UFRJ - Universidade Federal do Rio de Janeiro

Professor: Daniel Sadoc
Alunos:
Gabriel Martins Machado Christo - 117217732
Yago Alves da Costa - 115212477
Yuri Medeiros da Silva - 117061898

Segundo Trabalho de AD: Epidemias e Período Ocupado

Link para o colab: https://colab.research.google.com/drive/1iI-guYyibVbEU3iDNdL1NIm3y2PeDlAx?usp=sharing

5.1 Epidemias

5.1.1 Da relação do período ocupado das filas com epidemias

O período ocupado de uma fila pode ser pensado como o intervalo no qual temos chegadas contínuas de clientes na fila, ou seja, o período em que a fila não está vazia e o servidor está ocupado. Já o período em que a fila está sem receber ninguém e sem ninguém ser atendido pode ser chamado de período ocioso.

Entender o período ocupado da fila é necessário para entender a capacidade do sistema, a eficiência do serviço, e o tempo de espera dos cliente, eles são indicadores da demanda no sistema em relação à sua capacidade de processamento.

Como exemplo de uma métrica que pode ser entendida, e como veremos nos resultados das simulações, um serviço que possui poucos períodos ocupados tende a ter uma taxa de serviço(μ) maior ou igual a taxa de chegada (λ)

Branching process, epidemias, é o conceito de modelagem utilizado para descrever propagação de uma epidemia, ou, como foi primeiramente pensado, entender a propagação de um sobrenome.

A relação de filas com epidemias, pode ser entendida:

1. O período ocupado representa o tempo em que a fila está ativa e crescendo, o que se assemelha, com o crescimento da infecção de pessoas em um branching process.
2. Uma fila em um período ocioso, ou seja, quando não há mais nada para ser servido e ninguém na fila, pode ser comparada a quando a “fila” de pessoas infectadas na epidemia se extingue, ou seja, não temos mais transmissões da infecção.

5.1.1.2 Nossa abordagem

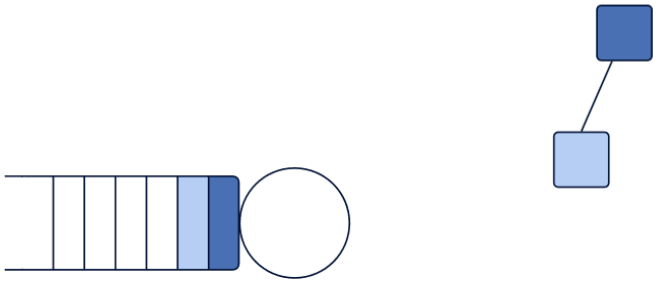
Nosso grupo conseguiu entender e interpretar melhor o trabalho com a utilização de grafos, seguindo, à risca, o processo de ramificação.
Obs: Apesar de não tão performático, foi muito útil para o entendimento.

Na nossa modelagem e heurística, que será melhor detalhada no item **5.1.3**, interpretamos o início de cada período ocioso como o fim da criação de uma árvore.

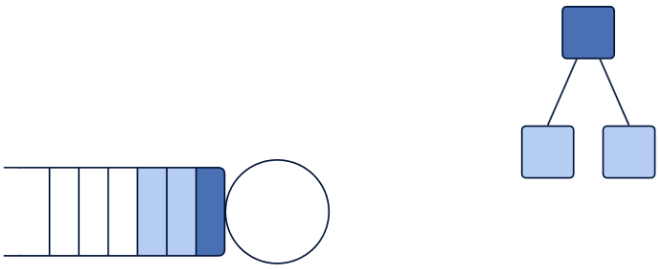
Exemplo:

- Começamos sempre com 1 filho.

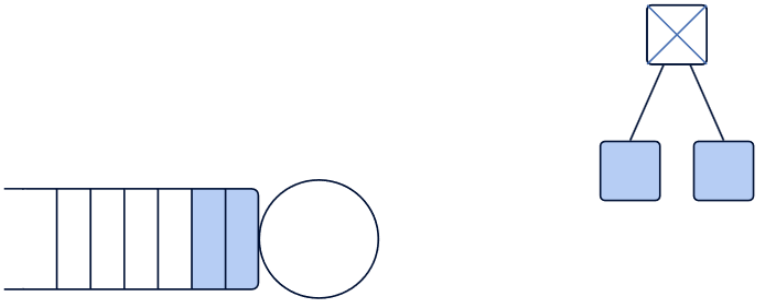
Dado que nosso sistema começa com 1 cliente, nossa fila inicial pode ser representada por (imagem acima), e esse primeiro cliente sendo a raiz da árvore e nosso primeiro nó pai.



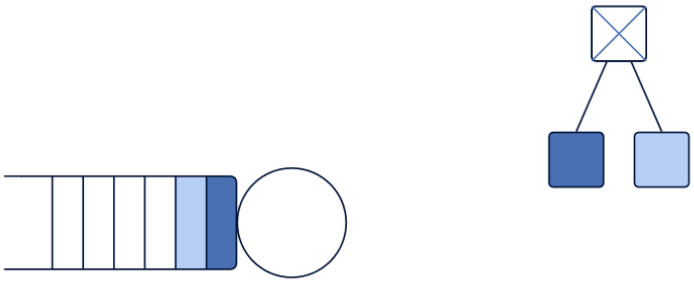
Todas as chegadas que ocorrem enquanto nosso pai está sendo atendida serão filhos desse primeiro pai.



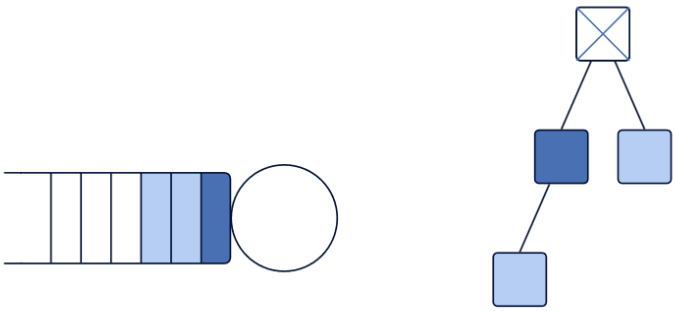
Chegada de mais um cliente na fila



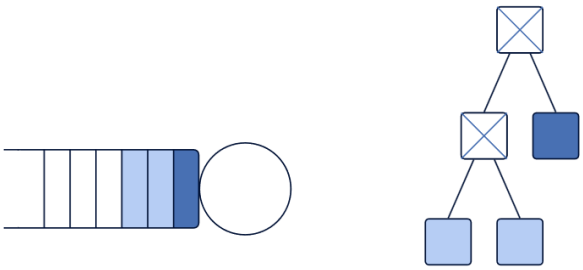
A partir do momento que o primeiro nó é servido, ele deixa de ser o nosso pai atual e o próximo nó pai é o próximo serviço que estará sendo servido/atendido..



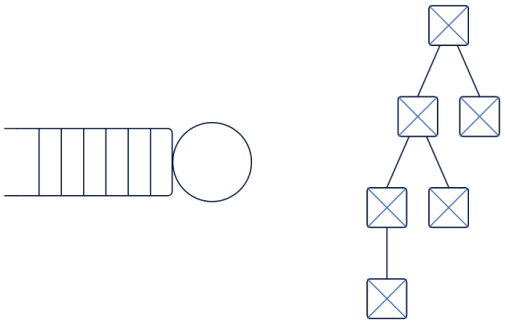
Novo pai é escolhida



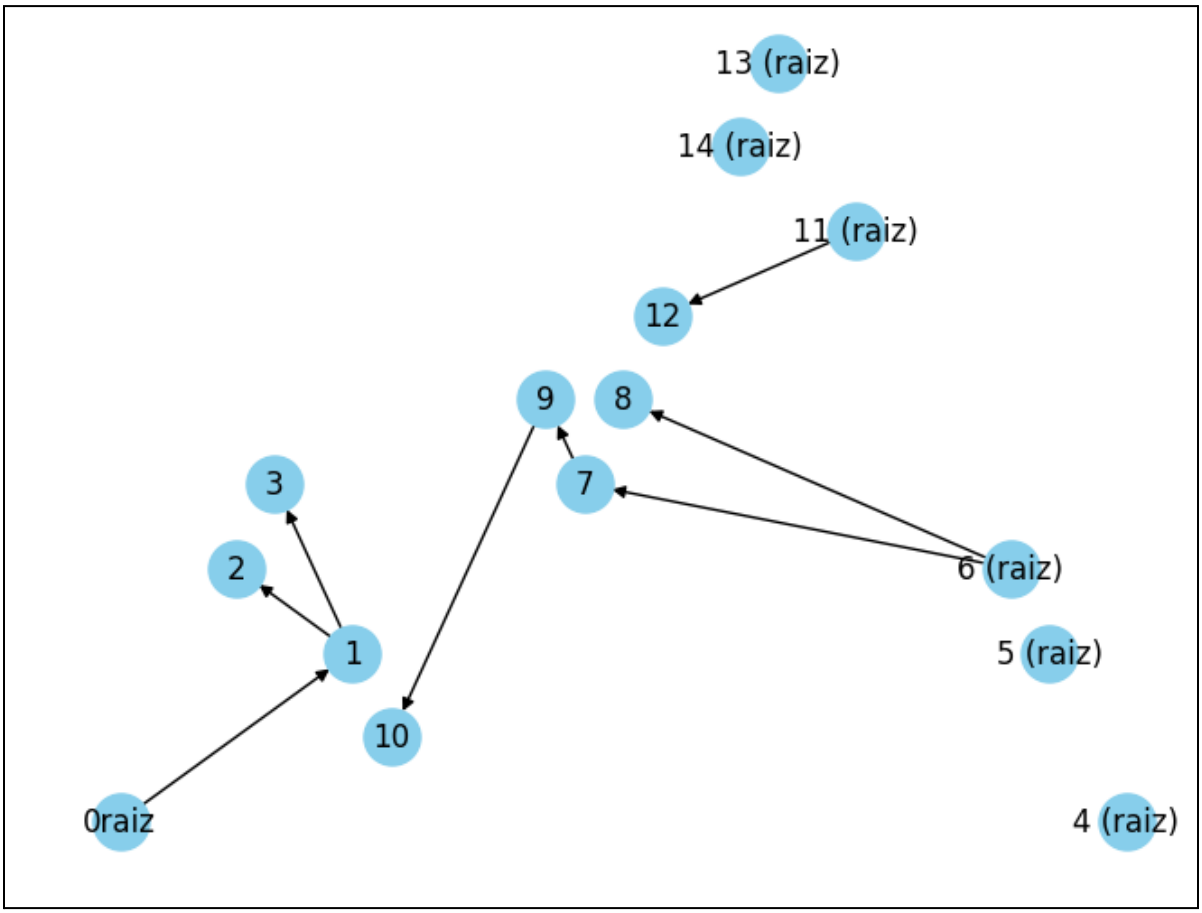
A partir do momento que chega um novo cliente (nova infecção), ele será filho do novo pai



Depois que o 2 é servido, o próximo nó pai será o 3



Esse processo continua até a árvore ser toda consumida (todos os clientes atendidos) — ou seja, uma infecção ser extinta. Ou quando o número de clientes mínimos a ser atendido é atingido, ou ainda quando tivermos uma árvore infinita (um caso supercrítico)



Exemplo de árvore gerada pelo período ocupado da fila do simulador durante uma rodada com um total de 13 clientes atendidos.
Podemos perceber que temos diversas árvores criadas, ou seja, entre a criação de uma árvore e outra tivemos um período ocioso (uma infecção terminou) e depois começou uma nova, até infectar a quantidade de clientes que desejávamos(13).

5.1.2 Probabilidade de extinção

O teorema que justifica a equação s= G(s) pode ser encontrado no teorema 4.2 do livro (**Robert P. Dobrow - Introduction to stochastic processes with R-John Wiley & Sons (2016)**) e sua prova encontra-se no fim do capítulo 4, página 173.
A forma através da qual obtivemos os resultados e o passo a passo foi obtido na **seção 6** deste relatório.
Resultados encontrados:

cenário	Probabilidade de extinção para caso **exponencial**	Probabilidade de extinção para caso **determinístico**
caso 1 (λ: 1 e μ: 2)	1	1.0
caso 2 (λ: 2 e μ: 4)	1	1.0
caso 3 (λ: 1.05 e μ: 1)	0.952380	0.90629
caso 4 (λ: 1.10 e μ: 1)	0.909090	0.82386

5.1.3 Heurística de árvores finitas e infinitas

Como o computador não consegue trabalhar propriamente com estruturas infinitas, definimos uma heurística que entende uma árvore infinita a partir da quantidade de nós da árvore.
Para isso realizamos alguns testes até achar a melhor heurística disponível.

5.1.3.1 Abordagem Dinâmica (não utilizada)

Nessa abordagem, começamos por considerar uma árvore infinita caso o tamanho dela seja maior que o mínimo de clientes. Como, em cada rodada podemos ter no máximo uma árvore infinita, a cada iteração subtraímos o número de clientes já atendidos.

```
limite = MIN_CLIENTES
for subgraph in todas_as_arvores_da_rodada:
  #print(subgraph)
  if subgraph.number_of_nodes() > (limite):
      #print("É INFINITA")
      arvs_infs += 1
  else:
      #print("NÃO É INFINITA")
      arvs_fin += 1
  arvores_inf_contador.append(arvs_infs)
  arvores_fin_contador.append(arvs_fin)

  # uma vez que a arvore ja foi contada, reduzimos o limite
  # para considerá-la infinita
  limite -=  subgraph.number_of_nodes()

fracao_periodos_ocupados = arvores_fin_contador.mean() /
(arvores_fin_contador.mean() + arvores_inf_contador.mean())
```

Contudo, essa abordagem apresentou resultados inconsistentes, e esbarramos com o problema de, caso tenhamos muitas árvores na floresta, pode ser que chegue um ponto que nosso limite seja 1 e toda árvore seja considerada infinita.

5.1.3.2 Abordagem Quantidade de nos

Aqui, continuamos com a ideia de limitar pelo #qtde de nós, a diferença é que deixamos um valor fixo para o limite

```
limite = MIN_CLIENTES * 0.9
for subgraph in todas_as_arvores_da_rodada:
  #print(subgraph)
  if subgraph.number_of_nodes() > (limite):
      #print("É INFINITA")
      arvs_infs += 1
  else:
      #print("NÃO É INFINITA")
      arvs_fin += 1
  arvores_inf_contador.append(arvs_infs)
  arvores_fin_contador.append(arvs_fin)

fracao_periodos_ocupados = arvores_fin_contador.mean() /
(arvores_fin_contador.mean() + arvores_inf_contador.mean())
```

5.1.3.3 Abordagem com algoritmo de decisão de monte-carlo

Analisando o algoritmo anterior, vimos que o limite definido de forma fixa pode ser errôneo em alguns casos, então, como achar o melhor limite para definirmos uma árvore como infinita ?

Utilizamos o algoritmo de simulações e decisão de monte-carlo - o algoritmo explora todos os possíveis estados de um problema (que no nosso caso são os possíveis limites para definição de uma árvore infinita) e, depois das simulações, temos o problema de decisão “qual o melhor limite”, e, para isso, fazemos utilização do resultado que calculamos analiticamente.

Algumas observações precisam ser ressaltadas:

- O algoritmo sempre termina, por mais que possa demorar pra muitas simulações (testamos com até 100_000 simulações).
- O algoritmo pode ser melhorado tornando a escolha da próxima v.a *aleatória*, utilizamos a forma sequencial pois inicialmente pensamos apenas em mapear o conjunto espaço.

Durante o trabalho realizamos diversos testes e análises, notamos que essa abordagem por monte-carlo se sobressai quando tratamos de casos deterministicos, já a abordagem **5.1.3.2** apresenta melhores resultados para os casos exponenciais.

```
# inicio da nossa simulação
start = 0.65
end = 1.1
increment = 0.03

# guarda a média de arvores finitas daquela simulação
grafico = []
# times guarda o limite utilizado para uma determinada
# media de arvores finitas daquela simulação
# limite é a porcentagem que será aplicada ao MIN_CLIENTES
times = []

current = start
while current <= end:
    # a cada simulação resetamos nossos contadores
    arvs_infs = 0
    arvs_fin = 0

    for subgraph in todas_as_arvores:
        if subgraph.number_of_nodes() > (MIN_CLIENTES * current):
            arvs_infs += 1
        else:
            arvs_fin += 1

        arvores_inf_contador.append(arvs_infs)
        arvores_fin_contador.append(arvs_fin)

    times.append(current)
    grafico.append(arvs_fin / (arvs_fin + arvs_infs))

    current += increment
```

5.1.3.4 Testes combinando as duas heurísticas anteriores (Algorítmo de backtracking)

Utilizando a heurística de monte-carlo anterior [**5.1.3.3**] foi possível utilizar da técnica de backtracking para feedar/alimentar a heurística simples, e assim obter sempre a probabilidade esperada analiticamente.

```python
#
def find_nearest(array, value):
    print(f"times: {array} analitico: {value}")
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return array[idx], idx


def monte_carlo_decision_of_coeficient(cenario, tipo):

    start = 0.3
    end = 1.2
    increment = 0.03
    current = start
    times = []
    grafico = []

    while current <= end:
        arvs_infs = 0
        arvs_fin = 0

        for subgraph in todas_as_arvores:
            if len(subgraph.vs) > (MIN_CLIENTES * current):
                arvs_infs += 1
            else:
                arvs_fin += 1

        times.append(current)
        grafico.append(arvs_fin / (arvs_fin + arvs_infs))
        current += increment

    resultado_analitico = get_resultado_analitico(cenario, tipo)

    grafico_rounded = [round(item, 5) for item in grafico]
    # print(round(resultado_analitico,5))
    valor, idx = find_nearest(grafico_rounded, value=round(resultado_analitico,5))
    # coeficiente para ser utilizado no calculo da heuristica simples
    # com MIN_CLIENTES * coeficiente
    return times[idx]
```

```python
#heuristica simples de calculo de arvores finitas com backtracking
# para alimentar o limite do tamanho do upper_bound da arvore
def simple_heuristic_frac_finite_trees(limite = MIN_CLIENTES * 0.9 ):
    arvs_infs = 0
    arvs_fin = 0
    frac_arv_fin_por_rodada = []

    for subgraph in todas_as_arvores:
        #if subgraph.number_of_nodes() > (MIN_CLIENTES * 0.9):
        if len(subgraph.vs) > (limite):
            arvs_infs += 1
        else:
            arvs_fin += 1
        frac_arv_fin_por_rodada.append(arvs_fin / (arvs_fin + arvs_infs))

    resultado = arvs_fin / (arvs_fin + arvs_infs)

    # calculando intervalo de confianca
    amostras = pd.Series(frac_arv_fin_por_rodada)
    media = amostras.mean()
    z = T_PERCENTILE # aumenta precisao
    s = amostras.std()
    n = N_RODADAS
    intervalo = (resultado - z*(s/sqrt(n)), resultado + z*(s/sqrt(n)))

    # print(f"resultado: {resultado}")
    # print(f"resultado-ic: {intervalo}\n")
    return resultado, intervalo
```

5.1.3.5 Conclusão sobre as heurísticas

Com exceção da primeira, as outras 3 heurísticas (simples, simulações de monte-carlo, simples com backtracking) são bem similares, elas foram surgindo como uma evolução greedy da heurística simples.

Na heurística simples nós setamos um upper-bound para o tamanho da árvore com base no mínimo de clientes que temos que atender, 90% dos clientes atendidos, contudo, isso pode não ser um comportamento muito assertivo para todos os cenários testados, podemos ter casos onde esse limiar funciona para um cenário e não para outro cenário.

Com base nisso, tentamos melhorar o algoritmo para ter um upper-bound definido dinamicamente, assim rodamos as simulações de monte-carlo para achar um coeficiente de porcentagem ideal (MIN_CLIENTES * coeficiente) para a heurística simples em cada cenário.

Resultados finais:

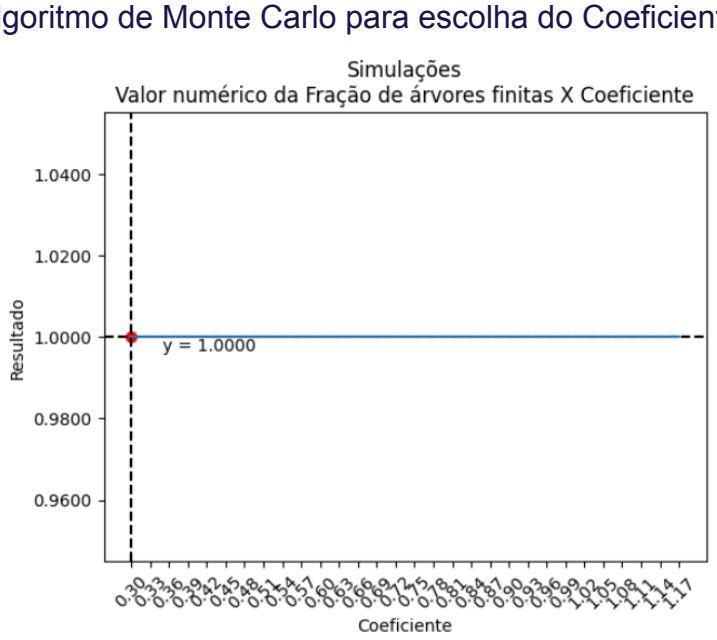
As 3 heurísticas dão resultados próximos dentro do intervalo de confiança esperado pelo valor analítico, contudo, a abordagem com backtracking pode apresentar, em alguns casos, resultados mais próximos. Apresentaremos a solução com ambas as abordagens

5.1.4 Comparativo resultados numéricos e analíticos

Observação: Tanto no cenário 1 quanto no cenário 2, não seria necessário utilizar a heurística de backtracking ou a de monte-carlo, contudo, apontamos os resultados *“for the sake of completeness”.*

Os resultados aqui apresentados foram rodados com 100 rodadas e 2500 clientes, também rodamos com 3200 rodadas e 200 clientes e encontramos valores bem próximos com os que aqui estão apresentados, diferença apenas nas últimas casas decimais.

Caso 1 - Cenário 1 (exponencial)



Resultado analtico	Resultado com a heurística simples	Resultado da heurística simples + Backtracking
1	1.0	1.0

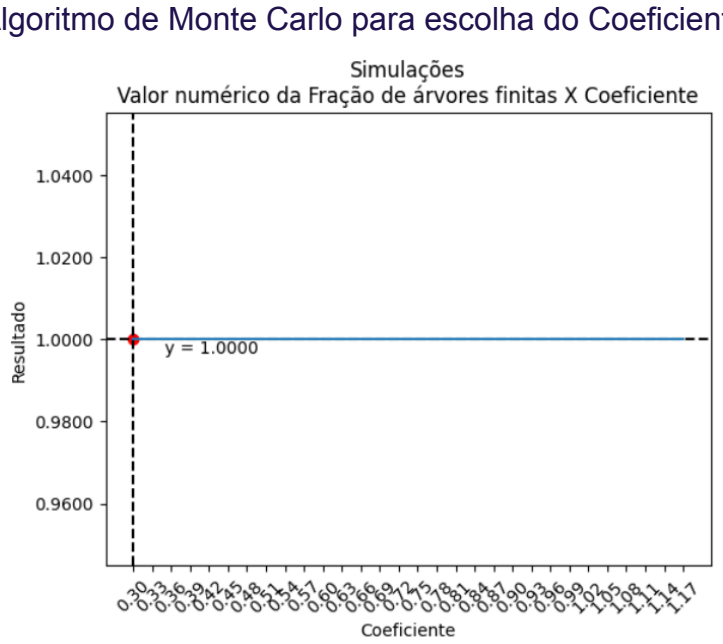
```
Valor analiticamente esperado de arvores finitas 1
Heurística simples:
Fração de arvores finitas 1.0
Intervalo de confianca (1.0, 1.0)

Algoritmo com Backtracking:
Fração de arvores finitas 1.0
intervalo de confianca: (1.0, 1.0)
```

Verificação de Corretude para Média da distribuição Offspring (calculado com a PDF do 5.2)

mu = 0.5123900879296562 (caso subcrítico, logo temos 100% de períodos ocupados finitos - probabilidade de termos árvores finitas é 1)

Compatível com resultados da tabela.

Caso 2 - Cenário 2 (exponencial)



Resultado analtico	Resultado com a heurística simples	Resultado da heurística simples + Backtracking
1	1.0	1.0

```
Valor analiticamente esperado de arvores finitas 1
Heurística simples:
Fração de arvores finitas 1.0

Algoritmo com Backtracking:
Fração de arvores finitas 1.0
intervalo de confianca: (1.0, 1.0)
```

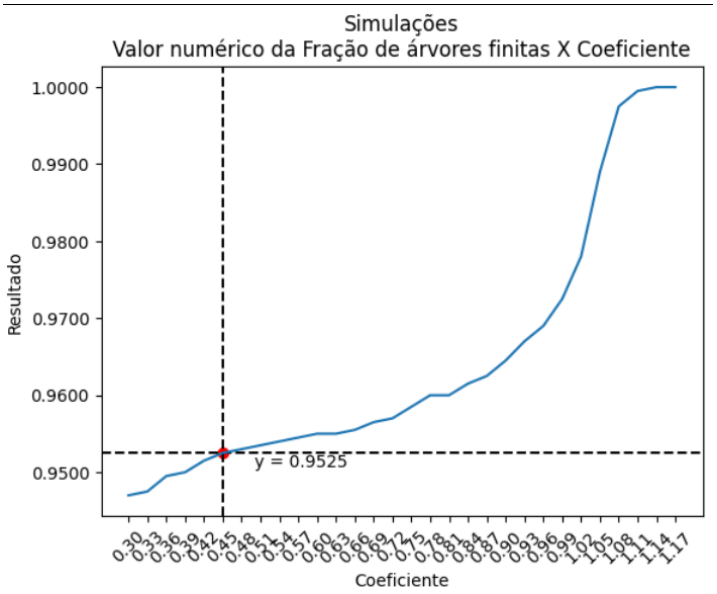
Verificação de Corretude para Média da distribuição Offspring (calculado com a PDF do 5.2)

mu = 0.49820071971211516 (caso subcrítico, logo temos 100% de períodos ocupados finitos - probabilidade de termos árvores finitas é 1)

Compatível com resultados da tabela.

Caso 3 - Cenário 3 (exponencial)

Algoritmo de Monte Carlo para escolha do Coeficiente



Resultado analtico	Resultado com a heurística simples	Resultado da heurística simples + Backtracking
0.952380	0.96448224	0.9524762

```
Valor analiticamente esperado de arvores finitas 0.9523809523809523
Heurística simples:
Fração de arvores finitas 0.9644822411205602
intervalo de confianca: (0.9520258511803555, 0.965538631060765)

Algoritmo com Backtracking:
Fração de arvores finitas 0.9524762381190596
intervalo de confianca: (0.951102891150711, 0.9538495850874081)
```

Nesse caso podemos ver como a heurística com o backtracking dá um valor mais ajustado para o resultado.

Verificação de Corretude para Média da distribuição Offspring (calculado com a PDF do 5.2)

mu = 0.9972495088408643 (caso crítico, logo podemos ter períodos ocupados finitos e infinitos)

Compatível com resultados

Caso 4 - Cenário 4 (exponencial)

Algoritmo de Monte Carlo para escolha do Coeficiente

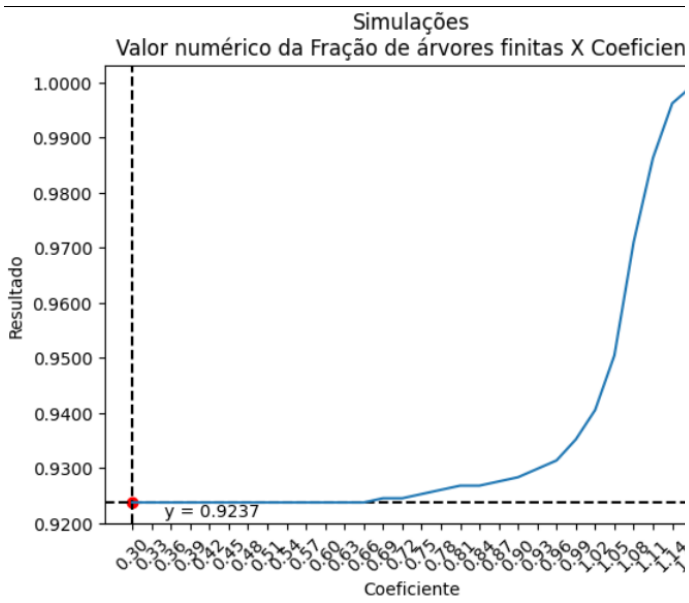


Imagem 1

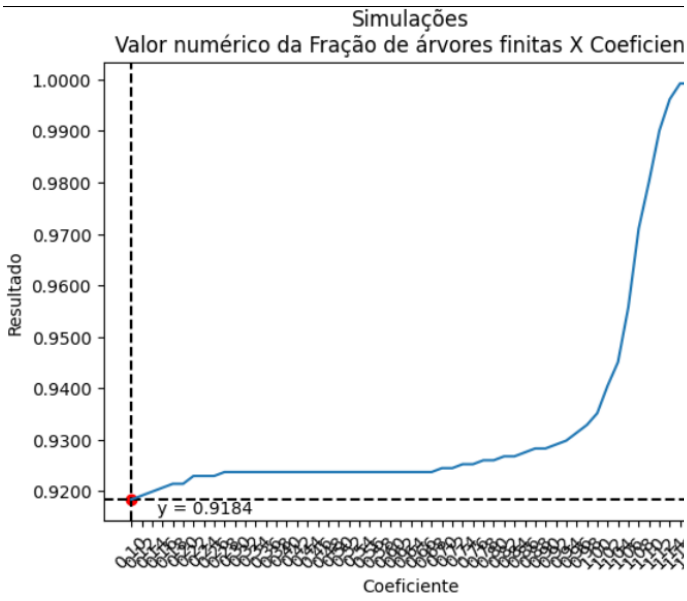


Imagem 2

Observação empírica:

Na primeira imagem temos um coeficiente de 30% do número de clientes atendidos como sendo o ideal, já na segunda imagem temos de 10%.

Nesse caso, como nossa taxa de chegadas é maior que a taxa de serviço, vemos que para ter um menor número de árvores finitas precisamos ter um coeficiente ainda menor, já que categoricamente estaremos quase sempre recebendo mais chegadas do que podemos atender($\lambda > \mu$).

Resultado analtico	Resultado com a heurística simples	Resultado da heurística simples + Backtracking
0.90909	0.92829	0.91838

```
Valor analiticamente esperado de arvores finitas 0.9090909090909091
Heurística simples:
Fração de arvores finitas 0.9282990083905416
intervalo de confianca: (0.9263117900633941, 0.930286226717689)
Algoritmo com Backtracking:
Fração de arvores finitas 0.9183829138062548
intervalo de confianca: (0.9064280202769023, 0.9203378073356073)
Valor analiticamente esperado de arvores finitas 0.9090909090909091
```

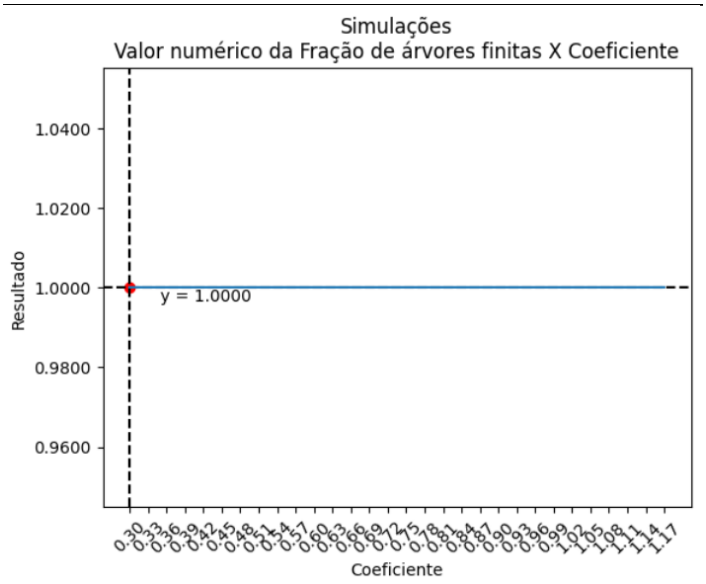
Verificação de Corretude para Média da distribuição Offspring (calculado com a PDF do 5.2)

mu = 0.9996233521657252 (caso crítico, logo podemos ter períodos ocupados finitos e infinitos)

Compatível com resultados da tabela

Caso 5 - Cenário 1 (determinístico)

Algoritmo de Monte Carlo para escolha do Coeficiente



Resultado analtico	Resultado com a heurística simples	Resultado da heurística simples + Backtracking
1	1.0	1.0

```
Valor analiticamente esperado de arvores finitas 1.0
Heurística simples:
Fração de arvores finitas 1.0
Intervalo de confianca (1.0, 1.0)

Algoritmo com Backtracking:
Fração de arvores finitas 1.0
intervalo de confianca: (1.0, 1.0)
```

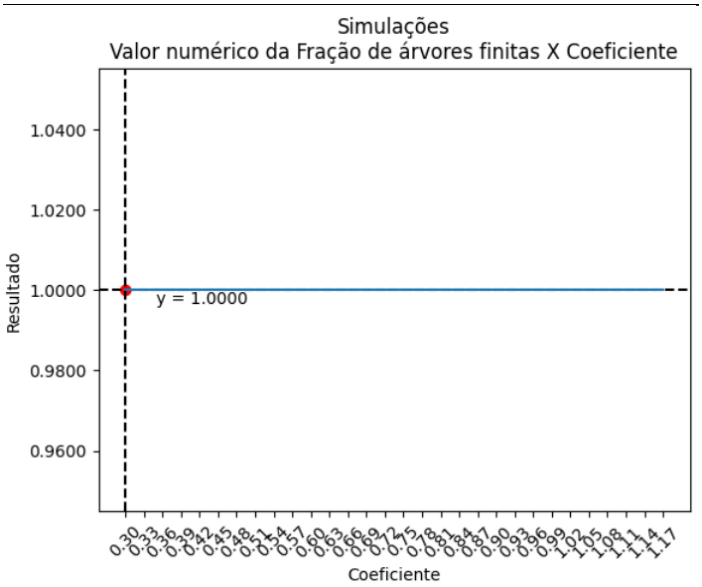
Verificação de Corretude para Média da distribuição Offspring (calculado com a PDF do 5.2)

mu = 0.5112000000000001 (caso subcrítico, logo temos 100% de períodos ocupados finitos)

Compatível com resultados da tabela

Caso 6 - Cenário 2 (determinístico)

Algoritmo de Monte Carlo para escolha do Coeficiente



Resultado analtico	Resultado com a heurística simples	Resultado da heurística simples + Backtracking
1	1.0	1.0

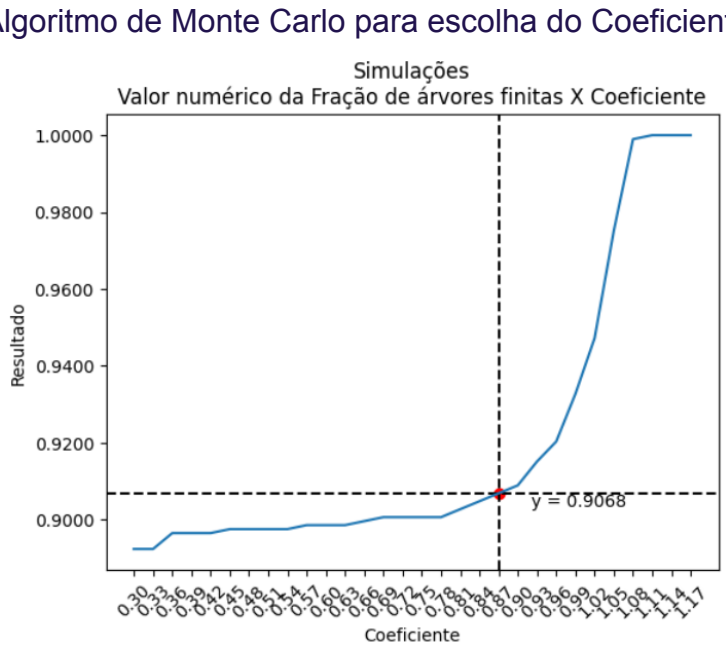
```
Valor analiticamente esperado de arvores finitas 1.0
Heurística simples:
Fração de arvores finitas 1.0

Algoritmo com Backtracking:
Fração de arvores finitas 1.0
intervalo de confianca: (1.0, 1.0)
```

Verificação de Corretude para Média da distribuição Offspring (calculado com a PDF do 5.2)

mu = 0.47461015593762496 (caso subcrítico, logo temos 100% de períodos ocupados finitos)

Compatível com resultados da tabela

Caso 7 - Cenário 3 (determinístico)



Resultado analtico	Resultado com a heurística simples	Resultado da heurística simples + Backtracking
0.90629	0.90890	0.90683

```
Valor analiticamente esperado de arvores finitas 0.9062981629270974
Heurística simples:
Fração de arvores finitas 0.9089026915113871
intervalo de confianca: (0.9014532414019378, 0.9163521416208364)

Algoritmo com Backtracking:
Fração de arvores finitas 0.906832298136646
intervalo de confianca: (0.8994683373232696, 0.9141962589500224)
```

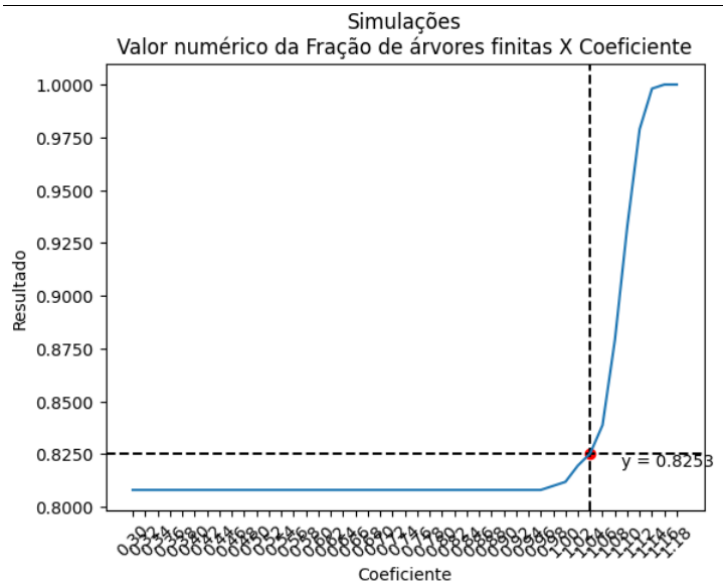
Verificação de Corretude para Média da distribuição Offspring (calculado com a PDF do 5.2)

mu = 0.9992363497518139 (caso crítico, logo podemos ter períodos ocupados finitos e infinitos)

Compatível com resultados da tabela

Caso 8 - Cenário 4 (determinístico)

Algoritmo de Monte Carlo para escolha do Coeficiente



Resultado analtico	Resultado com a heurística simples	Resultado da heurística simples + Backtracking
0.82386	0.80806	0.82533

```
Valor analiticamente esperado de arvores finitas 0.8238658563681878
Heurística simples:
Fração de arvores finitas 0.8080614203454894
intervalo de confianca: (0.7934318742511869, 0.822690966439792)
Algoritmo com Backtracking:
Fração de arvores finitas 0.8253358925143954
intervalo de confianca: (0.8103759649203062, 0.8402958201084847)
Valor analiticamente esperado de arvores finitas 0.8238658563681878
```

Verificação de Corretude para Média da distribuição Offspring (calculado com a PDF do 5.2)

mu = 0.9901925172539048 (caso crítico, logo podemos ter períodos ocupados finitos e infinitos)

Compatível com resultados

5.2 Estrutura das Árvores

Sobre as médias que são calculadas nessa parte, interpretamos alguns desses pontos com os conceitos que aprendemos sobre Branching Process.

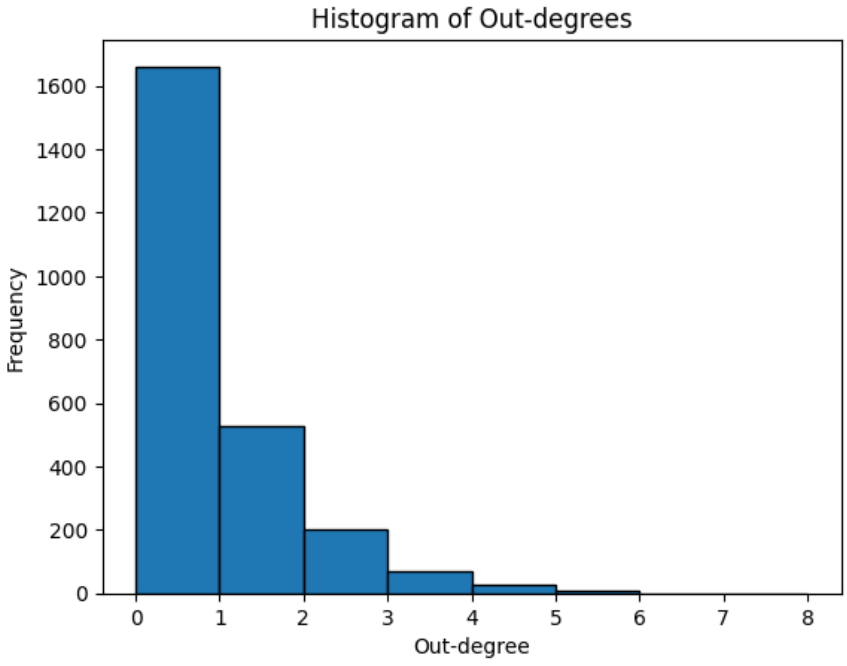
1) **Distribuição dos graus de saída** →Distribuição offspring (X_i)
2) **Média de saída da raiz** →Quantidade média de filhos do nó raiz de cada árvore, ou seja, tamanho médio da primeira geração
3) **Média do grau de saída máximo** →Média da geração realizada pelos nós com maior número de filhos de cada árvore
4) **Média das alturas das árvores** →Média do número de gerações até a extinção
5) **Média das alturas dos nós das árvores** → Média do número de gerações que a população pode atingir
6) **Média de duração do período ocupado** → Média do tempo de duração de uma epidemia
7) **Média do número de clientes atendidos por período ocupado** → Média do tamanho populacional de uma epidemia

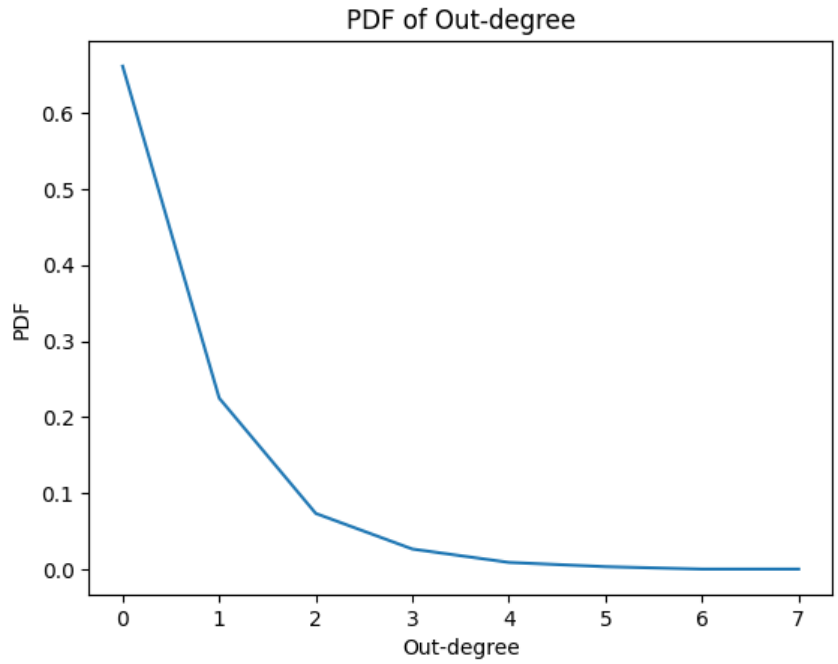
Simulações feitas com 100 rodadas e 2500 clientes

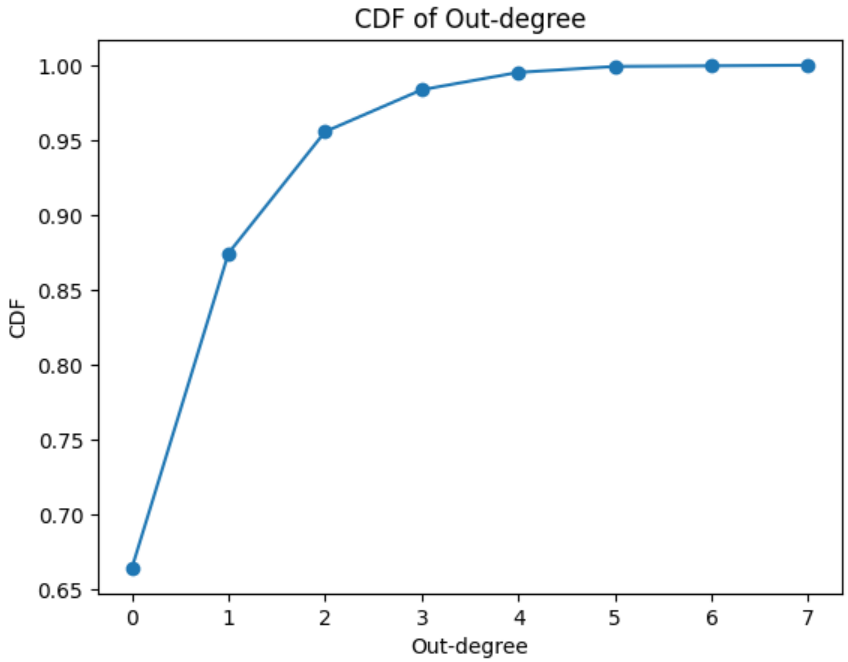
Caso 1

Lambda: 1
Mu: 2
Caso exponencial
Número de Rodadas: 100
Clientes: 2500

Distribuição dos graus de saída







Intervalos de confiança:

Grau médio de saída da raiz

0.49247605084316315, 0.5023781087969583

Média do grau de saída máximo

7.586625403094028, 7.993374596905972

Altura média da árvore

0.929057680838606, 0.9455656457268891

Média das alturas dos nós da árvore

0.29165851894881134, 0.5417917564083637

Média da duração do período ocupado

0.6578674259403887, 1.3417027148900251

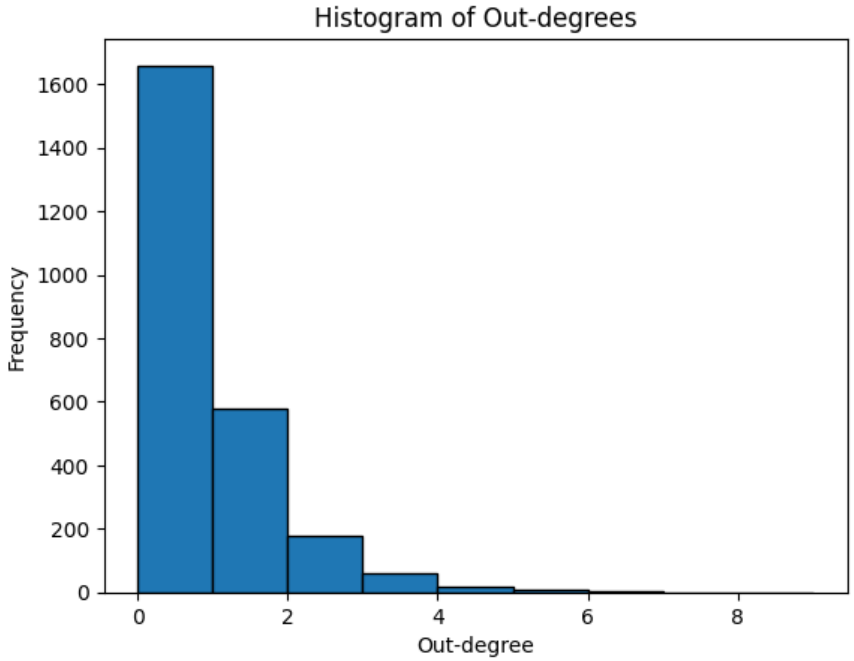
Média do número de clientes atendidos por período ocupado

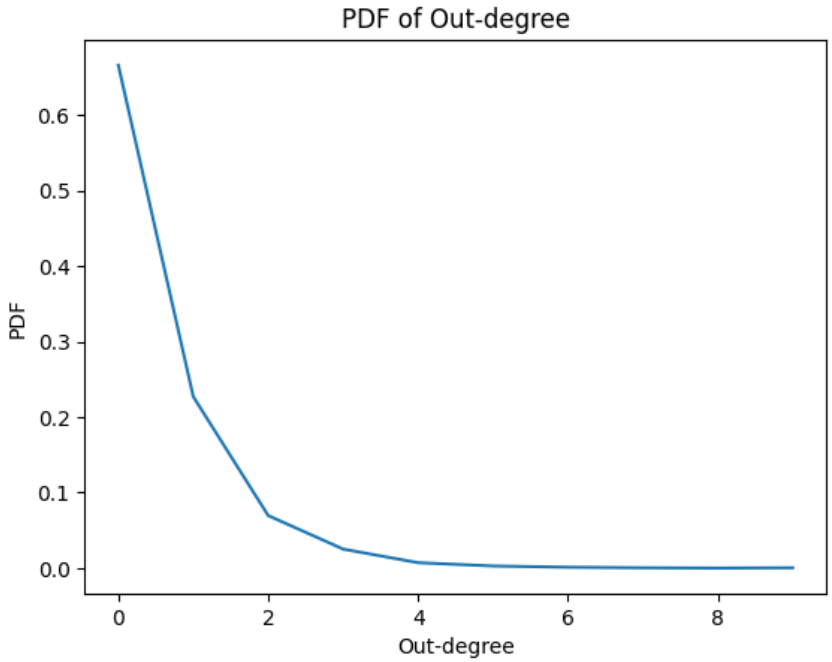
1.5164686734115524, 2.483851221662863

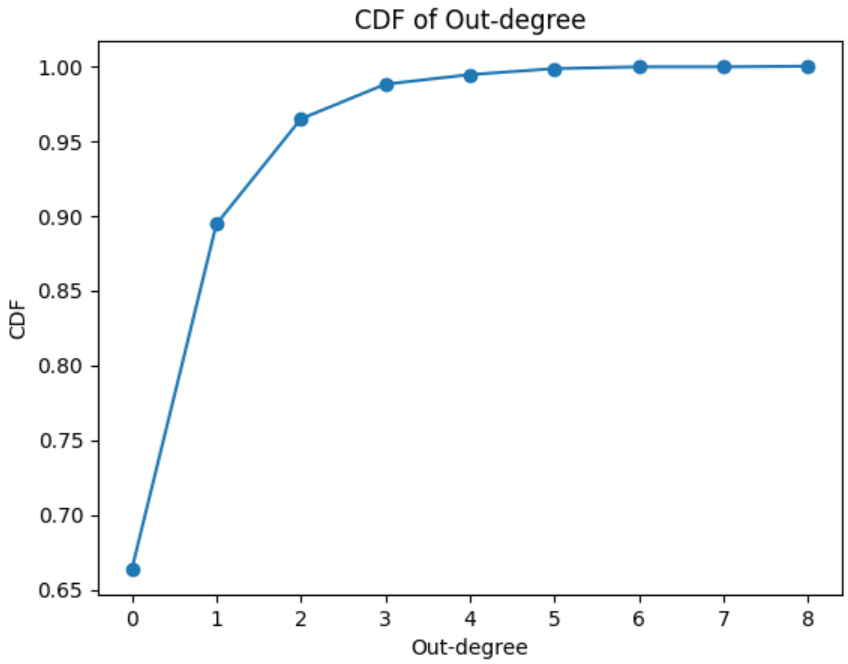
Caso 2

Lambda: 2
Mu: 4
Caso exponencial
Número de Rodadas: 100
Clientes: 2500

Distribuição dos graus de saída







Intervalos de confiança:

Grau médio de saída da raiz

0.5009572442116302, 0.5116478914264543

Média do grau de saída máximo

7.608138281431122, 8.091861718568877

Altura média da árvore

0.9406130039975882, 0.9579213648745686

Média das alturas dos nós da árvore

0.2963734293087061, 0.5472541184991215

Média da duração do período ocupado

0.3316897873373083, 0.6785199865650489

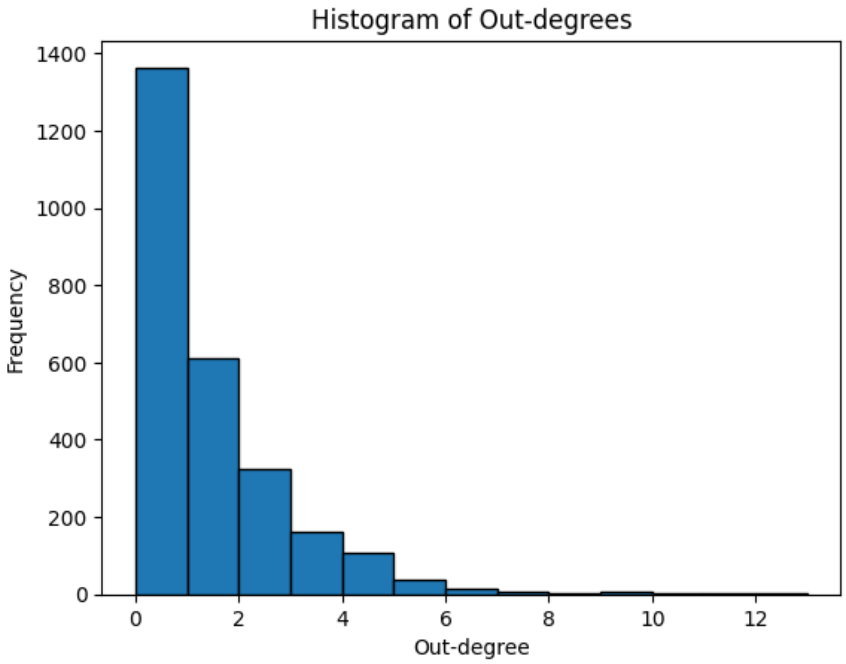
Média do número de clientes atendidos por período ocupado

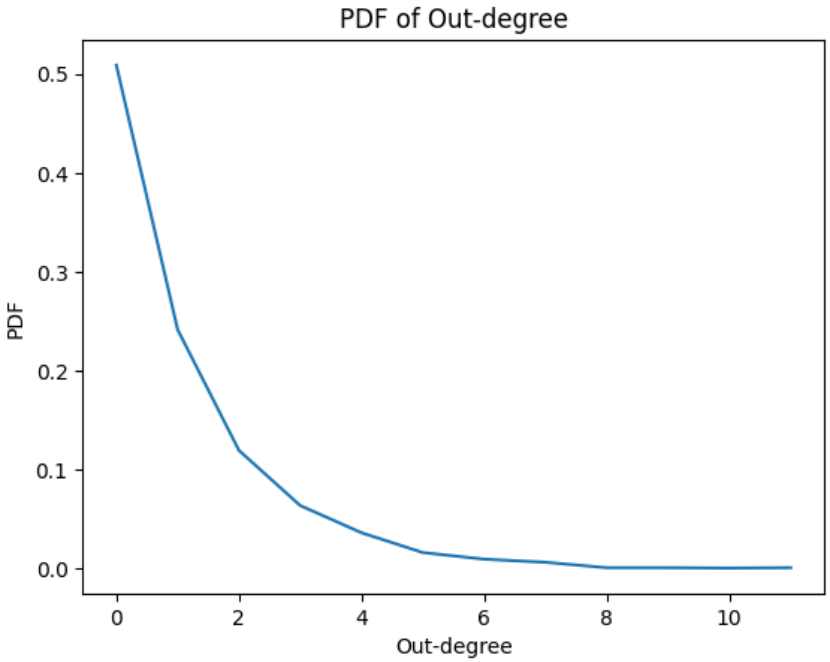
1.5219564635900769, 2.5023089758589627

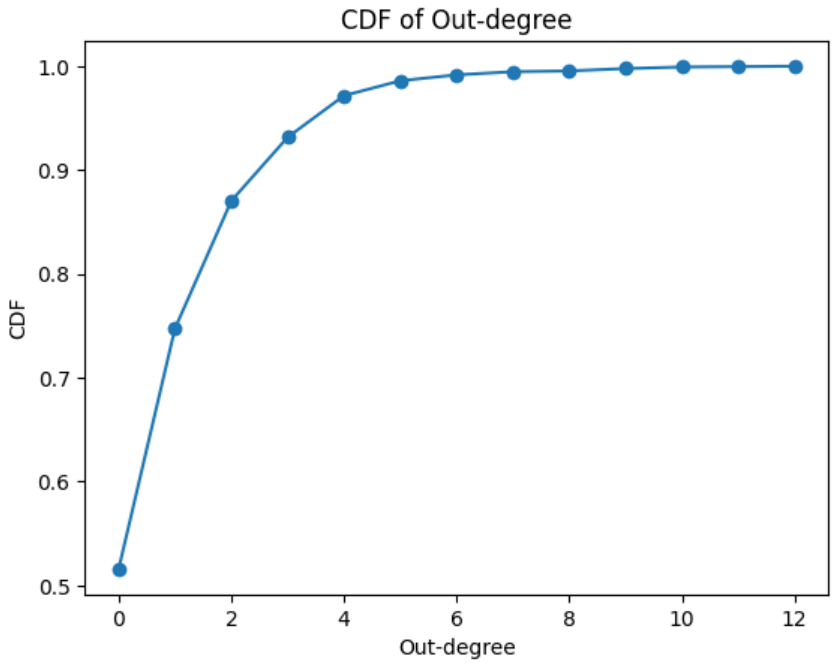
Caso 3

Lambda: 1.05
Mu: 1
Caso exponencial
Número de Rodadas: 100
Clientes: 2500

Distribuição dos graus de saída


Histogram of Out-degrees
Frequency
Out-degree
0
200
400
600
800
1000
1200
1400
0
2
4
6
8
10
12


PDF of Out-degree
PDF
Out-degree
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8
10



Intervalos de confiança:

Grau médio de saída da raiz

1.022329364584722, 1.2341441881446658

Média do grau de saída máximo

12.517519631453807, 13.282480368546194

Altura média da árvore

6.740467675079304, 9.415898845923202

Média das alturas dos nós da árvore

1.052966614918929, 2.722367649502857

Média da duração do período ocupado

-0.7528407358230762, 29.496031916784993

Nota: valor negativo devido intervalo de confiança. Podemos considerar como zero

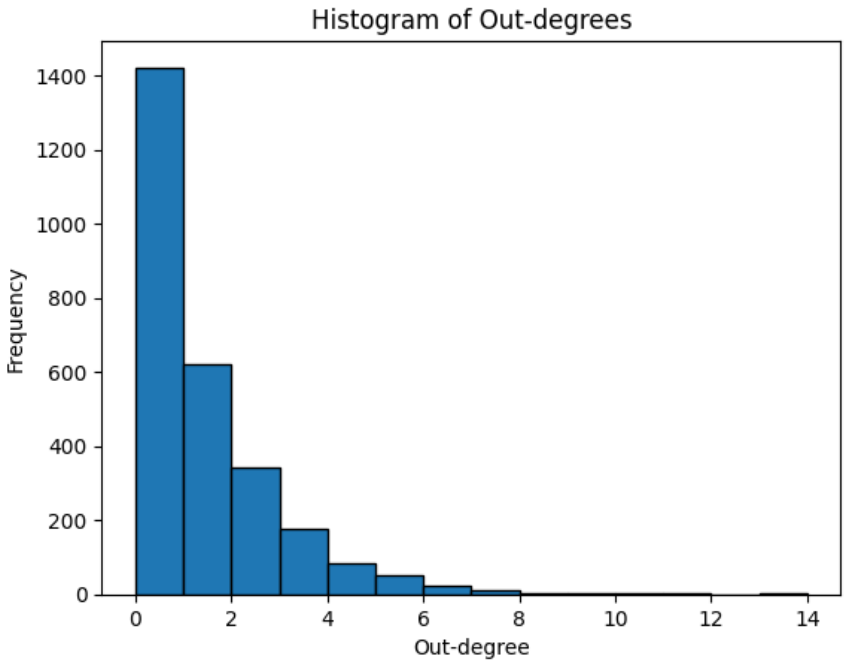
Média do número de clientes atendidos por período ocupado

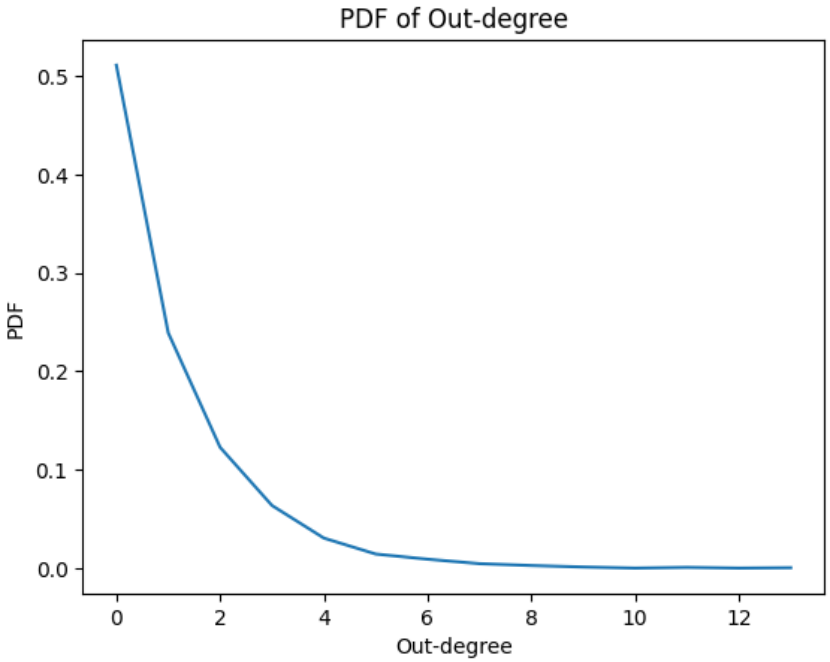
24.790409081920913, 223.86182621219672

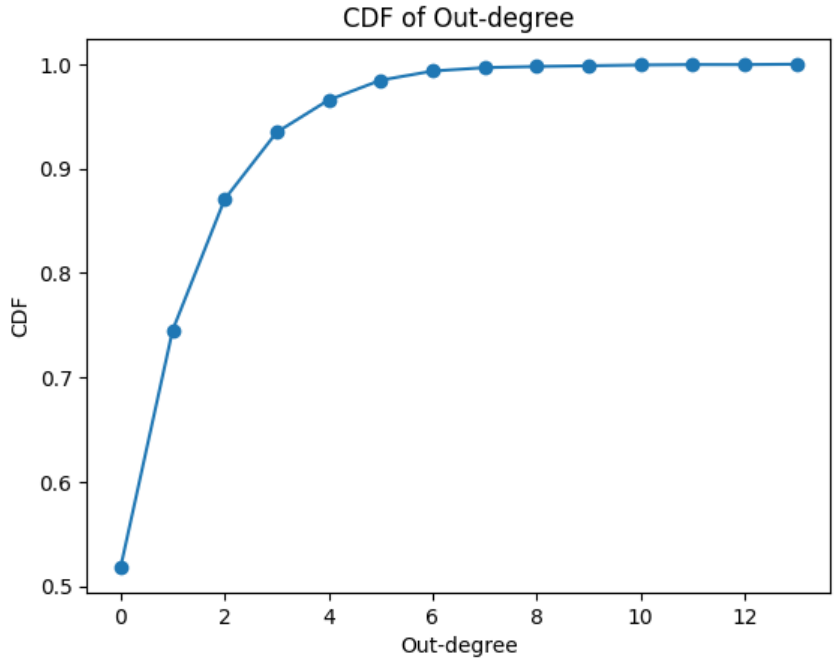
Caso 4

Lambda: 1.10
Mu: 1
Caso exponencial
Número de Rodadas: 100
Clientes: 2500

Distribuição dos graus de saída


Histogram of Out-degrees
Frequency
Out-degree
0
200
400
600
800
1000
1200
1400
0
2
4
6
8
10
12
14


PDF of Out-degree
PDF
Out-degree
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8
10
12



Intervalos de confiança:

Grau médio de saída da raiz

1.3966643075575433, 1.835720330320843

Média do grau de saída máximo

13.081444129560891, 13.738555870439109

Altura média da árvore

9.700607545138652, 13.30921390632258

Média das alturas dos nós da árvore

1.4352670362817637, 3.4898866345723647

Média da duração do período ocupado

0.9568839858828966, 15.701472444945281

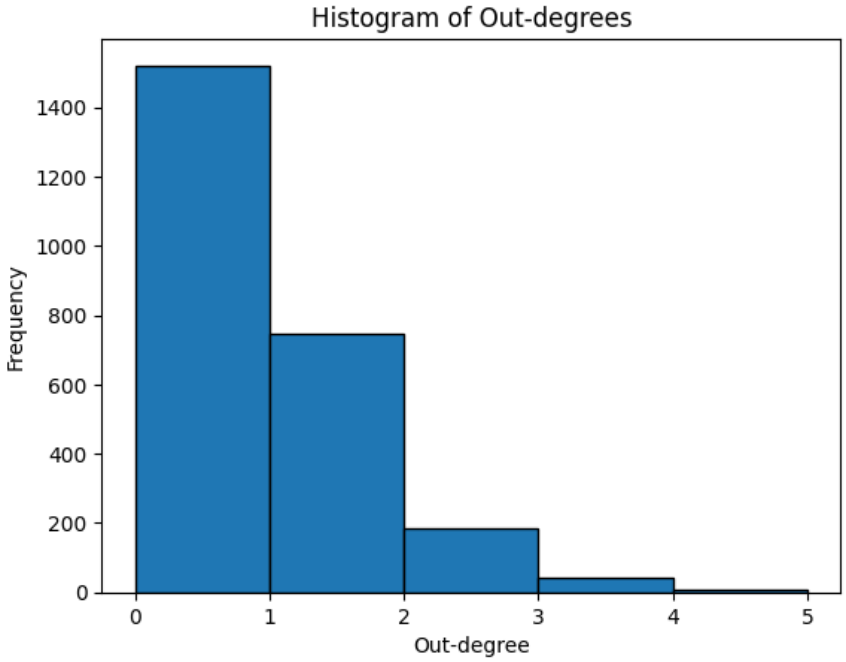
Média do número de clientes atendidos por período ocupado

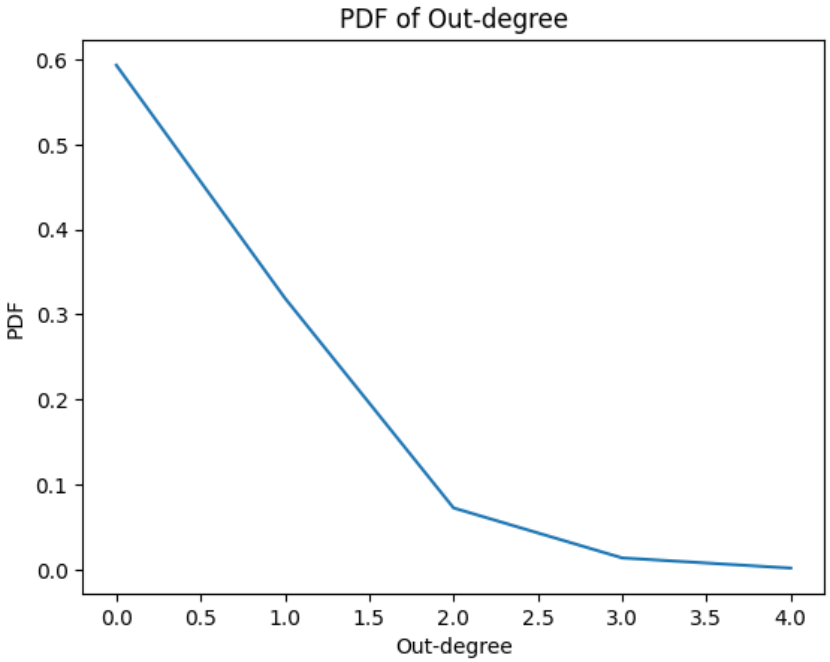
110.02005137194101, 418.71820366161603

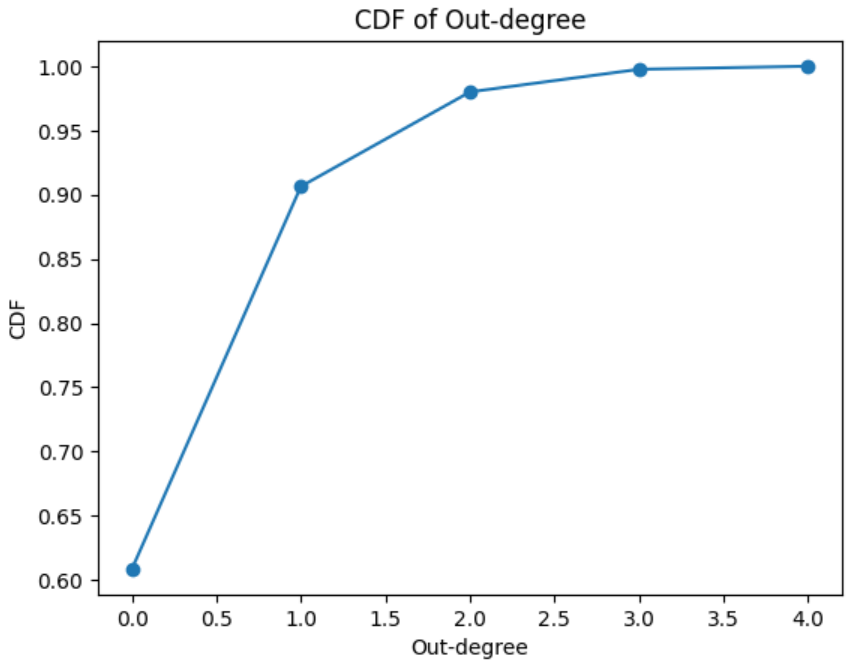
Caso 5

Lambda: 1
Mu: 2
Caso determinístico
Número de Rodadas: 100
Clientes: 2500

Distribuição dos graus de saída


Histogram of Out-degrees
Frequency
Out-degree
0
200
400
600
800
1000
1200
1400
0
1
2
3
4
5


PDF of Out-degree
PDF
Out-degree
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



Intervalos de confiança:

Grau médio de saída da raiz

0.4995102375791483, 0.5078460270304738

Média do grau de saída máximo

5.048117318409453, 5.251882681590548

Altura média da árvore

1.131973385424509, 1.1498747671459637

Média das alturas dos nós da árvore

0.3726260725257029, 0.6408462350466264

Média da duração do período ocupado

0.8076636042083669, 1.203913641784623

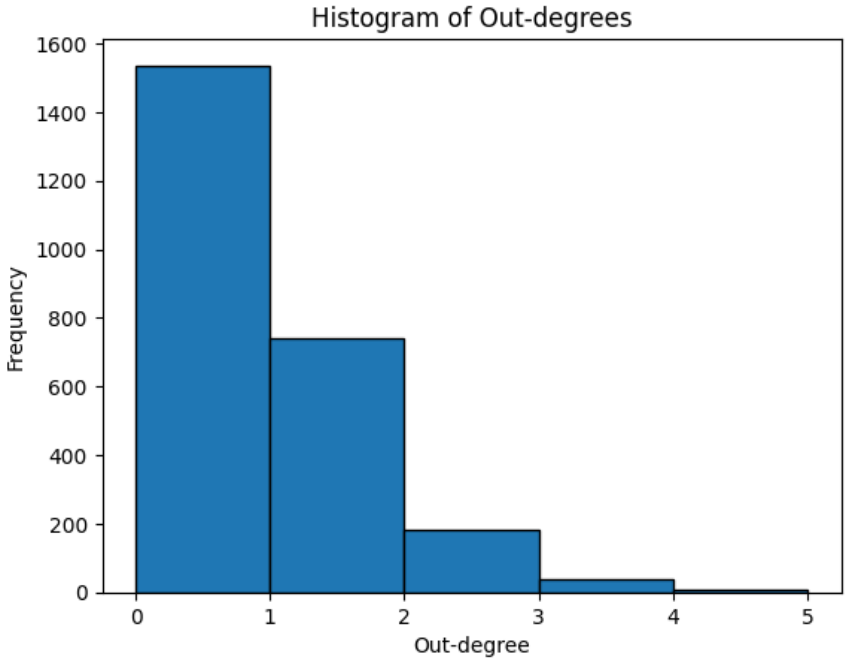
Média do número de clientes atendidos por período ocupado

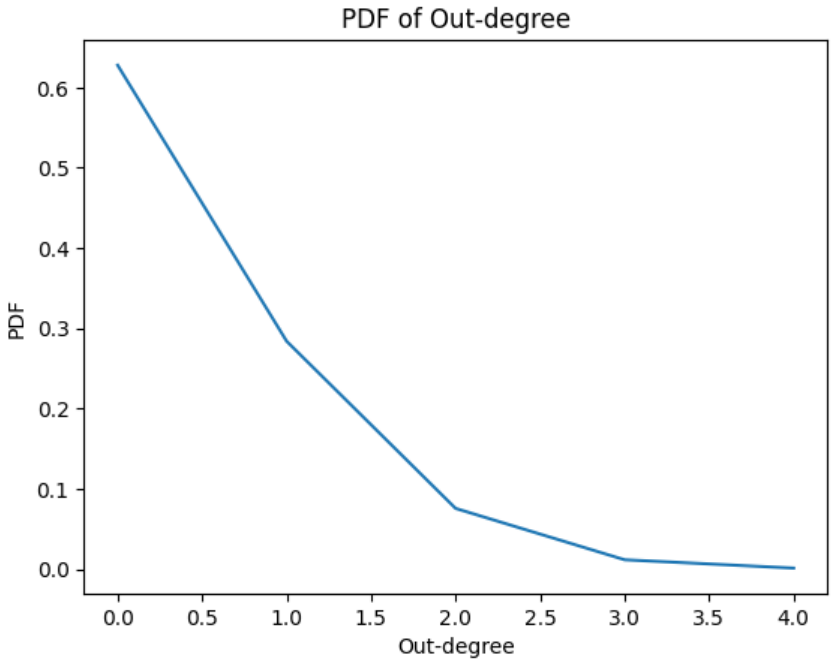
1.6148354010624892, 2.4074747535161594

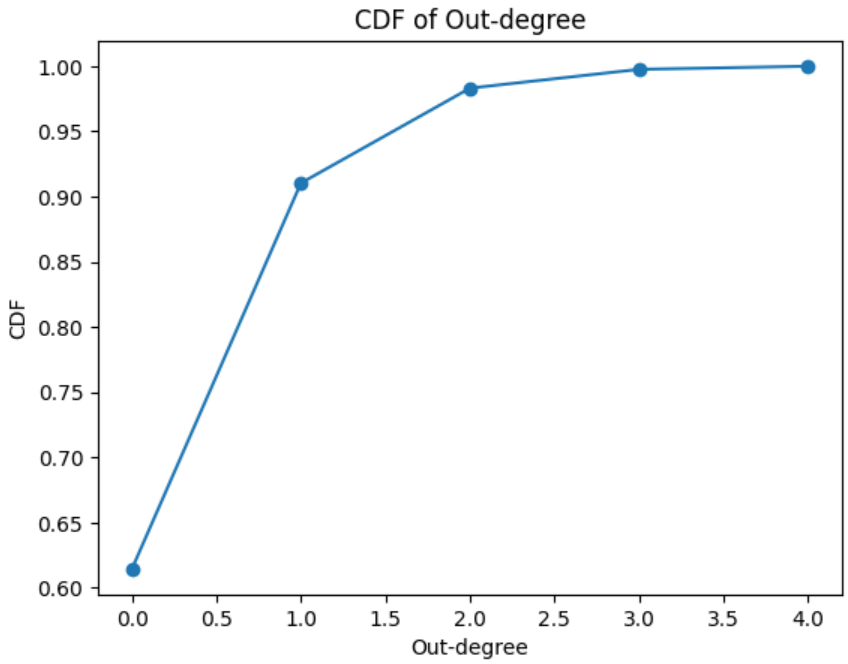
Caso 6

Lambda: 2
Mu: 4
Caso determinístico
Número de Rodadas: 100
Clientes: 2500

Distribuição dos graus de saída


Histogram of Out-degrees
Frequency
Out-degree
0
200
400
600
800
1000
1200
1400
1600
0
1
2
3
4
5


PDF of Out-degree
PDF
Out-degree
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



Intervalos de confiança:

Grau médio de saída da raiz

0.4979185404975007, 0.5057597949351583

Média do grau de saída máximo

5.0520000000000005, 5.248

Altura média da árvore

1.1289965952638767, 1.146268653450889

Média das alturas dos nós da árvore

0.37182543967715964, 0.6394845940774401

Média da duração do período ocupado

0.4031231338783703, 0.5990055386774751

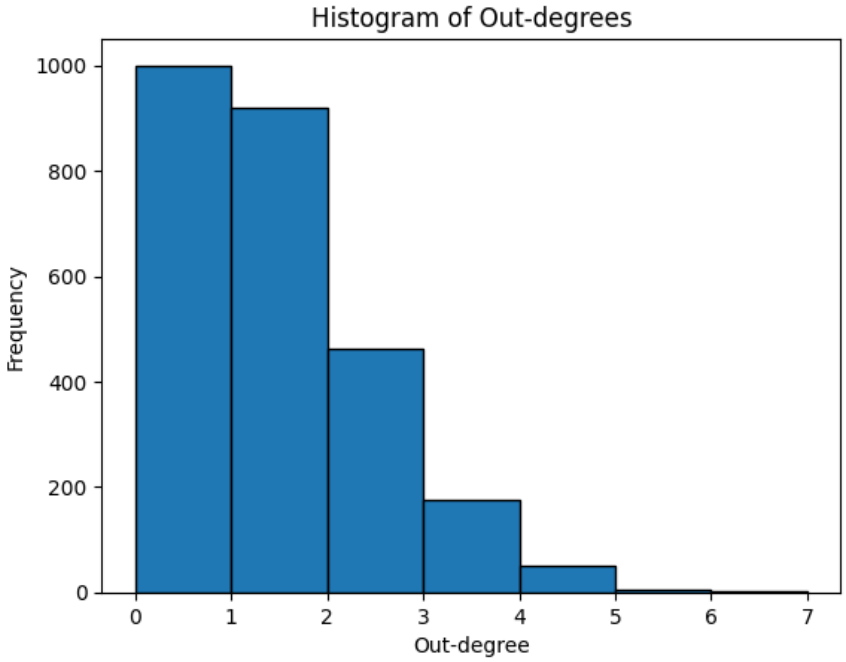
Média do número de clientes atendidos por período ocupado

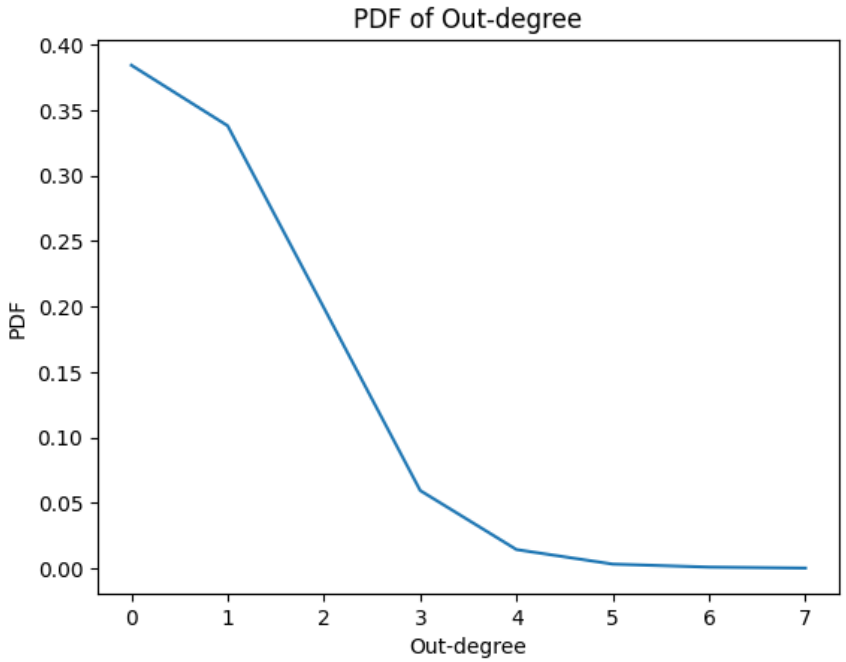
1.6118916456760737, 2.395093966526317

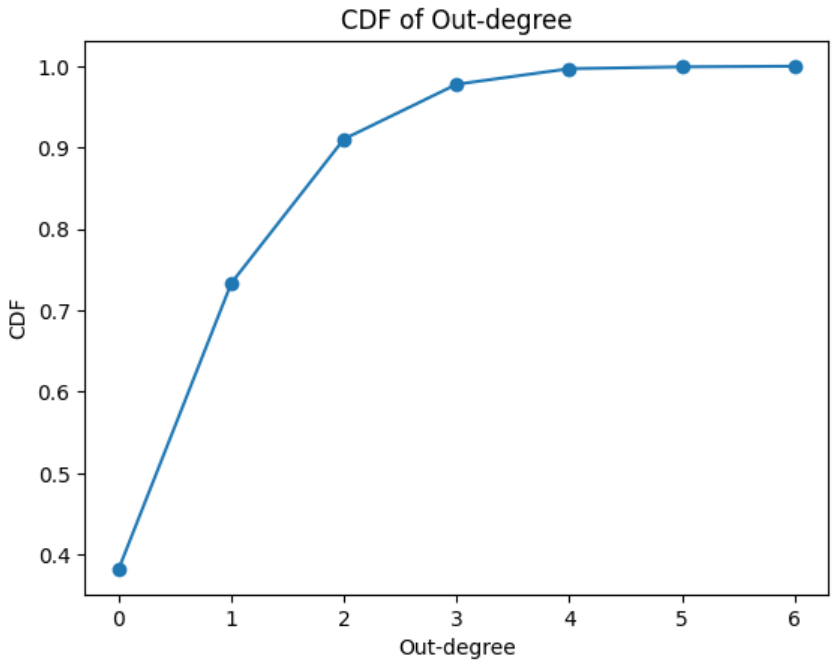
Caso 7

Lambda: 1.05
Mu: 1
Caso determinístico
Número de Rodadas: 100
Clientes: 2500

Distribuição dos graus de saída


Histogram of Out-degrees
Frequency
Out-degree
0
200
400
600
800
1000
0
1
2
3
4
5
6
7


PDF of Out-degree
PDF
Out-degree
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0
1
2
3
4
5
6
7



Intervalos de confiança:

Grau médio de saída da raiz

1.1306304941502092, 1.358350366932151

Média do grau de saída máximo

6.903750769068986, 7.176249230931014

Altura média da árvore

17.18212500432478, 25.431819842712066

Média das alturas dos nós da árvore

2.6655688955043297, 5.802377168909941

Média da duração do período ocupado

5.816138422841348, 37.15673869770336

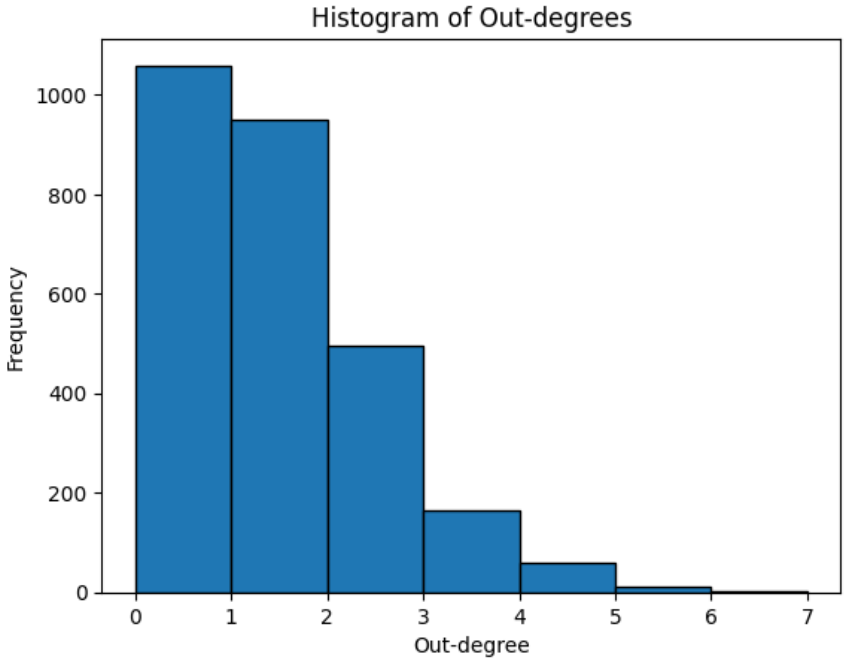
Média do número de clientes atendidos por período ocupado

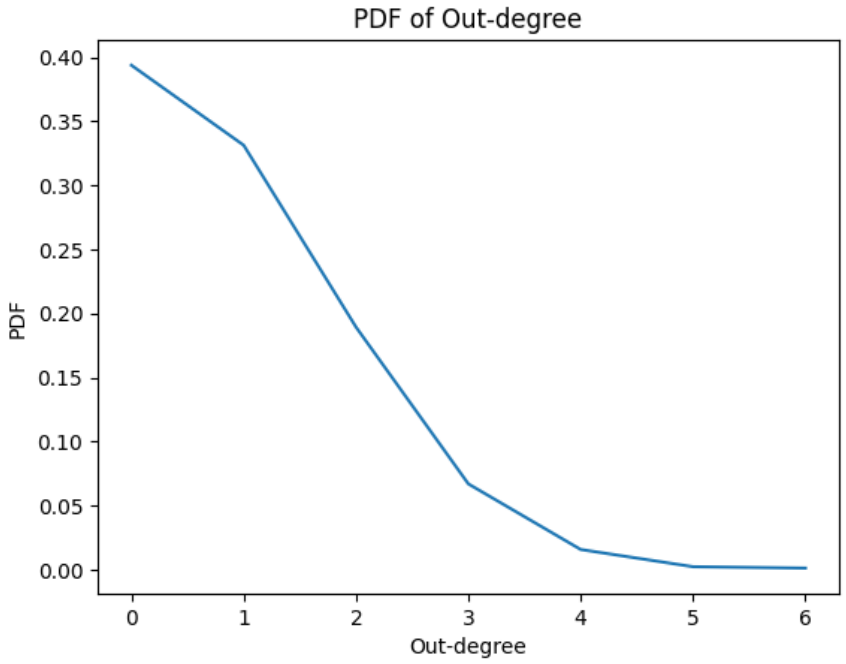
133.19278476162413, 433.96510997521796

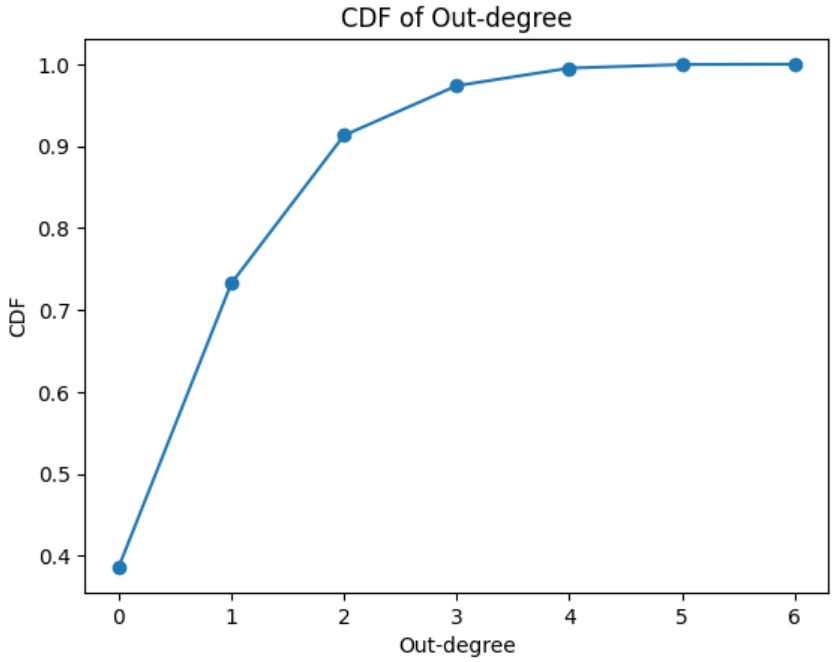
Caso 8

Lambda: 1.10
Mu: 1
Caso determinístico
Número de Rodadas: 100
Clientes: 2500

Distribuição dos graus de saída


Histogram of Out-degrees
Frequency
Out-degree
0
200
400
600
800
1000
0
1
2
3
4
5
6
7


PDF of Out-degree
PDF
Out-degree
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0
1
2
3
4
5
6



Intervalos de confiança:

Grau médio de saída da raiz

1.1692486632747392, 1.452226106384813

Média do grau de saída máximo

7.11531188289649, 7.38468811710351

Altura média da árvore

16.97493921955671, 22.76181810679845

Média das alturas dos nós da árvore

2.8549172174769746, 5.683293204626341

Média da duração do período ocupado

4.265230700796499, 20.289370663614235

Média do número de clientes atendidos por período ocupado

238.6206065194082, 620.8560289946105

5.3 Apresentação

Resultados com intervalo de confiança : Esses resultados estão contidos na seção 5.1.4 e na 5.2

5.4 Tempo de serviço determinístico

Para o tempo de serviço determinístico, consideramos $T = \frac{1}{\mu}$ e os resultados estão demonstrados nas seções anteriores, na 5.2 dos casos 5-8 e na seção 5.1.4 dos casos 5-8.

5.5 Tempo de serviço determinístico para Gamblers Ruins

Para realizar a comparação das soluções das simulações do trabalho 1, nos casos 3 e 4 da ruína do apostador (os que foram solicitados na primeira parte do trabalho). Primeiro, desenvolvemos um código python para resolver a equação com a integral da transformada da exponencial.

$$G(s) = \frac{\mu}{\mu + \lambda(1-s)} = \int_0^\infty e^{-\lambda x(1-s)} \mu e^{-\mu x} dx$$

No caso desta simulação onde a fila é finita e de no máximo 4 clientes, resolvemos a integral com o os limites de 0 a 4.

Segue o código utilizado neste trabalho:

```
def resolve_equacao_exponencial(s):
    x_values = np.linspace(0, 4, 400) # Discretização mais precisa
    integral_value = np.trapz(np.exp(-(_LAMBDA) * x_values * ((_MI) - s)) * np.exp(-(_MI) * x_values),
x_values)
    return s - integral_value
```

Caso 3 Gamblers' ruin:

Resolvendo a transformada

```
# Caso 3
_LAMBDA = 1.05
_MI = 1

# Suposição inicial para s
initial_guess = 0.5

s_solution = fsolve(resolve_equacao_exponencial, initial_guess)

print(f"Menor solução encontrada: {s_solution[0]}")
```

```
Menor solução encontrada: 0.8724359725563777
```

Solução encontrada na simulação da M/M/1

```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

TIME_MAX_ROUND = 100000

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


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"""\nFração 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: 0.875
Fração tempo até a ruina: 2.76538
Fração de passos até a ruina: 4.40000

Fração de fortuna: 0.125
Fração tempo até a fortuna: 7.77987
Fração de passos até a fortuna: 7.77987
```

Caso 4 Gamblers' ruin:

Resolvendo a integral para o caso 4

```
# Caso 4
_LAMBDA = 1.10
_MI = 1

# Suposição inicial para s
initial_guess = 0.5

s_solution = fsolve(resolve_equacao_exponencial, initial_guess)

print(f"Menor solução encontrada: {s_solution[0]}")
```

```
Menor solução encontrada: 0.8510895174716843
```

Solução encontrada na simulação da M/M/1

```
[ ]  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

     TIME_MAX_ROUND = 100000

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

     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_ruin:.5f}""")
     print(f"""Fração de passos até a ruina: {prob_steps_ruin:.5f}""")

     print(f"""\nFração 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: 0.85
Fração tempo até a ruina: 2.76538
Fração de passos até a ruina: 4.40000

Fração de fortuna: 0.15
Fração tempo até a fortuna: 4.28660
Fração de passos até a fortuna: 4.28660
```

Os valores da menor solução encontrada nos dois casos, são as frações do jogador ir à ruína. E esses valores batem com o mesmo resultado encontrado na simulação feita da M/M/1 no trabalho 1.

Corretude

Avaliamos a corretude da função de resolver a transformada exponencial, através da solução da solução pelo código do sympy apresentado acima e fazendo a prova real com a resolução da equação pelo **WolframAlpha.**

Caso 3:

Input

$$s = \int_0^4 e^{-1.05\,x(1-s)}\, e^{-x}\, dx$$

Result

$$s = \frac{0.952381 - 0.000261575\, e^{4.2\,s}}{1.95238 - s}$$

Solutions

$s \approx 0.872395$

$s \approx 1.16413$

$s \approx 1.95238$

Caso 4

Input

$$s = \int_0^4 e^{-1.1\,x(1-s)}\, e^{-x}\, dx$$

Result

$$s = \frac{0.909091 - 0.000204425\, e^{4.4\,s}}{1.90909 - s}$$

Solutions

$s \approx 0.851048$

$s \approx 1.1335$

$s \approx 1.90909$

6. Transformada G(s)

Transformada do número de clientes que chegam durante um serviço:

$$G(s) = e^{-\lambda T(1-s)}$$

Serviço determinístico

$$G(s) = \frac{\mu}{\mu + \lambda(1-s)} = \int_0^\infty e^{-\lambda x(1-s)} \mu e^{-\mu x} dx$$

Serviço exponencial

Essas equações são utilizadas para encontrarmos os valores analíticos do tópico **[5.1]**.
Como no 5.1 queremos saber a probabilidade de extinção, necessitamos resolver a equação

s= G(s)

Vamos resolver primeiro para o caso exponencial.

$$s = G(s)$$
$$s = \frac{\mu}{\mu + \lambda(1-s)}$$

$$s(\mu + \lambda(1 - s)) - \mu = 0$$
$$s\mu + \lambda s - \lambda s^2 - \mu = 0$$

reorganizando para aplicar bhaskara

$$-\lambda s^2 + s(\mu + \lambda) - \mu = 0$$

Aplicando a fórmula quadrática e resolvendo ficamos com duas raízes para s

$$s = 1 \text{ ou } s = \frac{\mu}{\lambda}$$

Assim, podemos já achar quais os valores analiticos esperados para os casos exponenciais, visto que queremos a menor raiz

cenário	raiz 1	raiz 2	raiz escolhida para probabilidade de extinção
caso 1 (λ: 1 e μ: 2)	1	$\frac{\mu}{\lambda} = \frac{2}{1} = 2$	1 (raiz 1)
caso 2 (λ: 2 e μ: 4)	1	$\frac{\mu}{\lambda} = \frac{4}{2} = 2$	1 (raiz 1)
caso 3 (λ: 1.05 e μ: 1)	1	$\frac{\mu}{\lambda} = \frac{1}{1.05} = 0.952380$	0.952380 (raiz 2)
caso 4 (λ: 1.10 e μ: 1)	1	$\frac{\mu}{\lambda} = \frac{1}{1.10} = 0.909090$	0.909090 (raiz 2)

Agora vamos realizar o mesmo processo para o **caso determinístico.**

$$s = G(s)$$
$$s = e^{-\lambda T(1-s)}$$
$$\ln(s) = ln(e^{-\lambda T(1-s)})$$
$$ln(s) = -\lambda T(1-s)$$
$$ln(s) + \lambda T(1-s) = 0$$

Dessa equação temos definido $T = 1/\mu$

$$ln(s) + \frac{\lambda}{\mu}(1-s) = 0$$

A partir desse momento não conseguimos resolver sem a ajuda computacional, nós fizemos um código utilizando python sympy e também demos um double check nos valores com o wolfram alpha.

```python
from scipy.optimize import root
import numpy as np
# Chute para raizes inicias
x0_1 = 0.3  # chute para a primeira raiz
x0_2 = 1.0  # chute para a segunda raiz

def equation_to_solve(x, a, b):
    # x = s, a = lambda, b = mu
    return x - np.exp(-a*b*(1 - x))


def get_resultado_analitico(cenario, tipo):
  if tipo == "d": # deterministico
    if cenario == 1:
      return root(equation_to_solve, x0_1, args=(1, 1/2.)).x[0]
    elif cenario == 2:
      return root(equation_to_solve, x0_1, args=(2, 1/4.)).x[0]
    elif cenario == 3:
      return root(equation_to_solve, x0_1, args=(1.05, 1.)).x[0]
    elif cenario == 4:
      return root(equation_to_solve, x0_1, args=(1.10, 1.)).x[0]
```

código utilizado para a resolução da equação.
X[0] retorna a primeira raiz, que no nosso caso é a menor.

cenário	raiz 1	raiz 2	raiz escolhida para probabilidade de extinção
caso 1 (λ: 1 e μ: 2)	1.	1.	1.0 (raiz 1)
caso 2 (λ: 2 e μ: 4)	1.	1.	1.0 (raiz 1)
caso 3 (λ: 1. 05 e μ: 1)	0.90629	1.	0.90629 (raiz 1)
caso 4 (λ: 1. 10 e μ: 1)	0.82386	1.	0.82386 (raiz 1)

```
caso 1 (deterministico):
 raiz 1: 1.0
 raiz 2: 1.0
caso 2 (deterministico):
 raiz 1: 1.0
 raiz 2: 1.0
caso 3 (deterministico):
 raiz 1: 0.9062981629270974
 raiz 2: 1.0
caso 4 (deterministico):
 raiz 1: 0.8238658563681878
 raiz 2: 1.0
```

7. Traceability

O Debug do simulador foi realizado principalmente de duas formas:

Breakpoints

```
import pdb
pdb.set_trace()
```

Código para configuração de breakpoint no Jupyter Notebook/Google Colab

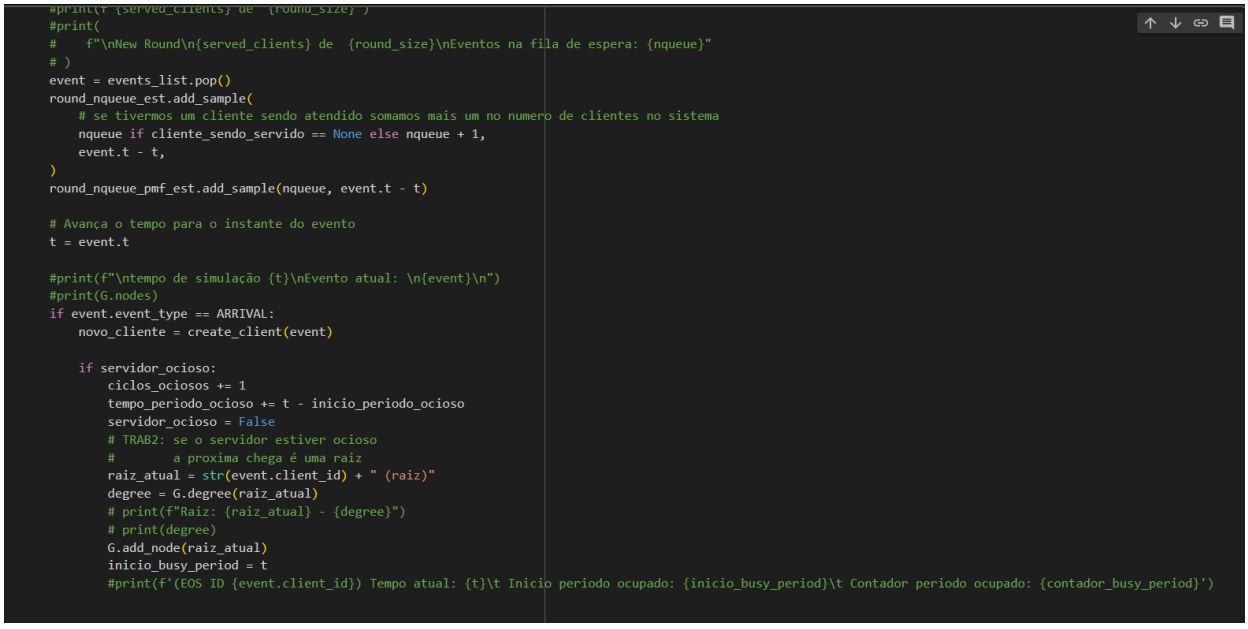
Pela utilização de breakpoints podemos interagir com o simulador no meio da execução do mesmo, permitindo análise de variáveis, stack trace, análise de erros, entre outros.

Prints

Com a utilização de prints podemos facilmente observar os valores de diversas variáveis ao longo da simulação. Por poluir consideravelmente a saída do simulador, os prints de Debug foram comentados, mas ainda estão presentes no código fonte.

```
caso exponencial
(EOS ID 0) Tempo atual: 0.20783778897885408      Inicio periodo ocupado: 0       Contador periodo ocupado: 0.20783778897885408
(EOS ID 1) Tempo atual: 1.0769987014648374       Inicio periodo ocupado: 1.0769987014648374      Contador periodo ocupado: 0
(EOS ID 1) Tempo atual: 1.191756054977989        Inicio periodo ocupado: 1.0769987014648374      Contador periodo ocupado: 0.11475735351315164
(EOS ID 2) Tempo atual: 4.1469715152665065       Inicio periodo ocupado: 4.1469715152665065      Contador periodo ocupado: 0
(EOS ID 3) Tempo atual: 5.211011588602196        Inicio periodo ocupado: 4.1469715152665065      Contador periodo ocupado: 1.0640400733356898
(EOS ID 4) Tempo atual: 5.556527402832257        Inicio periodo ocupado: 5.556527402832257       Contador periodo ocupado: 0
(EOS ID 9) Tempo atual: 9.665865201517786        Inicio periodo ocupado: 5.556527402832257       Contador periodo ocupado: 4.109337798685528
(EOS ID 10) Tempo atual: 10.369885335350794      Inicio periodo ocupado: 10.369885335350794      Contador periodo ocupado: 0
(EOS ID 12) Tempo atual: 12.152552712099604      Inicio periodo ocupado: 10.369885335350794      Contador periodo ocupado: 1.78266737674881
(EOS ID 13) Tempo atual: 12.481543360729646      Inicio periodo ocupado: 12.481543360729646      Contador periodo ocupado: 0
(EOS ID 13) Tempo atual: 12.497058372653564      Inicio periodo ocupado: 12.481543360729646      Contador periodo ocupado: 0.015515011923918465
(EOS ID 14) Tempo atual: 13.983650449506765      Inicio periodo ocupado: 13.983650449506765      Contador periodo ocupado: 0
(EOS ID 14) Tempo atual: 14.09956391906668       Inicio periodo ocupado: 13.983650449506765      Contador periodo ocupado: 0.11591346955991533
(EOS ID 15) Tempo atual: 14.560162465855704      Inicio periodo ocupado: 14.560162465855704      Contador periodo ocupado: 0
(EOS ID 15) Tempo atual: 14.87323581887779       Inicio periodo ocupado: 14.560162465855704      Contador periodo ocupado: 0.3130733530220855
(EOS ID 16) Tempo atual: 17.36798914023865       Inicio periodo ocupado: 17.36798914023865       Contador periodo ocupado: 0
(EOS ID 18) Tempo atual: 18.06984822485075       Inicio periodo ocupado: 17.36798914023865       Contador periodo ocupado: 0.7018590846121029
(EOS ID 19) Tempo atual: 18.76174476267837       Inicio periodo ocupado: 18.76174476267837       Contador periodo ocupado: 0
(EOS ID 22) Tempo atual: 21.273908614594053      Inicio periodo ocupado: 18.76174476267837       Contador periodo ocupado: 2.512163851915684
(EOS ID 23) Tempo atual: 22.222369149426616      Inicio periodo ocupado: 22.222369149426616      Contador periodo ocupado: 0
(EOS ID 27) Tempo atual: 25.27473560184366       Inicio periodo ocupado: 22.222369149426616      Contador periodo ocupado: 3.0523664524170435
(EOS ID 28) Tempo atual: 26.61136527700404       Inicio periodo ocupado: 26.61136527700404       Contador periodo ocupado: 0
(EOS ID 29) Tempo atual: 27.19039438635161       Inicio periodo ocupado: 26.61136527700404       Contador periodo ocupado: 0.5790291093475695
(EOS ID 30) Tempo atual: 28.45898390600731       Inicio periodo ocupado: 28.45898390600731       Contador periodo ocupado: 0
(EOS ID 30) Tempo atual: 28.52205893917987       Inicio periodo ocupado: 28.45898390600731       Contador periodo ocupado: 0.06307503317255936
(EOS ID 31) Tempo atual: 29.853533818932778      Inicio periodo ocupado: 29.853533818932778      Contador periodo ocupado: 0
(EOS ID 32) Tempo atual: 30.99727743022772       Inicio periodo ocupado: 29.853533818932778      Contador periodo ocupado: 1.1437436112949406
(EOS ID 33) Tempo atual: 31.23267894367401       Inicio periodo ocupado: 31.23267894367401       Contador periodo ocupado: 0
(EOS ID 34) Tempo atual: 32.077969423738296      Inicio periodo ocupado: 31.23267894367401       Contador periodo ocupado: 0.8452904800642855
(EOS ID 35) Tempo atual: 35.13458325973147       Inicio periodo ocupado: 35.13458325973147       Contador periodo ocupado: 0
(EOS ID 36) Tempo atual: 36.61197722475832       Inicio periodo ocupado: 35.13458325973147       Contador periodo ocupado: 1.4773939650268488
(EOS ID 37) Tempo atual: 38.19603732179339       Inicio periodo ocupado: 38.19603732179339       Contador periodo ocupado: 0
(EOS ID 37) Tempo atual: 38.20238454990107       Inicio periodo ocupado: 38.19603732179339       Contador periodo ocupado: 0.006347228107678404
(EOS ID 38) Tempo atual: 38.716613135390496      Inicio periodo ocupado: 38.716613135390496      Contador periodo ocupado: 0
(EOS ID 54) Tempo atual: 49.92371365553016       Inicio periodo ocupado: 38.716613135390496      Contador periodo ocupado: 11.207100520139662
(EOS ID 55) Tempo atual: 50.48598395453987       Inicio periodo ocupado: 50.48598395453987       Contador periodo ocupado: 0
(EOS ID 55) Tempo atual: 50.53069353954609       Inicio periodo ocupado: 50.48598395453987       Contador periodo ocupado: 0.04470958500622402
(EOS ID 56) Tempo atual: 50.59531807776893       Inicio periodo ocupado: 50.59531807776893       Contador periodo ocupado: 0
(EOS ID 56) Tempo atual: 50.774773380476624      Inicio periodo ocupado: 50.59531807776893       Contador periodo ocupado: 0.1794553027076944
(EOS ID 57) Tempo atual: 52.15010752361236       Inicio periodo ocupado: 52.15010752361236       Contador periodo ocupado: 0
(EOS ID 57) Tempo atual: 52.69854859637774       Inicio periodo ocupado: 52.15010752361236       Contador periodo ocupado: 0.5484410727653781
(EOS ID 58) Tempo atual: 54.04527416795683       Inicio periodo ocupado: 54.04527416795683       Contador periodo ocupado: 0
(EOS ID 58) Tempo atual: 54.50026995892061       Inicio periodo ocupado: 54.04527416795683       Contador periodo ocupado: 0.4549957909637783
(EOS ID 59) Tempo atual: 55.31932556544118       Inicio periodo ocupado: 55.31932556544118       Contador periodo ocupado: 0
```

Exemplo de print para fins de Debug da apuração da média de duração dos períodos ocupados



```
#print(f"{served_clients} de  {round_size}")
#print(
#    f"\nNew Round\n{served_clients} de  {round_size}\nEventos na fila de espera: {nqueue}"
# )
event = events_list.pop()
round_nqueue_est.add_sample(
    # se tivermos um cliente sendo atendido somamos mais um no numero de clientes no sistema
    nqueue if cliente_sendo_servido == None else nqueue + 1,
    event.t - t,
)
round_nqueue_pmf_est.add_sample(nqueue, event.t - t)

# Avança o tempo para o instante do evento
t = event.t

#print(f"\ntempo de simulação {t}\nEvento atual: \n{event}\n")
#print(G.nodes)
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
        # TRAB2: se o servidor estiver ocioso
        #        a proxima chega é uma raiz
        raiz_atual = str(event.client_id) + " (raiz)"
        degree = G.degree(raiz_atual)
        # print(f"Raiz: {raiz_atual} - {degree}")
        # print(degree)
        G.add_node(raiz_atual)
        inicio_busy_period = t
        #print(f'(EOS ID {event.client_id}) Tempo atual: {t}\t Inicio periodo ocupado: {inicio_busy_period}\t Contador periodo ocupado: {contador_busy_period}')
```

Exemplo de prints comentados que utilizamos.

8.Bibliografia

Robert P. Dobrow - Introduction to stochastic processes with R-John Wiley & Sons (2016)

[Analyzing the M/G/1 Queue with a Branching Process]