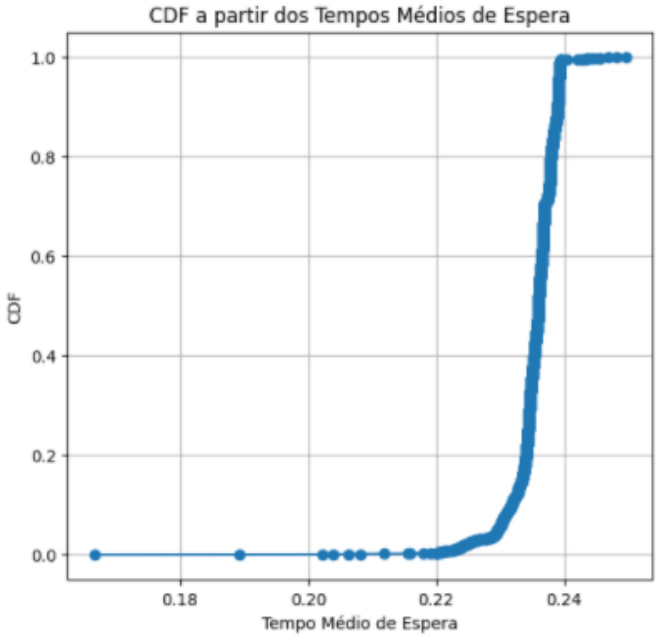
```
[CORRETUDE] media do tempo de servico: 24.859343881080225 - 25.030951170788082
[CORRETUDE] media do tempo total no sistema (tempo ocioso + tempo de servico): 50.37654173098245 - 50.72664897167404
[CORRETUDE] Rho: 0.49345915499932325
```

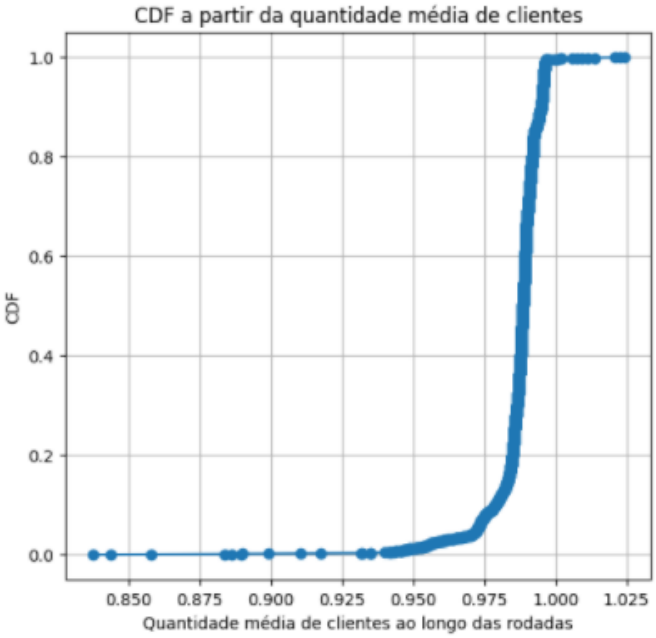
Mesma coisa para o cálculo do Rho

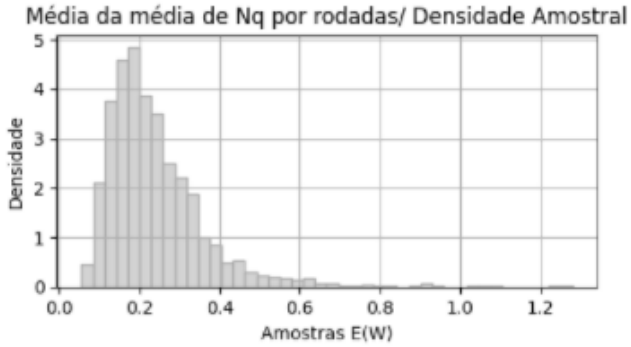
Tabela de Valores Obtidos para 1º Set (3200 rodadas e 100 clientes)

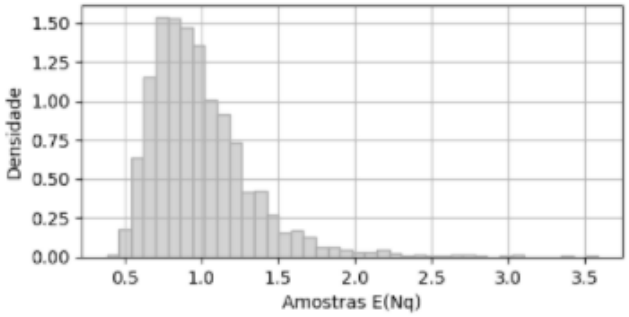
Medida	**Intervalo de Confiança**
média Nq - número de clientes na fila	0.984 - 1.008
variância Nq	1.108 -1.191
Média W - Tempo de espera na fila	0.235 - 0.243
Variância W	0.156 - 0.167
Média de vezes que o sistema atinge o estado zero por rodada	50.201 - 50.780
média do tempo total de simulação	50.37 - 50.72
média do tempo de serviço da simulação	24.85 - 25.03
média do tempo total dos clientes no sistema	0.472 - 0.505

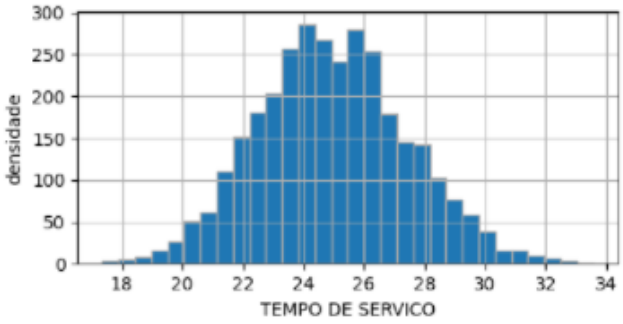
1º Set de Valores de Entrada

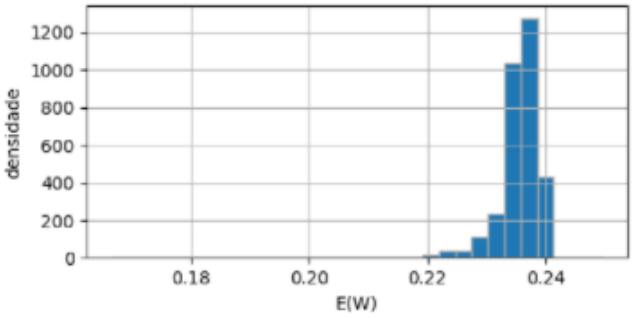

CDF a partir dos Tempos Médios de Espera
CDF
Tempo Médio de Espera

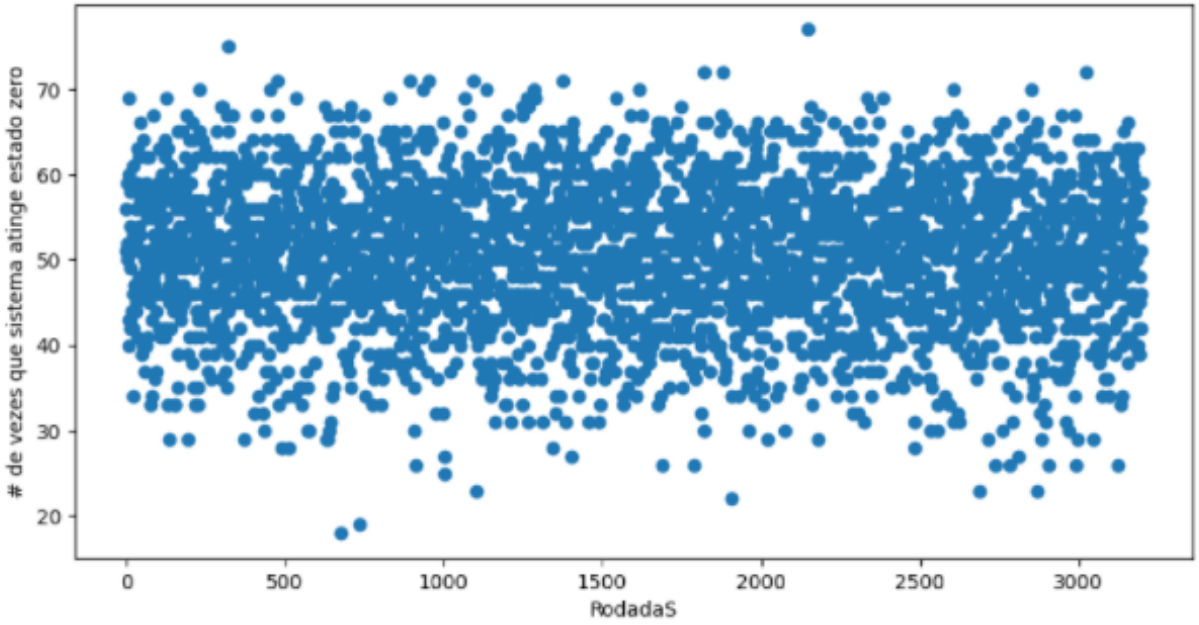

CDF a partir da quantidade média de clientes
CDF
Quantidade média de clientes ao longo das rodadas

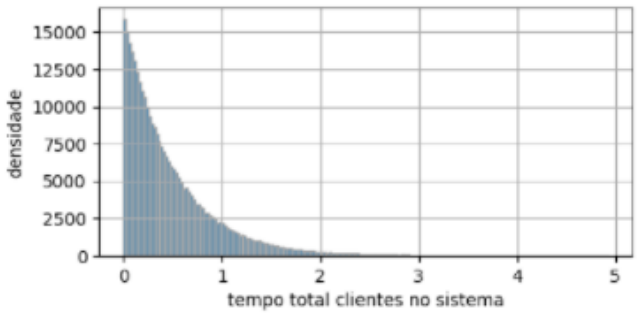

Média da média de Nq por rodadas/ Densidade Amostral
Densidade
Amostras E(W)

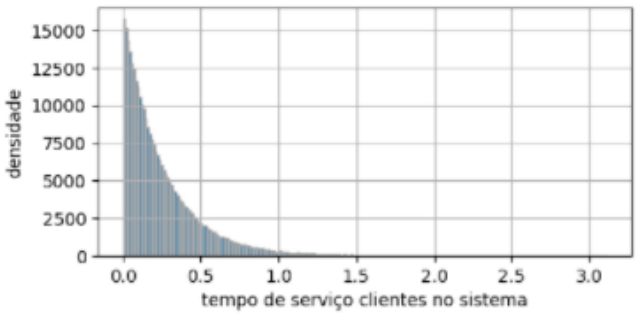

Densidade
Amostras E(Nq)


densidade
TEMPO DE SERVICO


densidade
E(W)


de vezes que sistema atinge estado zero
RodadaS


densidade
tempo total clientes no sistema


densidade
tempo de serviço clientes no sistema

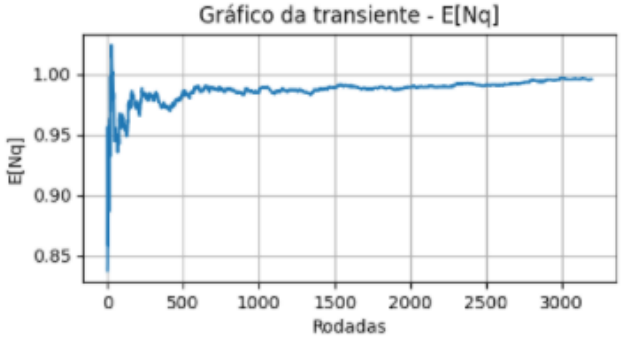


Gráfico da transiente - E[Nq]
E[Nq]
Rodadas


Gráfico da transiente - E[W]
E[W]
Rodadas

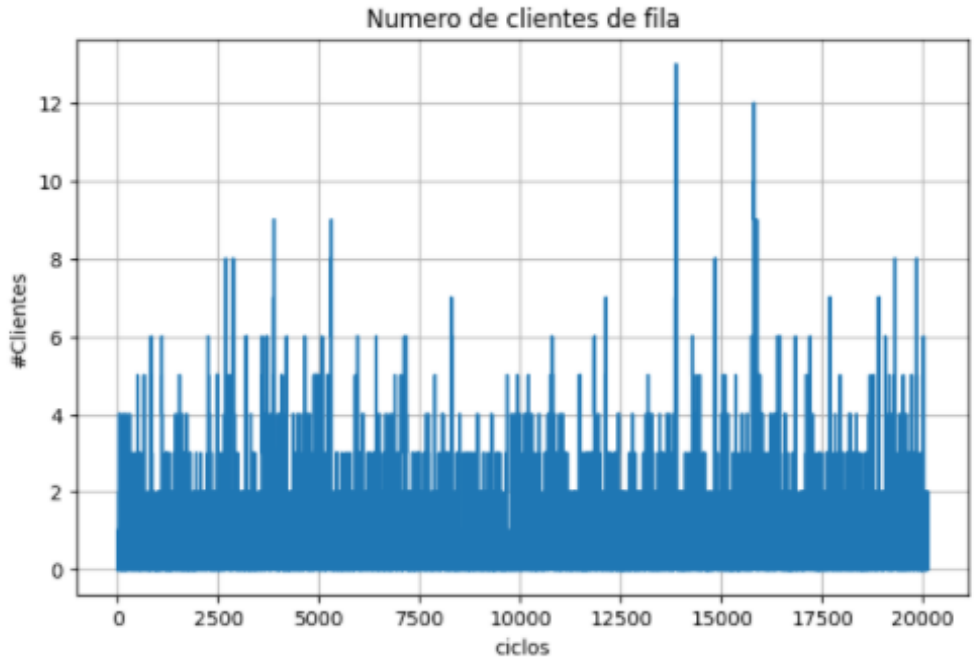
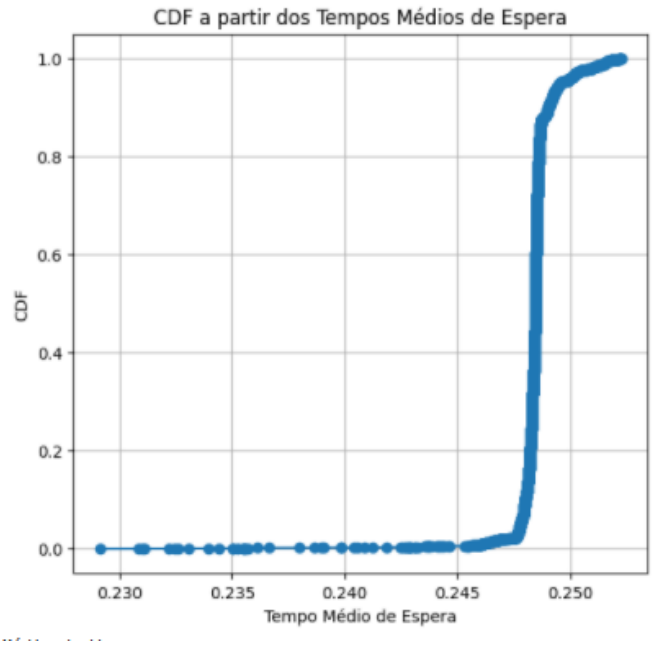
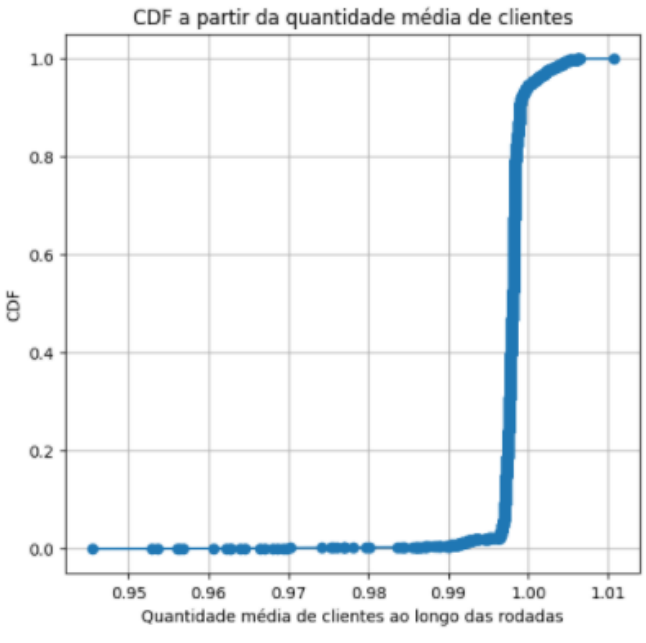


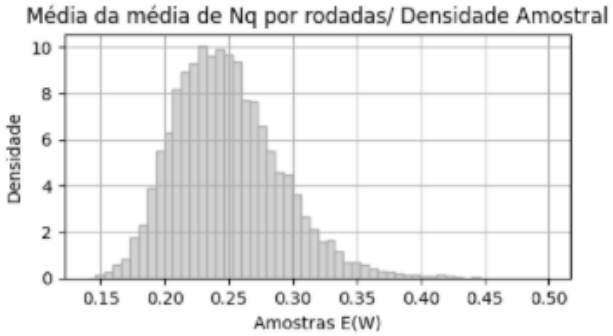
Tabela de Valores Obtidos para 2º Set (10000 rodadas e 1000 clientes)

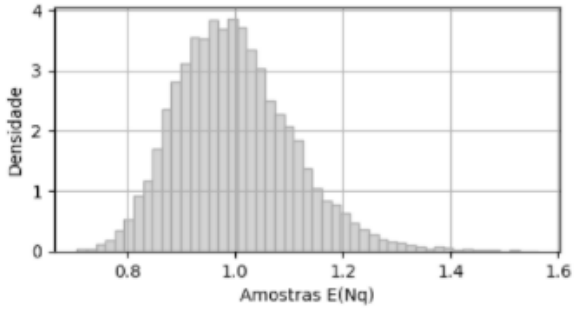
Medida	**Intervalo de Confiança**
média Nq - número de clientes na fila	0.997 - 1.001
variância Nq	1.227 - 1.245
Média W - Tempo de espera na fila	0.248 - 0.249
Variância W	0.183 - 0.186
Média de vezes que o sistema atinge o estado zero por rodada	500.073 - 501.140
média do tempo total de simulação	500.33 - 500.95
média do tempo de serviço da simulação	249.83 - 250.14
média do tempo total dos clientes no sistema	0.489 - 0.508

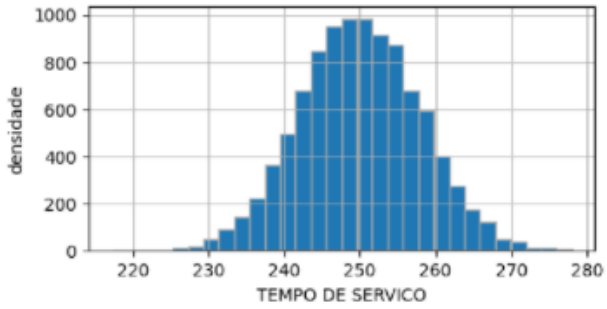
2º Set de Valores de Entrada

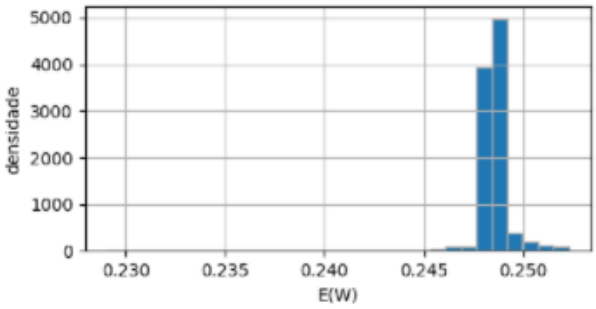


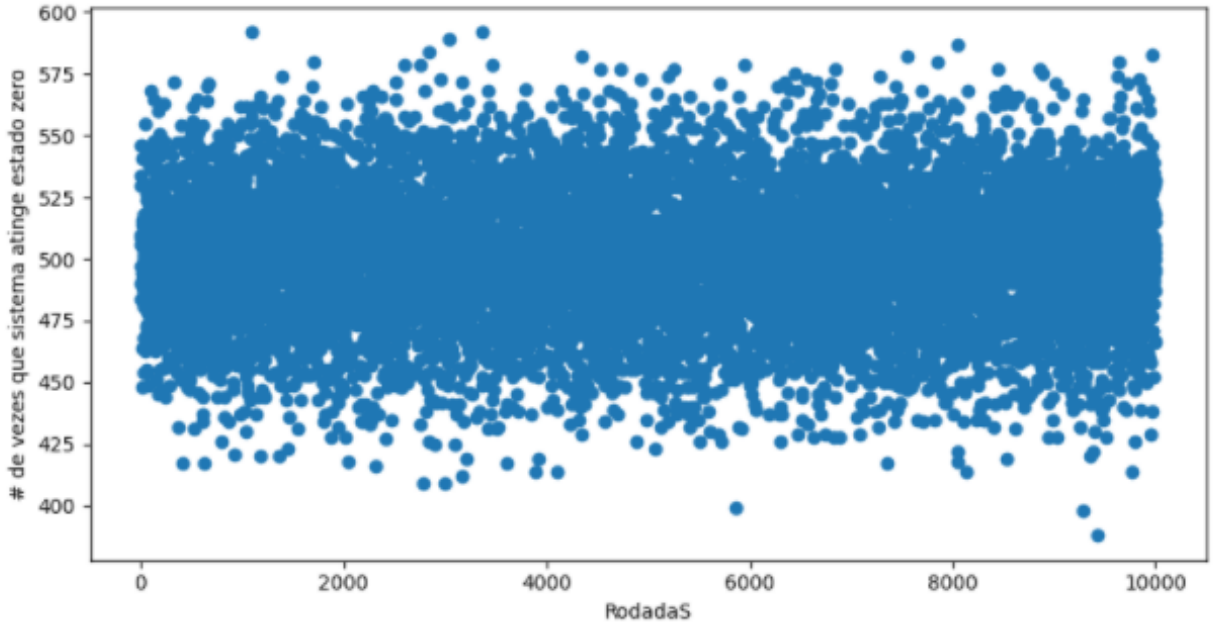


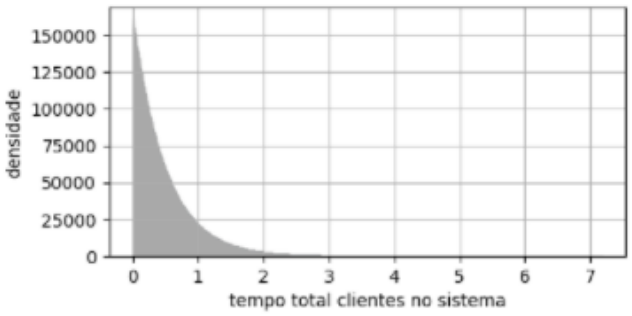








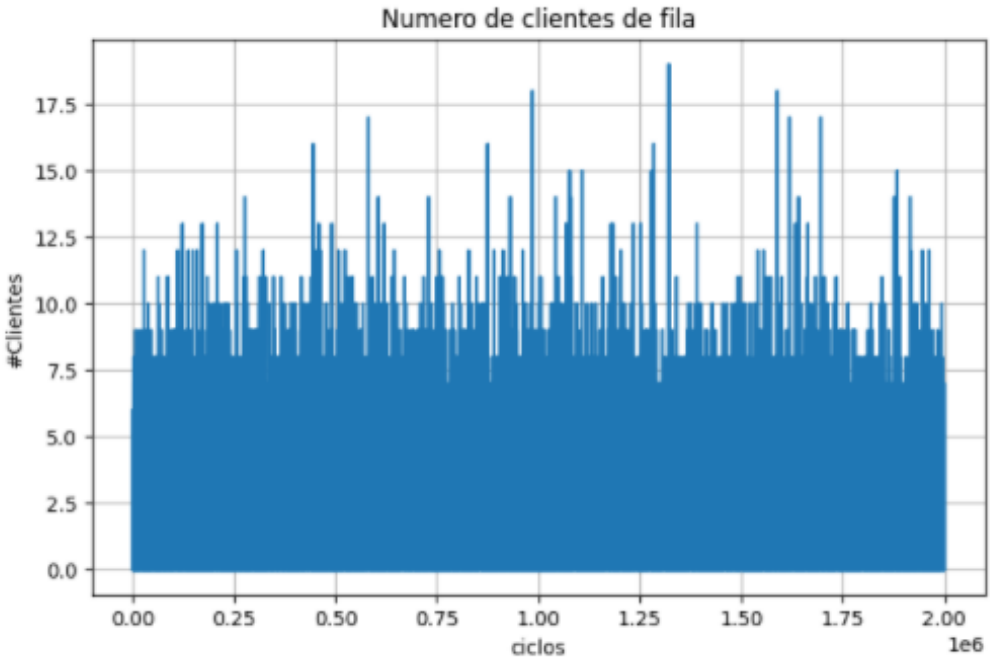

de vezes que sistema atinge estado zero
RodadaS


densidade
tempo total clientes no sistema


densidade
tempo de serviço clientes no sistema


Gráfico da transiente - E[Nq]
E[Nq]
Rodadas


Gráfico da transiente - E[W]
E[W]
Rodadas



2.2.3 Cenário 3

Fila M/M/1	ρ	λ	μ
Cenário 3	1.05	1.05	1

Teste de corretude

```
[CORRETUDE] Calculando a Lei de little com base nos estimadores de
[CORRETUDE] media de tempo na fila/ media de tempo total  =  lambda
[CORRETUDE] Littles Law: 1.0524998138203132
```

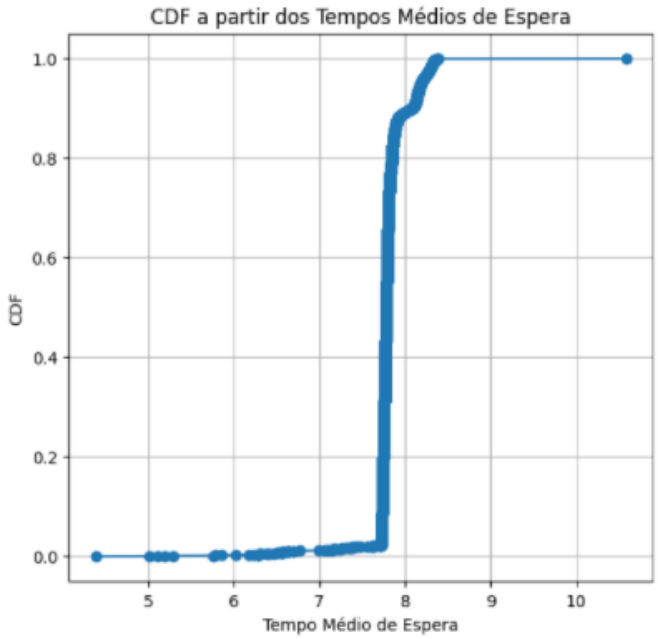
Calculando o lambda pelas médias de tempo obtidas na simulação, encontramos justamente o lambda proposto.

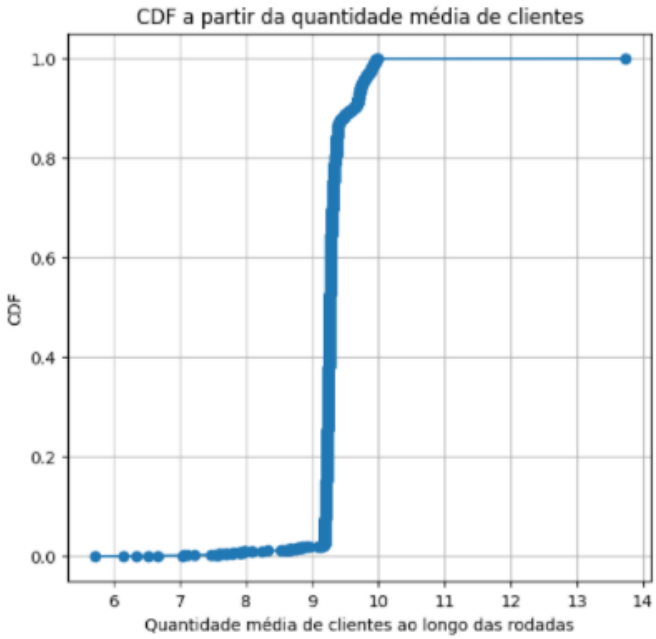
Tabela de Valores Obtidos para 1º Set (3200 rodadas e 100 clientes)

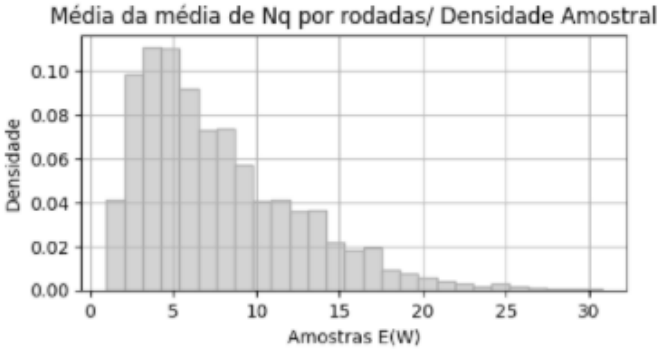
Medida	**Intervalo de Confiança**
média Nq - número de clientes na fila	9.004 - 9.377
variância Nq	36.514 - 39.501
Média W - Tempo de espera na fila	7.563 - 7.896
Variância W	29.578 - 31.938
Média de vezes que o sistema atinge o estado zero por rodada	8.013 - 8.490

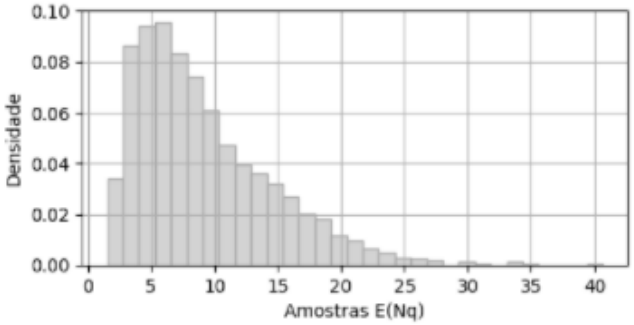
média do tempo total de simulação	108.73 - 109.36
média do tempo de serviço da simulação	99.91 - 100.61
média do tempo total dos clientes no sistema	8.476 - 8.988

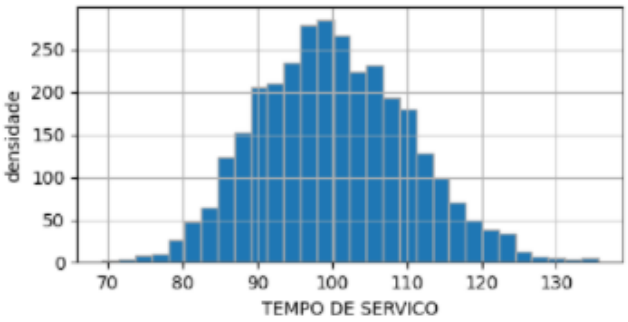
1º Set de Valores de Entrada

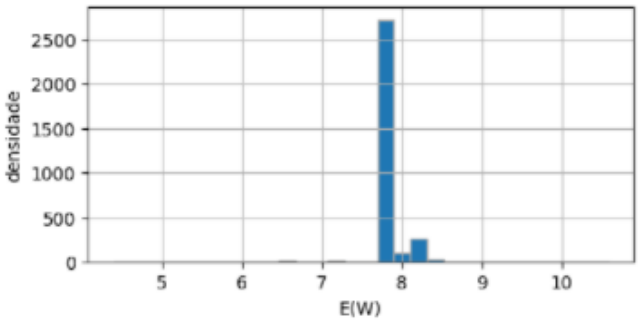


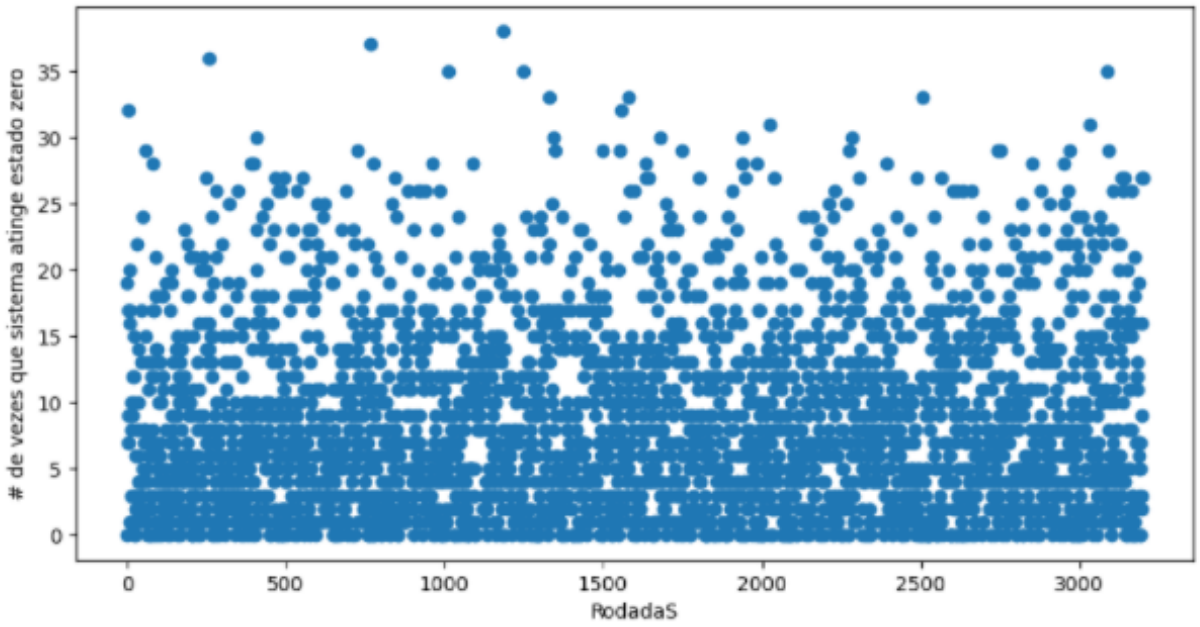


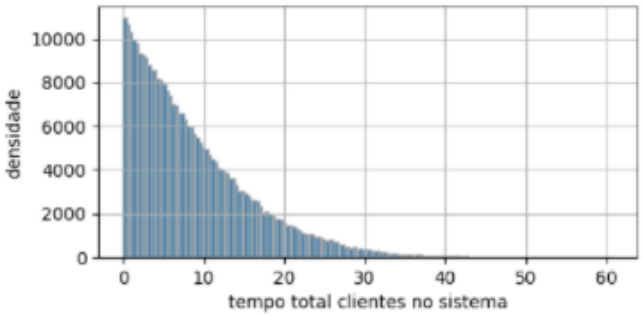









de vezes que sistema atinge estado zero
RodadaS


densidade
tempo total clientes no sistema


densidade
tempo de serviço clientes no sistema


Gráfico da transiente - E[Nq]
E[Nq]
Rodadas


Gráfico da transiente - E[W]
E[W]
Rodadas

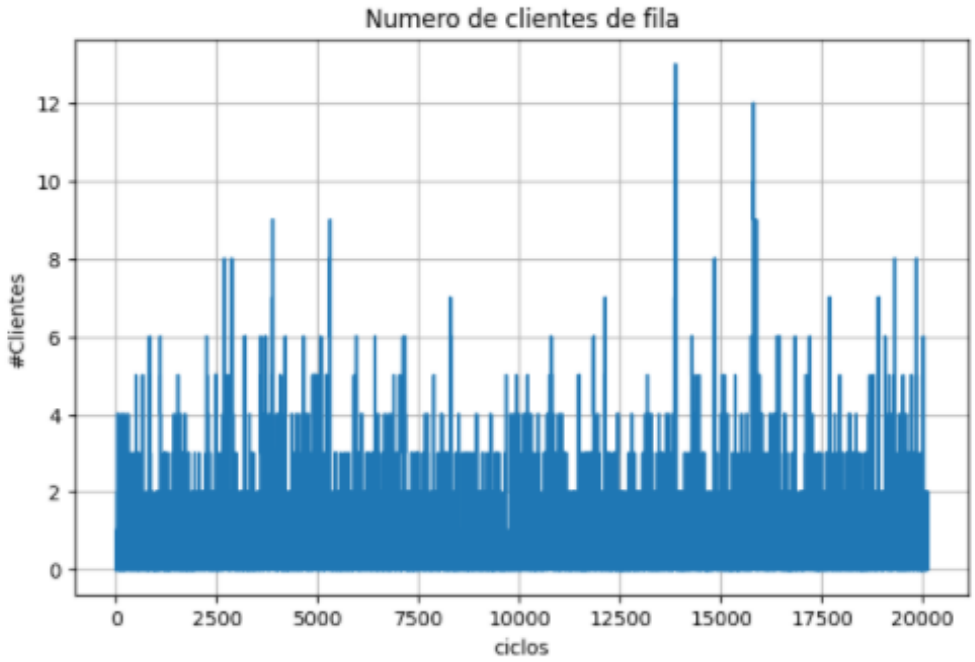
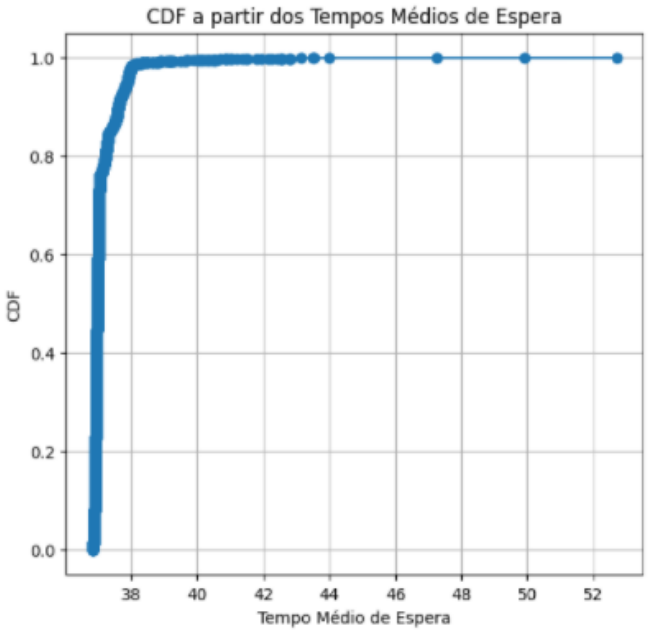
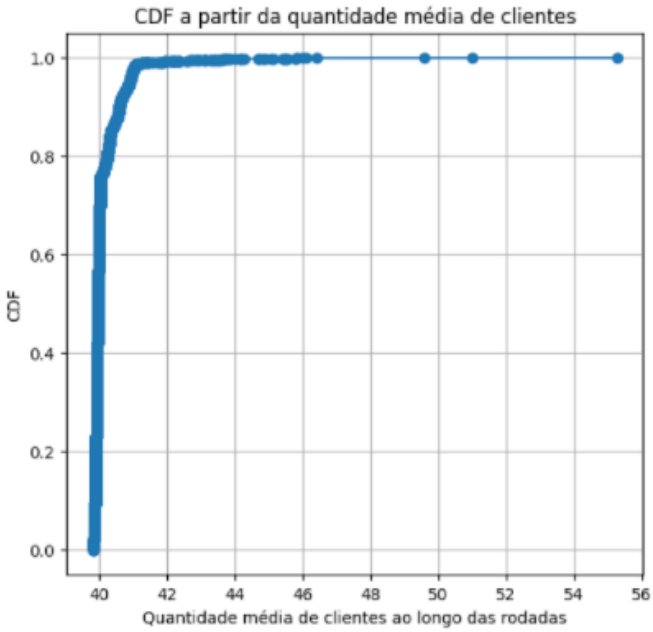


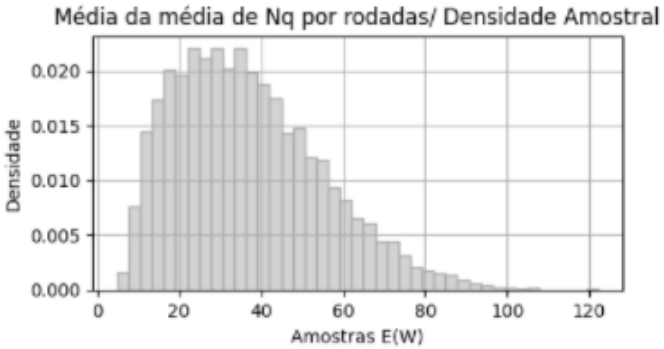
Tabela de Valores Obtidos para 2º Set (10000 rodadas e 1000 clientes)

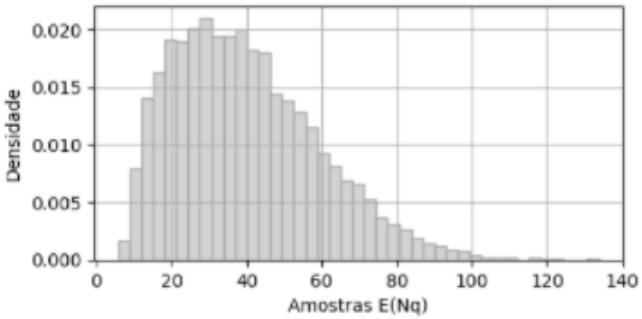
Medida	**Intervalo de Confiança**
média Nq - número de clientes na fila	39.496 - 40.256
variância Nq	601.537 - 624.432
Média W - Tempo de espera na fila	36.547 - 37.255
Variância W	528.434 - 548.286
Média de vezes que o sistema atinge o estado zero por rodada	17.037 - 17.691
média do tempo total de simulação	1017.39 - 1018.50
média do tempo de serviço da simulação	999.84 - 1001.07
média do tempo total dos clientes no sistema	37.325 - 38.478

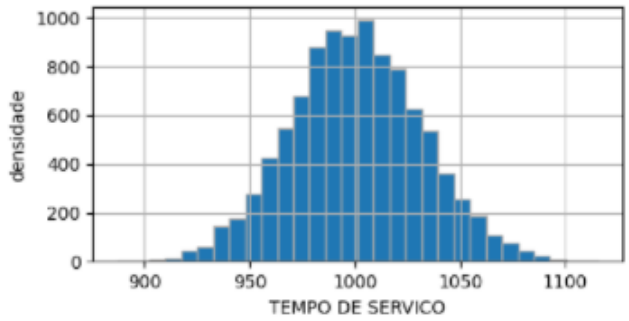
2º Set de Valores de Entrada

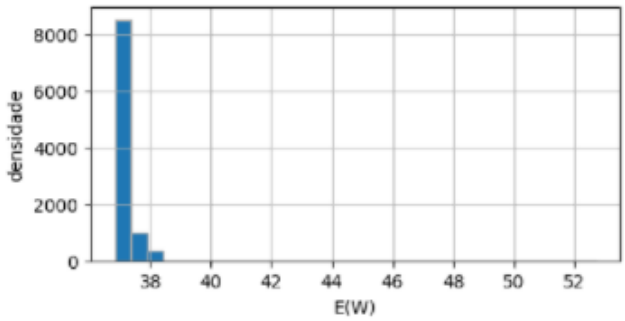

CDF a partir dos Tempos Médios de Espera
CDF
Tempo Médio de Espera

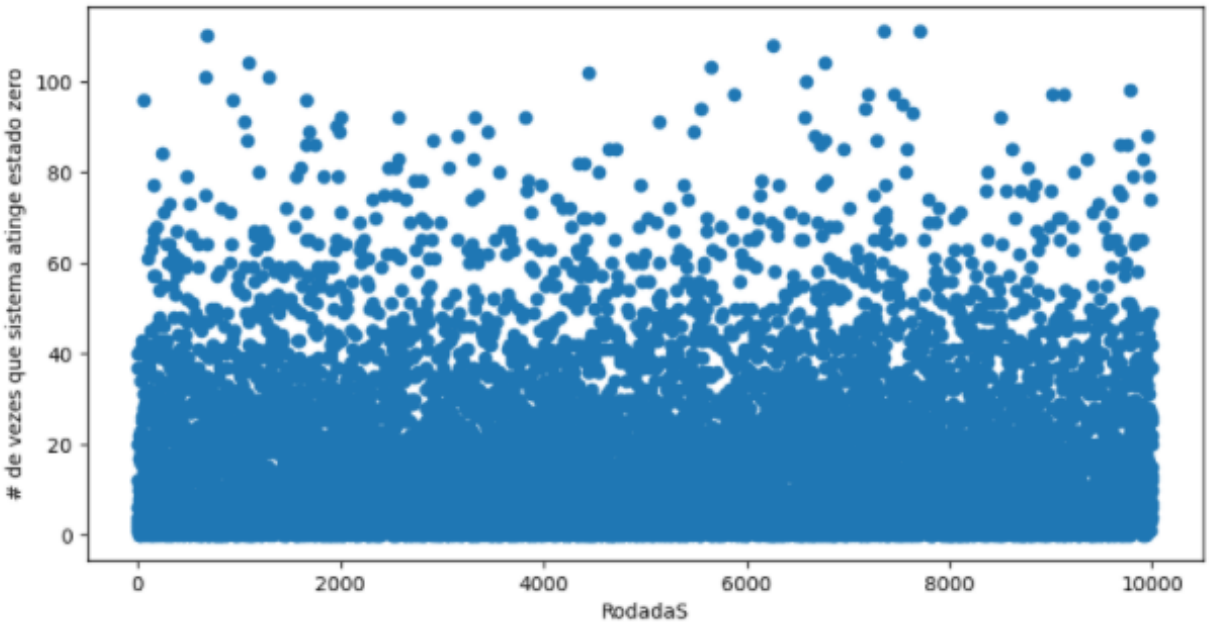

CDF a partir da quantidade média de clientes
CDF
Quantidade média de clientes ao longo das rodadas

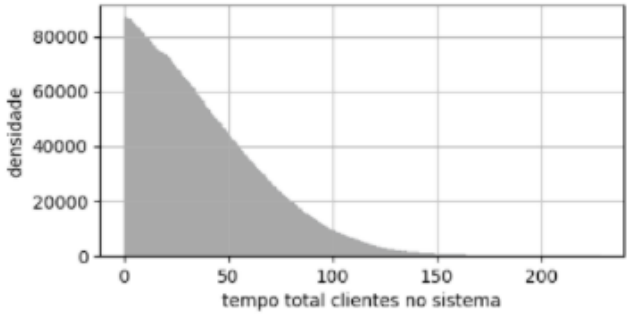

Média da média de Nq por rodadas/ Densidade Amostral
Densidade
Amostras E(W)


Densidade
Amostras E(Nq)


densidade
TEMPO DE SERVICO


densidade
E(W)

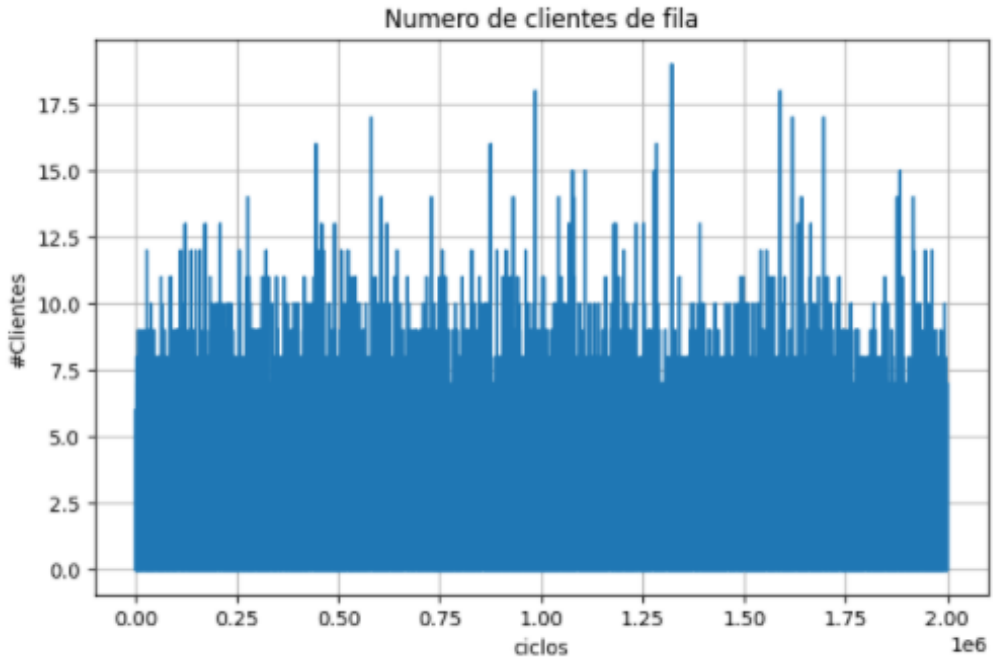

de vezes que sistema atinge estado zero
RodadaS


densidade
tempo total clientes no sistema


densidade
tempo de serviço clientes no sistema


Gráfico da transiente - E[Nq]
E[Nq]
Rodadas


Gráfico da transiente - E[W]
E[W]
Rodadas



2.2.4 Cenário 4

Fila M/M/1	ρ	λ	μ
Cenário 4	1.10	1.10	1

Teste de corretude

```
[CORRETUDE] Calculando a Lei de little com base nos estimadores de
[CORRETUDE] media de tempo na fila/ media de tempo total  =  lambda
[CORRETUDE] Littles Law: 1.1068215437990645
```

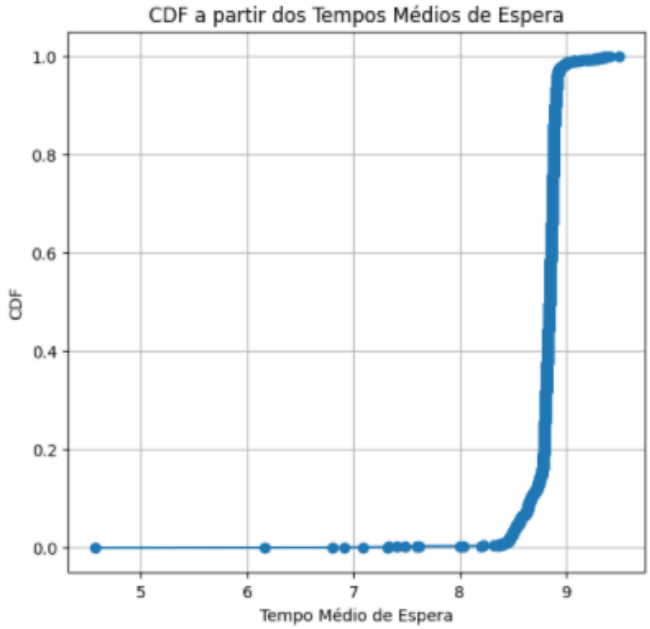
Calculando o lambda pelas médias de tempo obtidas na simulação, encontramos justamente o lambda proposto.

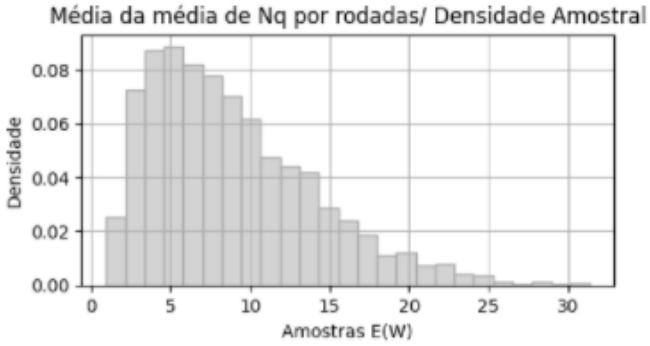
Tabela de Valores Obtidos para 1º Set (3200 rodadas e 100 clientes)

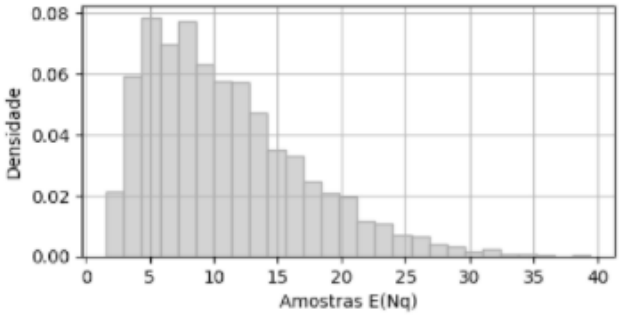
Medida	**Intervalo de Confiança**
média Nq - número de clientes na fila	10.734 - 11.159
variância Nq	48.741 - 52.577
Média W - Tempo de espera na fila	8.710 - 9.068
Variância W	36.255 - 38.939

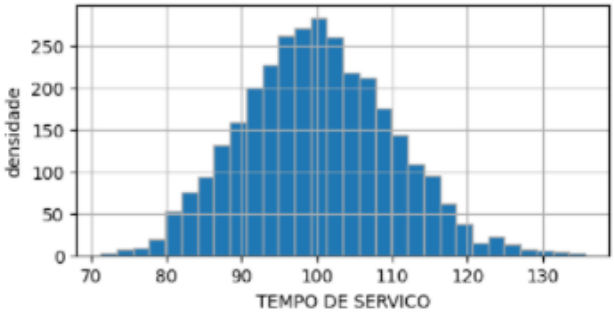
Média de vezes que o sistema atinge o estado zero por rodada	6.453 - 6.882
média do tempo total de simulação	106.75 - 107.38
média do tempo de serviço da simulação	99.76 - 100.46
média do tempo total dos clientes no sistema	9.611 - 10.169

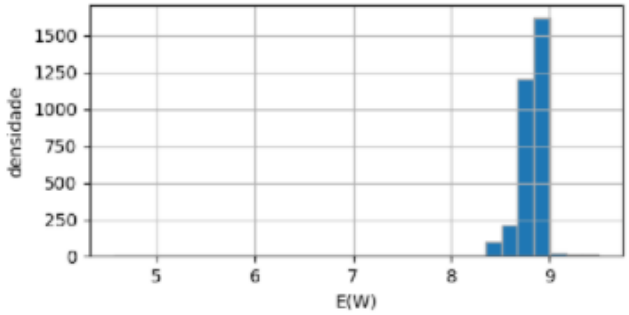
1º Set de Valores de Entrada

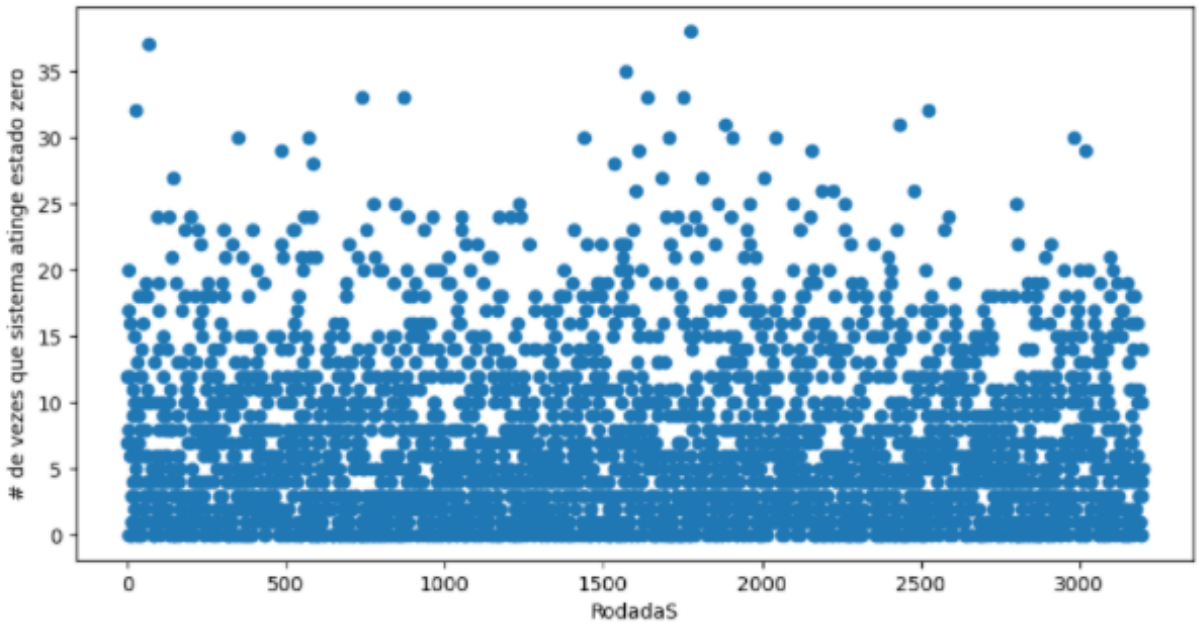




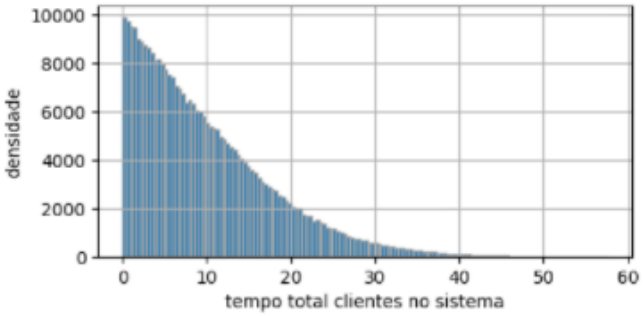








Tanto no cenário 4 como no 3, enxergamos, no gráfico acima, que a média de vezes que o sistema atinge zero está concentrado entre a faixa [0-5] do eixo y, enquanto que nos cenários 1 e 2, ela assume valores maiores. Isso se deve aos cenários 3 e 4 terem a taxa de chegadas maior que a taxa de saída, com $\rho > 1$.



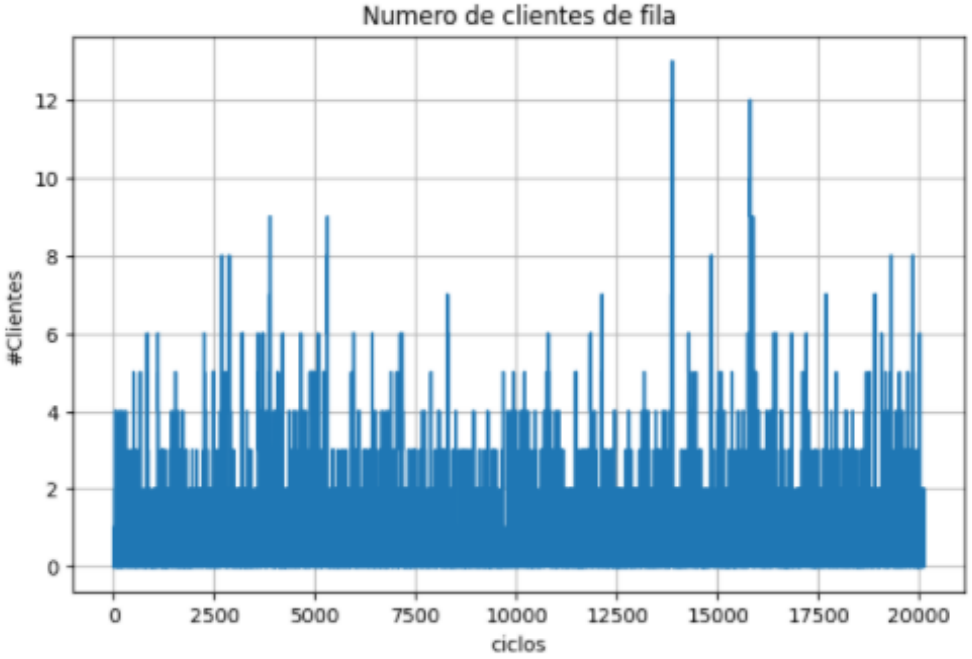
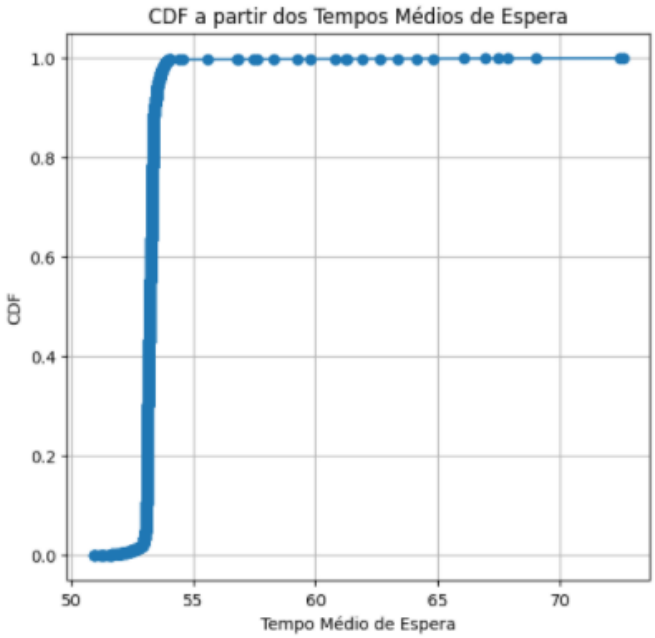


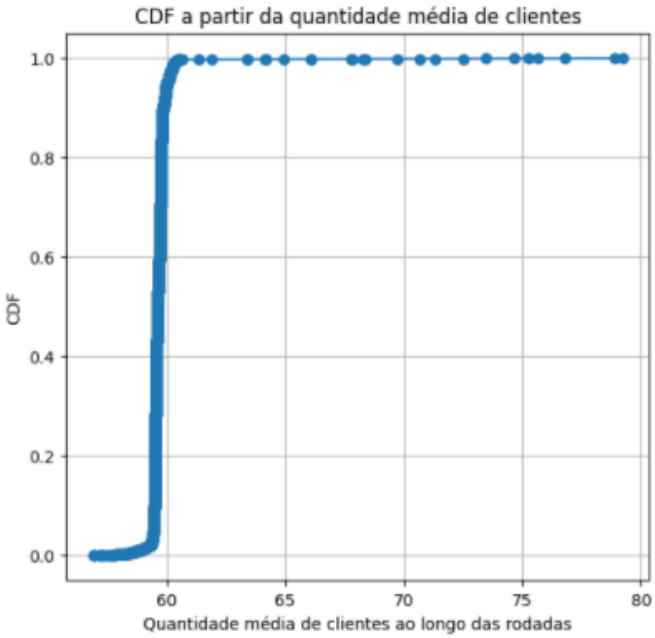
Tabela de Valores Obtidos para 2º Set (10000 rodadas e 1000 clientes)

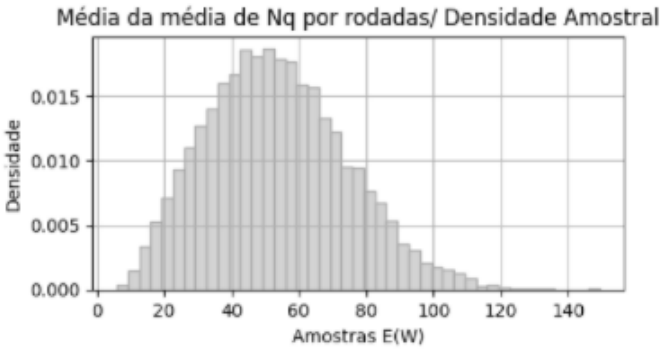
Medida	**Intervalo de Confiança**
média Nq - número de clientes na fila	59.368 - 60.290
variância Nq	1249.199 - 1287.536
Média W - Tempo de espera na fila	52.975 - 53.793

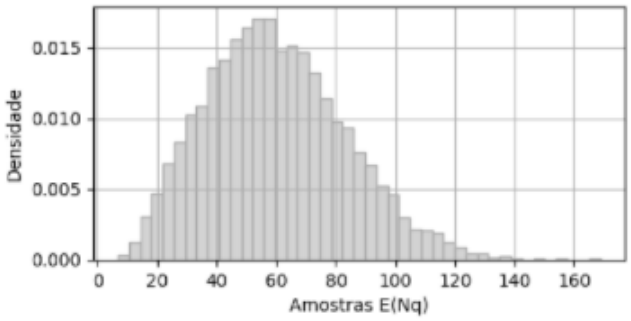
Variância W	1009.126 - 1039.443
Média de vezes que o sistema atinge o estado zero por rodada	9.669 - 10.075
média do tempo total de simulação	1009.24 - 1010.42
média do tempo de serviço da simulação	999.37 - 1000.61
média do tempo total dos clientes no sistema	53.635 - 55.133

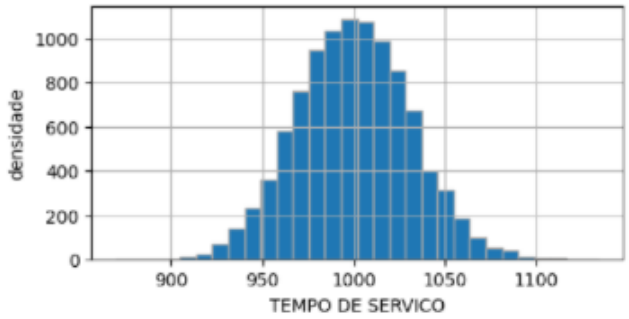
2º Set de Valores de Entrada

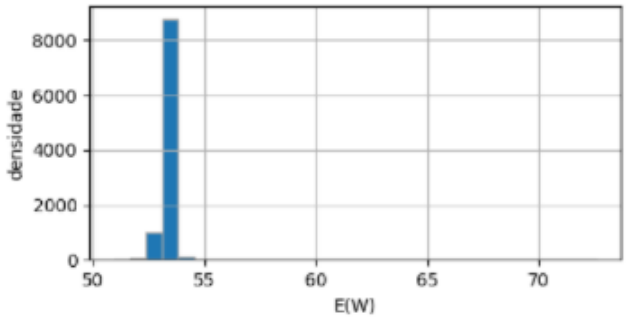


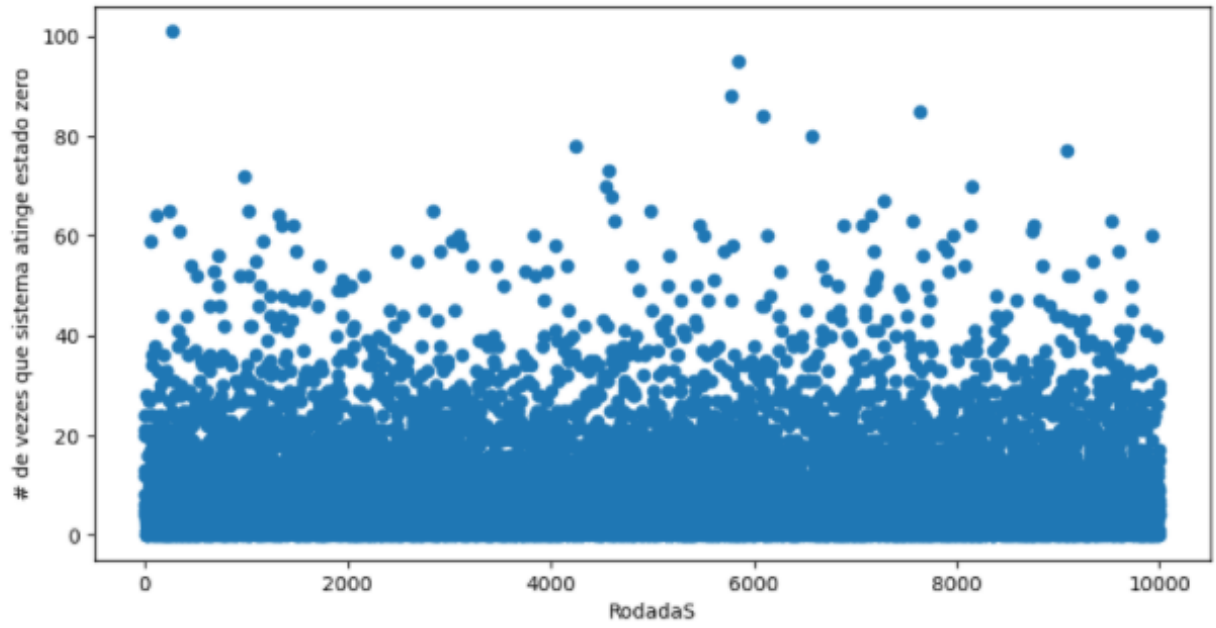


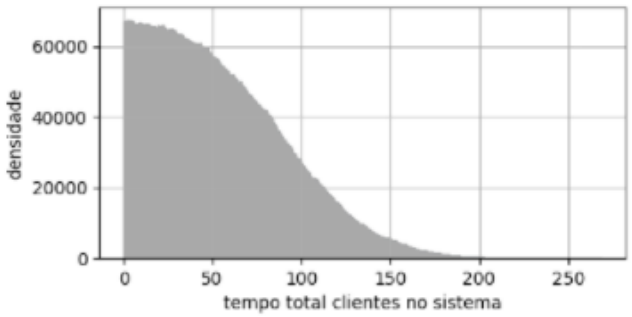

Média da média de Nq por rodadas/ Densidade Amostral
Densidade
0.015
0.010
0.005
0.000
0
20
40
60
80
100
120
140
Amostras E(W)


Densidade
0.015
0.010
0.005
0.000
0
20
40
60
80
100
120
140
160
Amostras E(Nq)


densidade
1000
800
600
400
200
0
900
950
1000
1050
1100
TEMPO DE SERVICO


densidade
8000
6000
4000
2000
0
50
55
60
65
70
E(W)

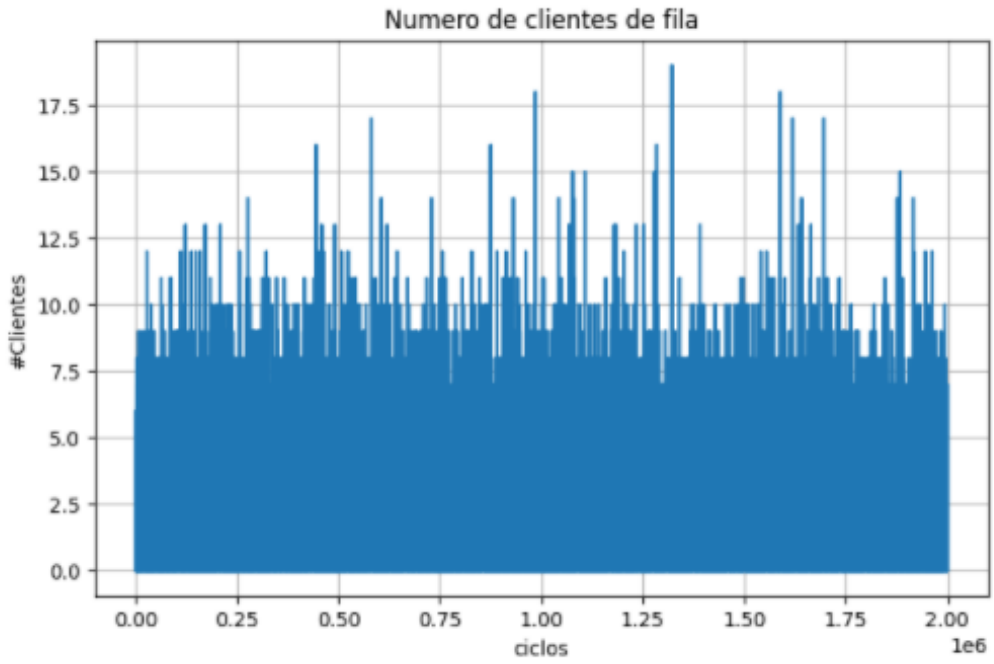

de vezes que sistema atinge estado zero
100
80
60
40
20
0
0
2000
4000
6000
8000
10000
RodadaS


densidade
tempo total clientes no sistema


densidade
tempo de serviço clientes no sistema


Gráfico da transiente - E[Nq]
E[Nq]
Rodadas


Gráfico da transiente - E[W]
E[W]
Rodadas


Numero de clientes de fila
#Clientes
ciclos
1e6

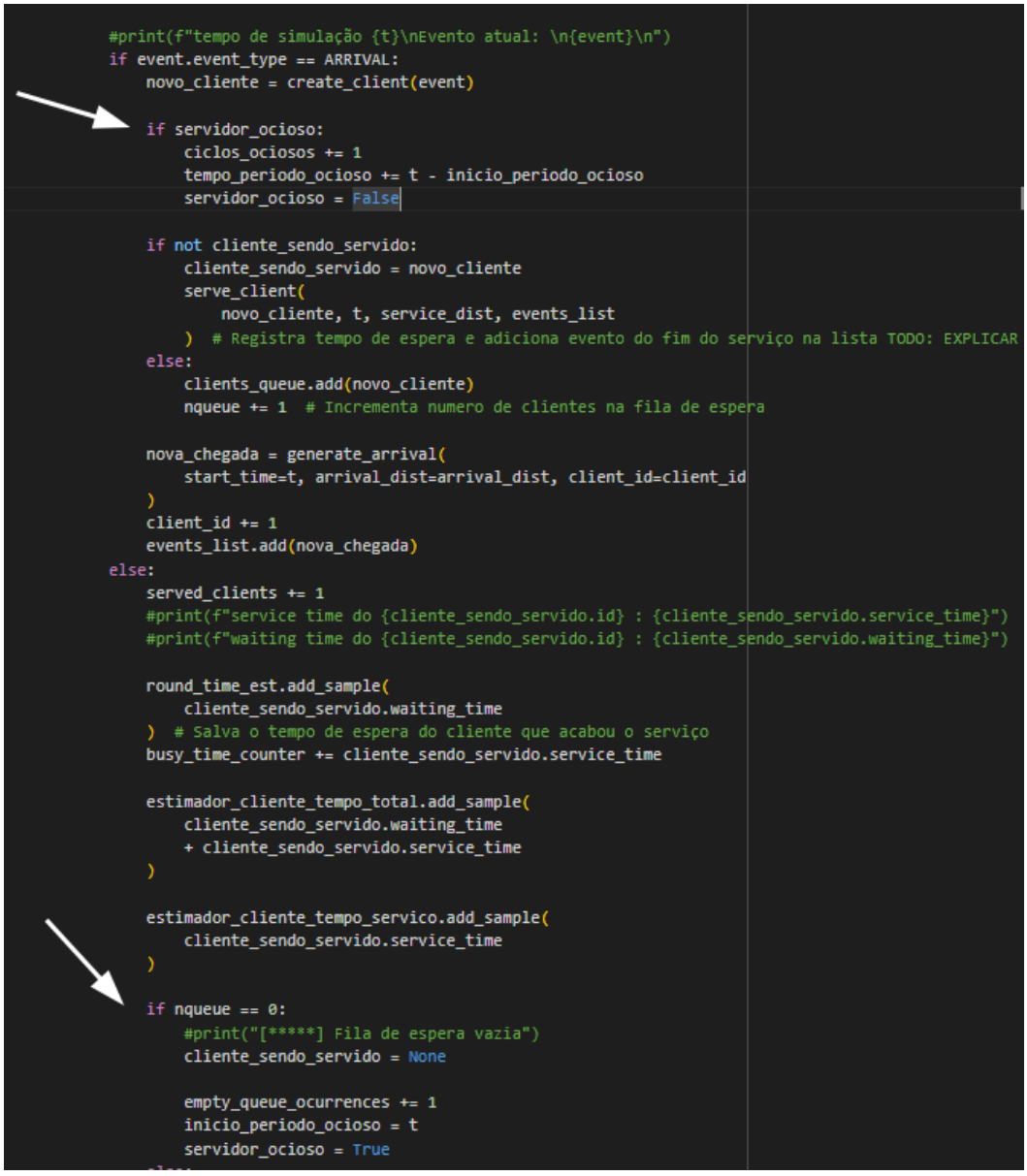
2.3 Fração de períodos ocupados que terminam

2.3.1 Heurística

A heurística adotada para registrar a fração (probabilidade) dos períodos ocupados que terminam foi:

```python
# metricas de nascimento e morte
servidor_ocioso = False
ciclos_ociosos = 0
tempo_periodo_ocioso = 0
inicio_periodo_ocioso = 0  # tempo da ultima morte
```

Essas variáveis servem para salvar se o servidor/sistema está ocioso, ou seja, sem ninguém na fila e sem ninguém sendo servido, e, para contar a quantidade de ciclos e tempo que o sistema fica ocioso. Com essas variáveis,



```python
#print(f"tempo de simulação {t}\nEvento atual: \n{event}\n")
if event.event_type == ARRIVAL:
    novo_cliente = create_client(event)

    if servidor_ocioso:
        ciclos_ociosos += 1
        tempo_periodo_ocioso += t - inicio_periodo_ocioso
        servidor_ocioso = False

    if not cliente_sendo_servido:
        cliente_sendo_servido = novo_cliente
        serve_client(
            novo_cliente, t, service_dist, events_list
        )  # Registra tempo de espera e adiciona evento do fim do serviço na lista TODO: EXPLICAR
    else:
        clients_queue.add(novo_cliente)
        nqueue += 1  # Incrementa numero de clientes na fila de espera

    nova_chegada = generate_arrival(
        start_time=t, arrival_dist=arrival_dist, client_id=client_id
    )
    client_id += 1
    events_list.add(nova_chegada)
else:
    served_clients += 1
    #print(f"service time do {cliente_sendo_servido.id} : {cliente_sendo_servido.service_time}")
    #print(f"waiting time do {cliente_sendo_servido.id} : {cliente_sendo_servido.waiting_time}")

    round_time_est.add_sample(
        cliente_sendo_servido.waiting_time
    )  # Salva o tempo de espera do cliente que acabou o serviço
    busy_time_counter += cliente_sendo_servido.service_time

    estimador_cliente_tempo_total.add_sample(
        cliente_sendo_servido.waiting_time
        + cliente_sendo_servido.service_time
    )

    estimador_cliente_tempo_servico.add_sample(
        cliente_sendo_servido.service_time
    )

    if nqueue == 0:
        #print("[*****] Fila de espera vazia")
        cliente_sendo_servido = None

        empty_queue_ocurrences += 1
        inicio_periodo_ocioso = t
        servidor_ocioso = True
```

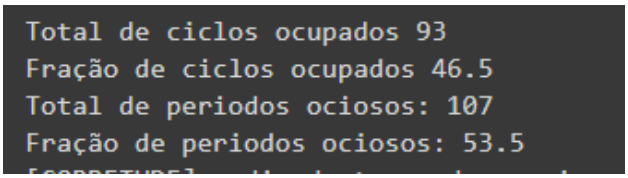
Assim que a fila e o servidor esvaziam (seta mais abaixo), marcamos o servidor como ocioso e registramos o tempo que ele começou a ficar ocioso.
Na próxima chegada(Arrival), então, caso o servidor esteja como ocioso, acrescemos a contagem de um ciclo ocioso e registramos o tempo total que ele ficou ocioso.

No final, teremos na variável ciclos_ociosos a Fração de Períodos Ociosos, como queremos a fração de períodos ocupados fazemos

Fração de Períodos Ocupados = 1 - Fração de Períodos Ociosos

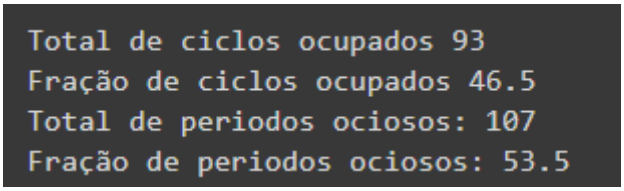
2.3.2 Resultados

Como resultado, no cenário 1 por exemplo

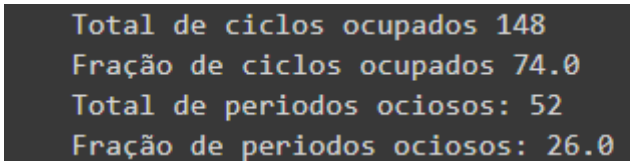


Onde se somarmos a fração de ciclos ocupados + fração de períodos ociosos = 100%.

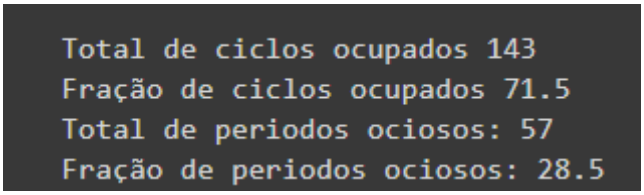
Para o cenário 2:



Para o cenário 3:



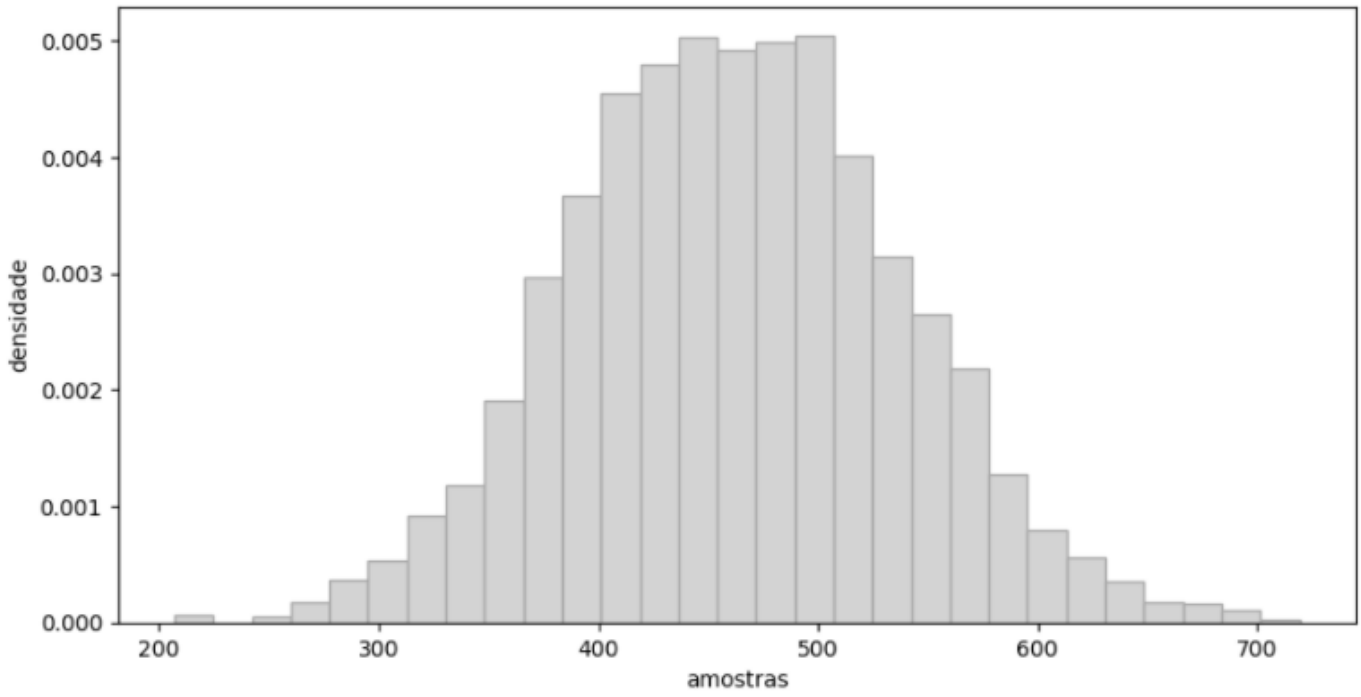
Para o cenário 4:



Notamos então, que no cenário 3 e 4, devido às taxas de lambda e mu, temos o sistema passa por muito mais ciclos ocupados do ciclos ociosos.

2.4 Dificuldades e Observações

Notamos que para um número grande de atendimentos no sistema, as amostras geradas pela exponencial passaram a apresentar uma distribuição normal.



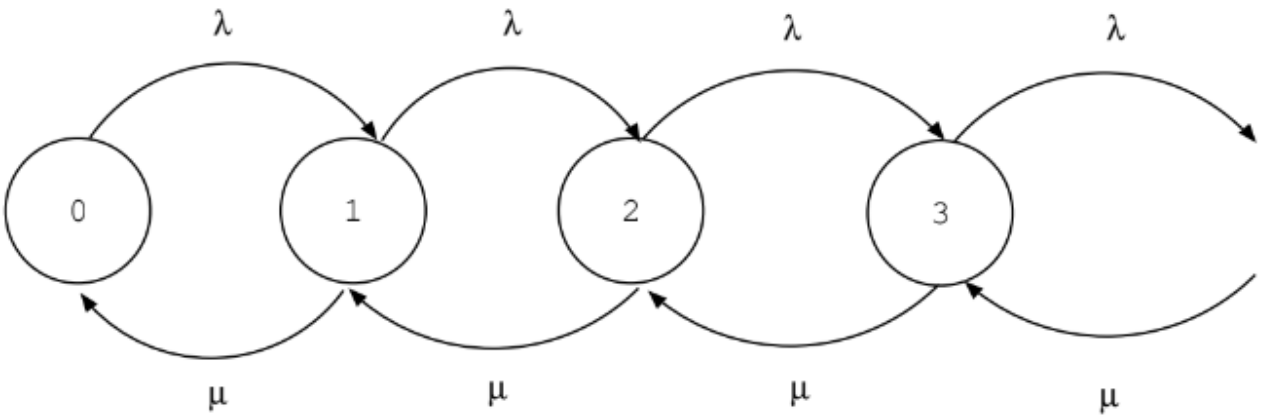
Distribuição de amostras para número mínimo de atendimentos igual à 10.000

Outro fato interessante foi o cálculo da média de número de clientes na fila durante as rodadas. Essa foi a única métrica que os resultados atuais divergiram dos resultados da entrega preliminar. Graças às verificações de corretude adicionadas, verificamos que os resultados atuais são os corretos, devido a um erro na coleção das amostras em relação ao tamanho da fila que constava na entrega preliminar. Daí temos um exemplo claro da importância da avaliação de corretude para fins de teste do simulador.

3. Ruína do Apostador

O problema estatístico da ruína do apostador consiste em, dado um apostador que começa em um estado qualquer *i*, ele tem probabilidade lambda λ de vencer uma aposta e aumentar seu ganho indo para o estado *i+1*, ou probabilidade μ de perder e retornar ao estado *i-1*.

Assim, o caso básico, onde o apostador pode ganhar infinitamente ou perder até zero pode ser modelado como a cadeia de markov abaixo, onde cada nó representa um estado do sistema e temos λ como a taxa de nascimento e μ como a taxa de morte.



[slides simulador página 14]

Essa mesma cadeia de Markov pode ser representada como uma fila M/M/1, nosso objeto de estudo.

[slides simulador página 14]

Trataremos a taxa de chegada na fila como λ (probabilidade de vitória / taxa de nascimento) e taxa de serviço μ (probabilidade de perder / taxa de morte). Dessa forma podemos modelar o problema como uma fila para rodar as simulações ou como uma cadeia de Markov para resolver analiticamente.

Utilizaremos a mesma fila implementada anteriormente, vale ressaltar, que como a fila começa com 1 cliente é o mesmo que dizer que o apostador começa no estado 1 (com 1 dinheiros).

3.1 Ruína do apostador: qual a probabilidade de a fila atingir o tamanho máximo K, e ocorrer um estouro, antes de ela esvaziar completamente?

Essa pergunta pode ser entendida como uma comparação entre uma fila também M/M/1/oo and M/M/1/K, então, podemos reescrevê-la como:

> *Dado uma fila M/M/1/K, queremos saber qual o máximo K que podemos atingir antes que ocorra um overflow, onde K representa a capacidade.*

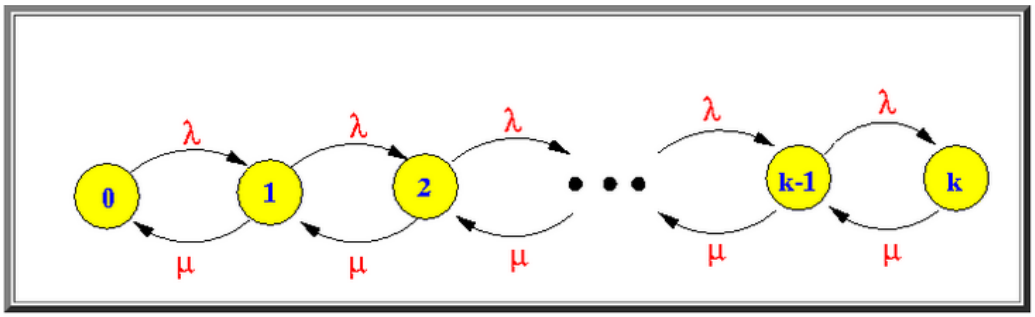
Definiremos então:

$$N := \text{tamanho atingido pela fila}$$

Então, no fim, queremos encontrar,

$$1 - P(N>=k)$$

que pode ser interpretado como a probabilidade total menos a probabilidade de N ser igual ou maior que K (condição de overflow)



[1]

Primeiramente, vamos encontrar as equações de equilíbrio

$$\pi_0 \lambda = \pi_1 \mu$$
$$\pi_1 \lambda = \pi_2 \mu$$
$$\pi_3 \lambda = \pi_4 \mu$$
$$\dots$$
$$\pi_k \lambda = \pi_{k+1} \mu$$

e, sabemos também que

$$\sum_{k=0}^{inf} \pi_K = 1$$

Como sabemos que $\rho = \frac{\lambda}{\mu}$, podemos, nas equações de equilíbrio, realizar essa substituição

$$\pi_k \lambda = \pi_{k+1} \mu \text{ , } k = 0,..,inf$$
$$\pi_k = \frac{\lambda}{\mu} \pi_{k+1}$$
$$\pi_k = \rho \, \pi_{k+1}$$

substituindo no somatório

$$\sum_{k=0}^{inf} \rho \, \pi_{k+1} = 1$$
$$x_1\rho^0 + x_2\rho^1 + \dots + x_k\rho^{k-1} = 1$$

com a fórmula da série geométrica, ficamos com

$$1 = \frac{\pi_0}{1-p}$$

$$\pi_0 = 1 - \rho$$

e, progredindo:

$$\pi_1 = \rho(1 - \rho)$$
$$\pi_2 = \rho^2(1 - \rho)$$
$$\dots$$
$$\pi_k = \rho^k(1 - \rho)$$

Assim, achamos a probabilidade de um sistema com buffer infinito estar no estado N, que pode ser dado por

$$Pr[N = k] = \rho^{k}(1 - \rho)$$

Como queremos achar a probabilidade da fila sofrer um buffer overflow, precisamos, então, achar a probabilidade de $Pr[N >= k]$, com K inclusivo, pois ao chegar em k já ocorre um estouro.

$$Pr[N >= k] = 1 - Pr[N < k]$$

$$Pr[N >= k] = 1 - (Pr[N = k - 1] + Pr[N = k - 2] + ... + Pr[N = 0])$$

$$Pr[N >= k] = 1 - [\rho^{k-1}(1 - \rho) + \rho^{k-2}(1 - \rho) + ... + \rho^{0}(1 - \rho)]$$

$$Pr[N >= k] = 1 - [(1 - \rho) * (\frac{1 - \rho^{k}}{1 - \rho})]$$

$$Pr[N >= k] = 1 - [(1 - \rho^{k})]$$

$$Pr[N >= k] = \rho^{k} \text{ ,com } \rho < 1$$

Como queremos a probabilidade antes dela esvaziar completamente precisamos que que a taxa de chegada (λ) seja menor do que a taxa de serviço (μ), logo ρ < 1

3.2 2.3 Fração de períodos ocupados que terminam

Segundo a análise da ruína do apostador, que você deverá reproduzir no relatório, qual a probabilidade de eventualmente o apostador atingir o estado 0, assumindo que ele começou no estado 1? Note que a probabilidade de incrementar o estado é de λ/(λ+μ) e a probabilidade de decrementar é de μ/(λ+μ)

Após a simulação dos Casos 3 e 4 na ruína do apostador como uma fila M/M/1, podemos perceber de fato que a taxa de chegada é maior do que a taxa de serviço, o sistema tende a acumular clientes (ou, no caso do jogador, acumular ganhos) ao longo do tempo.

3.2.1 Simulação do cenário 3 para a ruína do apostador

Caso 3

```python
def exp10():
  return generate_exp(1.0)

def exp105():
  return generate_exp(1.05)

LAMBDA = exp105

MU = exp10

QUEUE_MAX_SIZE = 4

N_RODADAS_GAMBLERS_RUIN = 40

valores = simulate2(N_RODADAS_GAMBLERS_RUIN, QUEUE_MAX_SIZE, LAMBDA, MU)


prob_time_ruin = valores["expected_time_to_ruin"]/valores["empty_queue_ocurrences"]
prob_steps_ruin = valores["expected_steps_to_ruin"]/valores["empty_queue_ocurrences"]
prob_ruin = valores["empty_queue_ocurrences"]/N_RODADAS_GAMBLERS_RUIN

prob_time_win = valores["expected_time_to_win"]/valores["full_queue_ocurrences"]
prob_steps_win = valores["expected_time_to_win"]/valores["full_queue_ocurrences"]
prob_win = valores["full_queue_ocurrences"]/N_RODADAS_GAMBLERS_RUIN

print(f"""Fração de ruinas: {prob_ruin}""")
print(f"""Fração tempo até a ruina: {prob_time_ruin:.5f}""")
print(f"Fração de passos até a ruina: {prob_steps_ruin:.5f}")

print(f"""Fração de fortuna: {prob_win}""")
print(f"""Fração tempo até a fortuna: {prob_time_win:.5f}""")
print(f"Fração de passos até a fortuna: {prob_steps_win:.5f}")
```

```
Fração de ruinas: 0.8
Fração tempo até a ruina: 2.02976
Fração de passos até a ruina: 2.87500
Fração de fortuna: 0.2
Fração tempo até a fortuna: 5.17654
Fração de passos até a fortuna: 5.17654
```

Começando com o estado 1 temos a probabilidade de 0.8 de chegar a ruína e de 0.2 de chegar a fortuna.

3.2.1 Simulação do cenário 4 para a ruína do apostador

Caso 4

```
def exp10():
  return generate_exp(1.0)

def exp110():
  return generate_exp(1.10)

LAMBDA = exp110

MU = exp10

QUEUE_MAX_SIZE = 4

N_RODADAS_GAMBLERS_RUIN = 40

valores = simulate2(N_RODADAS_GAMBLERS_RUIN, QUEUE_MAX_SIZE, LAMBDA, MU)


prob_time = valores["expected_time_to_ruin"]/valores["empty_queue_ocurrences"]
prob_steps = valores["expected_steps_to_ruin"]/valores["empty_queue_ocurrences"]
prob_ruin = valores["empty_queue_ocurrences"]/N_RODADAS_GAMBLERS_RUIN

prob_time_win = valores["expected_time_to_win"]/valores["full_queue_ocurrences"]
prob_steps_win = valores["expected_time_to_win"]/valores["full_queue_ocurrences"]
prob_win = valores["full_queue_ocurrences"]/N_RODADAS_GAMBLERS_RUIN

print(f"""Fração de ruinas: {prob_ruin}""")
print(f"""Fração tempo até a ruina: {prob_time:.5f}""")
print(f"Fração de passos até a ruina: {prob_steps:.5f}")

print(f"""Fração de fortuna: {prob_win}""")
print(f"""Fração tempo até a fortuna: {prob_time_win:.5f}""")
print(f"Fração de passos até a fortuna: {prob_steps_win:.5f}")
```

```
Fração de ruinas: 0.775
Fração tempo até a ruina: 3.16598
Fração de passos até a ruina: 4.45161
Fração de fortuna: 0.225
Fração tempo até a fortuna: 3.65719
Fração de passos até a fortuna: 3.65719
```

Começando com o estado 1 temos a probabilidade de 0.775 de chegar a ruína e de 0.225 de chegar a fortuna.

3.3 Solução analítica

Comparemos os resultados da solução analítica obtida com os resultados da *seção 3.2* referentes a simulação da ruína do apostador nos cenários 3 e 4

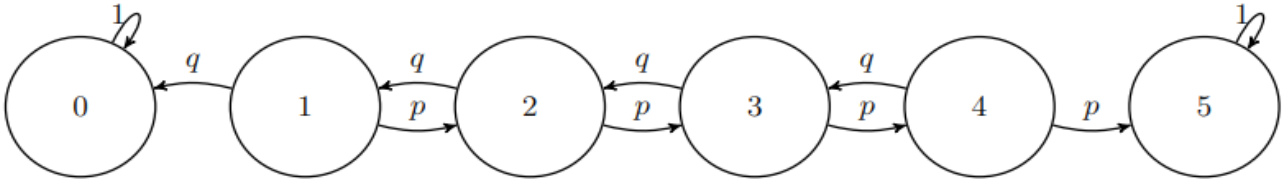


Figura 4: Diagrama para o caso da ruína do apostador finita. O estado inicial é sempre o estado 1.

[Imagem retirada do enunciado do trabalho]

Para encontrarmos a solução analítica, modelamos a cadeia de markov como na imagem acima onde a probabilidade de incrementar o estado é $\frac{\lambda}{(\lambda+\mu)}$ e a probabilidade de decrementar é $\frac{\mu}{(\lambda+\mu)}$

Podemos notar também as características que essa cadeia absorvente, aperíodica e irredutível.

3.3.1 Cenário 3

Para o cenário 3, temos os valores definidos como

$$\rho = 1.05, \lambda = 1.05, \mu = 1$$

Assim, nossos valores de probabilidade são:

$$\text{incrementar: } p = \frac{\lambda}{(\lambda+\mu)} = \frac{1.05}{1.05+1} = 0.5121951219512195$$

$$\text{decrementar: } q = \frac{\mu}{(\lambda+\mu)} = \frac{1}{1.05+1} = 0.48780487804878053$$

Assim, ficamos com a seguinte matriz de probabilidade

$$P_{ij} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ q & 0 & p & 0 & 0 & 0 \\ 0 & q & 0 & p & 0 & 0 \\ 0 & 0 & q & 0 & p & 0 \\ 0 & 0 & 0 & q & 0 & p \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Substituindo p e q

$$Pij = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0.4878 & 0 & 0.5122 & 0 & 0 & 0 \\ 0 & 0.4878 & 0 & 0.5122 & 0 & 0 \\ 0 & 0 & 0.4878 & 0 & 0.5122 & 0 \\ 0 & 0 & 0 & 0.4878 & 0 & 0.5122 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Pelas características da matriz podemos resolver pelo método da potência para resolver.

$$P^n = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0.780024 & 0 & 0 & 0 & 0 & 0.219976 \\ 0.57052305 & 0 & 0 & 0 & 0 & 0.42947695 \\ 0.37099834 & 0 & 0 & 0 & 0 & 0.62900166 \\ 0.1809748 & 0 & 0 & 0 & 0 & 0.8190252 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz P elevada a N

Como sempre começamos no estado 1 precisamos ver os valores gerados na linha 1, para chegar na ruína queremos a posição $P_{1,0}$ e para chegar no overflow queremos a posição $P_{1,5}$

```
Fração de ruinas: 0.8
Fração tempo até a ruina: 2.02976
Fração de passos até a ruina: 2.87500
Fração de fortuna: 0.2
Fração tempo até a fortuna: 5.17654
Fração de passos até a fortuna: 5.17654
```

Probabilidades encontradas na simulação

Comparando com os resultados obtidos através da simulação vemos que $P_{1,0}$= 0.780024 e os dados da simulação vemos que a ruína é 0.8 .

Comparando com os resultados obtidos através da simulação vemos que $P_{1,5}$= 0.219976 e os dados da simulação vemos que a fortuna/overflow é 0.2 .

3.3.2 Cenário 4

Para o cenário 4, temos os valores definidos como

$$\rho = 1.10,\ \lambda = 1.10,\ \mu = 1$$

Assim, nossos valores de probabilidade são:

$$\text{incrementar: } p = \frac{\lambda}{(\lambda+\mu)} = \frac{1.10}{1.10+1} = 0.5238095238095238$$

$$\text{decrementar: } q = \frac{\mu}{(\lambda+\mu)} = \frac{1}{1.10+1} = 0.47619047619047616$$

Assim, ficamos com a seguinte matriz de probabilidade

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0.47619047619047616 & 0 & 0.5238095238095238 & 0 & 0 & 0 \\ 0 & 0.47619047619047616 & 0 & 0.5238095238095238 & 0 & 0 \\ 0 & 0 & 0.47619047619047616 & 0 & 0.5238095238095238 & 0 \\ 0 & 0 & 0 & 0.47619047619047616 & 0 & 0.5238095238095238 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Pelo método da potência

$$P^n = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.76018411 & 0.0 & 0.0 & 0.0 & 0.0 & 0.23981589 \\ 0.54216966 & 0.0 & 0.0 & 0.0 & 0.0 & 0.45783034 \\ 0.34397471 & 0.0 & 0.0 & 0.0 & 0.0 & 0.65602529 \\ 0.16379748 & 0.0 & 0.0 & 0.0 & 0.0 & 0.83620252 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Comparando com os resultados obtidos através da simulação vemos que $P_{1,0}$= 0.780024 e os dados da simulação vemos que a ruína é 0.775

Comparando com os resultados obtidos através da simulação vemos que $P_{1,5}$= 0.219976 e os dados da simulação vemos que a ruína é 0.225

```
Fração de ruinas: 0.775
Fração tempo até a ruina: 3.16598
Fração de passos até a ruina: 4.45161
Fração de fortuna: 0.225
Fração tempo até a fortuna: 3.65719
Fração de passos até a fortuna: 3.65719
```

Probabilidades encontradas na simulação

3.3.3 Código utilizado para a solução analítica

```
1
2 def exp_gamblers(lambda, mu):
3     p = lambda/(lambda+mu)
4     q = mu/(lambda+mu)
5
6     Pij = np.array([[
7         #  0  1  2  3  4  5
8     [1, 0, 0, 0, 0, 0],
9     [q, 0, p, 0, 0, 0],
10    [0, q, 0, p, 0, 0],
11    [0, 0, q, 0, p, 0],
12    [0, 0, 0, q, 0, p],
13    [0, 0, 0, 0, 0, 1],
14    ]])
15
16
17    exponent = 10000000
18    np.set_printoptions(suppress=True)
19    result = np.linalg.matrix_power(Pij, exponent)
20    print(result)
21
22 # cenario 3
23 exp_gamblers(lambda=1.05, mu=1)
24 # cenario 4
25 exp_gamblers(lambda=1.10, mu=1)
```

O código utilizado para realizar a potenciação foi

4. Considerações finais

4.1 Dúvidas e Dificuldades gerais

$$\rho = \begin{cases} \lambda/\mu, & \text{se } \lambda < \mu \\ 1, & \text{caso contrário} \end{cases}$$

página 14 apostila extra

- fórmula (1.8) se o p (ro) é 1 no caso onde lambda < mu, porque utilizamos o p (ro) como 1.05 e não como 1 ?

- Observamos também que aumentando o número mínimo de clientes a serem atendidos a amostra gerada pela exponencial começa a se aproximar de uma distribuição normal

5. Fontes

[1] https://www.cs.emory.edu/~cheung/Courses/558/Syllabus/00/queueing/queueing.html#:~:text=Finite%20buffer%20capacity%3A%20M/M/1/k
https://ir.canterbury.ac.nz/server/api/core/bitstreams/67e5375c-f396-4231-8dd6-991d5851a44b/content
https://www.cs.emory.edu/~cheung/Courses/558/Syllabus/00/queueing/queueing.html#:~:text=Finite%20buffer%20capacity%3A%20M/M/1/k

https://medium.com/mlearning-ai/central-limit-theorem-lets-learn-with-an-example-using-python-4801b8d1c6b5#:~:text=Central%20Limit%20Theorem%20(CLT)%20states,follow%20Gaussian%20(Normal)%20Distribution.

[slides simulador pagina 14]https://classroom.google.com/u/4/c/NjE4MzAwMDA5NjQz