



**INSTITUTO
FEDERAL**

Santa Catarina

Câmpus
São José

Relatório de Miniprojeto: Cadeias de Markov a Tempo Contínuo - CTMC

Aplicação: Controle de Admissão

Arthur Cadore Matuella Barcella
Deivid Fortunato Frederico

20 de Maio de 2025

Engenharia de Telecomunicações - IFSC-SJ

Sumário

1. Introdução	3
1.1. Modelagem do Sistema	3
1.2. Parâmetros do Sistema	3
1.3. Problema Proposto	3
1.3.1. Parte 1: (Tarefas)	3
1.3.2. Parte 2 (Cálculos):	3
1.3.3. Parte 3 (Simulação):	3
2. Solução dos exercícios	4
2.1. Resolução Analítica	4
2.1.1. Probabilidade de Rejeição	6
2.1.2. Cálculo dos Indicadores	7
2.2. Simulação da CTMC	9
2.3. Análise das comparações entre os métodos	15
2.3.1. A proximidade entre os métodos	16
2.3.2. Qual a abordagem mais intuitiva	16
2.3.3. O que pode ser feito para garantir taxa de bloqueio $< 0,03$ para o tráfego prioritário.	16
3. Conclusão	16

1. Introdução

O presente relatório tem como objetivo apresentar o desenvolvimento de um miniprojeto na disciplina de Engenharia de Telecomunicações, com foco na modelagem e análise de sistemas utilizando Cadeias de Markov a Tempo Contínuo (CTMC). O projeto foi realizado por Arthur Cadore Matuella Barcella e Deivid Fortunato Frederico, alunos do curso de Engenharia de Telecomunicações do IFSC-SJ.

1.1. Modelagem do Sistema

A atividade envolve o seguinte contexto:

- A rede possui duas classes de tráfego: prioritário (T_1) e não prioritário (T_2).
- Uma sessão estabelecida para uma classe requer 1 unidade de tráfego (1 unidade poderia ser 64kbps por exemplo);
- Capacidade total: $C = 5$ unidades.
- O sistema deve sempre ser capaz de atender no mínimo 2 sessões de tráfego prioritário. Ou seja, é mantida uma reserva mínima de $R = 2$ unidades para atender a classe prioritária (T_1).

1.2. Parâmetros do Sistema

Parâmetros:

- Chegadas: Poisson com taxas λ_1 (T_1) e λ_2 (T_2)
- Tempo de serviço: Exponencial com taxas μ_1 (T_1) e μ_2 (T_2)

1.3. Problema Proposto

Objetivo: Analisar desempenho do sistema com controle de admissão via CTMC.

1.3.1. Parte 1: (Tarefas)

- Modelar CTMC com $C = 5$, $R = 2$
- Identificar estados válidos e matriz Q
- Resolver analiticamente $\pi Q = 0$ para distribuição estacionária π

1.3.2. Parte 2 (Cálculos):

- Probabilidade de bloqueio (rejeição) de requisições de cada classe (T_1 , T_2)
- Utilização média: $U = \frac{1}{C} \sum_{n_1, n_2} (n_1 + n_2) \pi(n_1, n_2)$
- Conexões médias: $L_1 = \sum n_1 \pi(n_1, n_2)$, $L_2 = \sum n_2 \pi(n_1, n_2)$
- Fração em estados máximos ($i + j = 5$)

Parâmetros: $\lambda_1 = 10$, $\lambda_2 = 15$, $\mu_1 = 15$, $\mu_2 = 25$ (req/min)

1.3.3. Parte 3 (Simulação):

- Simular CTMC por $T = 10,000$ min
- Comparar métodos: A) Tempo de estadia exponencial B) Relógios concorrentes

- Discutir aproximação, intuitividade e como garantir $P_{\text{bloqueio}} < 0.03$ para T_1

2. Solução dos exercícios

Os códigos usados na tarefa estão disponíveis no repositório do projeto no GitHub (<https://github.com/arthurcadore/eng-telecom-workbook/tree/main/semester-9/ADS/homework4>).

Conjunto de Estados

Estados válidos (n_1, n_2) onde: $n_1 + n_2 \leq C$ e $n_2 \leq C - R$

(0, 0), (0, 1), (0, 2), (0, 3),
 (1, 0), (1, 1), (1, 2), (1, 3),
 (2, 0), (2, 1), (2, 2), (2, 3),
 (3, 0), (3, 1), (3, 2),
 (4, 0), (4, 1),
 (5, 0).

2.1. Resolução Analítica

Matriz de Transição Q

$$Q(i, j) = \begin{cases} \lambda_1, & \text{se } (n_1+1, n_2) \text{ é válido} \\ \lambda_2, & \text{se } (n_1, n_2+1) \text{ é válido} \\ n_1\mu_1, & \text{se } (n_1-1, n_2) \text{ é válido} \\ n_2\mu_2, & \text{se } (n_1, n_2-1) \text{ é válido} \\ -\sum_{k \neq i} Q(i, k) & \text{diagonal principal} \end{cases}$$

Bibliotecas e Parâmetros usados no código:

```
1 import numpy as np
2 import itertools
3 import matplotlib.pyplot as plt
4 import random
5 import seaborn as sns
6 import pandas as pd
7
8 # Parâmetros
9 C = 5 # capacidade total
10 R = 2 # reserva para tráfego prioritário (classe 1)
11 λ1 = 10 / 60 # taxa de chegada classe 1 (por minuto)
12 λ2 = 15 / 60 # taxa de chegada classe 2 (por minuto)
13 μ1 = 15 / 60 # taxa de serviço classe 1 (por minuto)
14 μ2 = 25 / 60 # taxa de serviço classe 2 (por minuto)
15
16 # Tempo total de simulação
17 T_max = 10_000 # minutos
```

A seguir, apresenta-se o código utilizado para gerar a matriz de transição e resolver o sistema estacionário $\pi Q = 0$:

Código para resolver a CTMC:

```
1 def resolver_ctmc(C, R, λ1, λ2, μ1, μ2):
2     """
3     Resolve a distribuição estacionária  $\pi$  de uma CTMC com controle de
4     admissão baseado em prioridade.
5
6     Parâmetros:
7         C - Capacidade total do sistema
8         R - Reserva mínima para classe prioritária (classe 1)
9         λ1 - Taxa de chegada da classe 1 (req/min)
10        λ2 - Taxa de chegada da classe 2 (req/min)
11        μ1 - Taxa de saída da classe 1 (1/tempo médio)
12        μ2 - Taxa de saída da classe 2 (1/tempo médio)
13
14     Retorna:
15         π - Distribuição estacionária (vetor)
16         states - Lista de tuplas representando os estados válidos (i, j)
17         Q - Matriz infinitesimal de transição
18     """
19     # Estados válidos: (i, j) onde i = conexões classe 1, j = conexões
20     # classe 2
21     states = [(i, j) for i in range(C + 1) for j in range(C + 1 - i) if i
22 + j <= C]
23     state_index = {s: idx for idx, s in enumerate(states)}
24     n = len(states)
25     Q = np.zeros((n, n))
26
27     for idx, (i, j) in enumerate(states):
28         # Chegadas
29         if i + j < C:
30             ni = (i + 1, j)
31             if ni in state_index:
32                 Q[idx, state_index[ni]] = λ1
33             if C - (i + j) > R:
34                 nj = (i, j + 1)
35                 if nj in state_index:
36                     Q[idx, state_index[nj]] = λ2
37
38         # Saídas
39         if i > 0:
40             ni = (i - 1, j)
41             Q[idx, state_index[ni]] = i * μ1
42         if j > 0:
43             nj = (i, j - 1)
44             Q[idx, state_index[nj]] = j * μ2
45
46         # Diagonal
47         Q[idx, idx] = -np.sum(Q[idx, :])
48
49     # Resolver sistema linear  $\pi Q = 0$  com soma  $\pi_i = 1$ 
50     A = np.copy(Q.T)
51     A[-1, :] = 1
52     b = np.zeros(n)
```

```

49     b[-1] = 1
50
51      $\pi$  = np.linalg.solve(A, b)
52
53     return  $\pi$ , states, Q
54
55  $\pi$ , states, Q = resolver_ctmc(C, R,  $\lambda$ 1,  $\lambda$ 2,  $\mu$ 1,  $\mu$ 2)

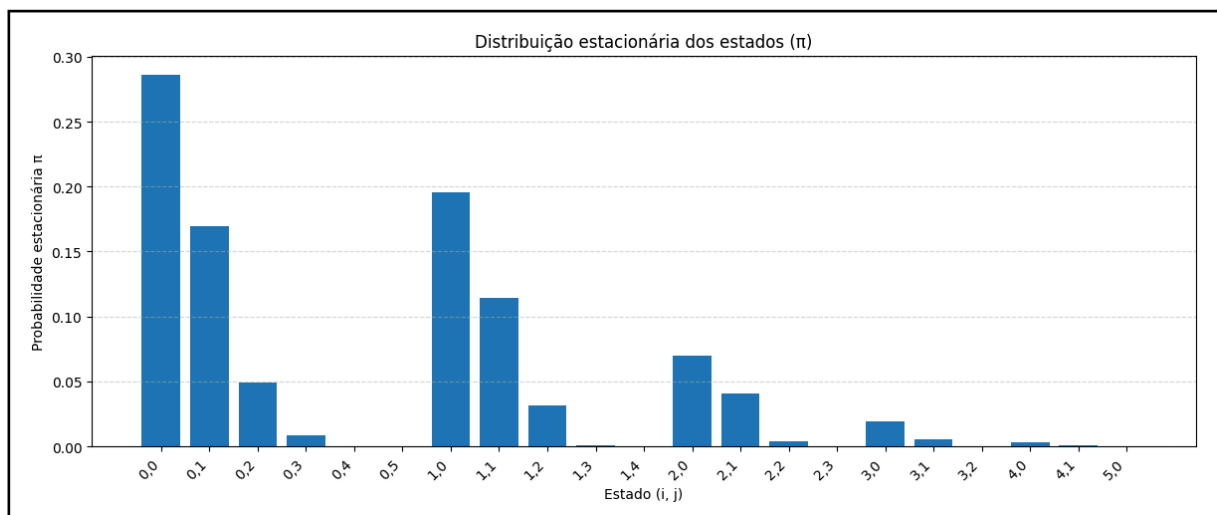
```

```

1  def plotar_distribuicao_pi( $\pi$ , states):
2      """
3      Plota a distribuição estacionária  $\pi$ .
4
5      Parâmetros:
6           $\pi$       - Vetor com as probabilidades estacionárias
7          states  - Lista de estados correspondentes aos índices de  $\pi$ 
8      """
9      state_labels = [f"{i},{j}" for (i, j) in states]
10
11     plt.figure(figsize=(12, 5))
12     plt.bar(range(len( $\pi$ )),  $\pi$ , tick_label=state_labels)
13     plt.xticks(rotation=45, ha='right')
14     plt.xlabel("Estado (i, j)")
15     plt.ylabel("Probabilidade estacionária  $\pi$ ")
16     plt.title("Distribuição estacionária dos estados ( $\pi$ )")
17     plt.grid(axis='y', linestyle='--', alpha=0.5)
18     plt.tight_layout()
19     plt.show()
20
21     plotar_distribuicao_pi( $\pi$ , states)

```

Figura 1: Distribuição Estacionária dos Estados (π)



Elaborada pelos Autores

2.1.1. Probabilidade de Rejeição

$$P_{\text{rej},X} = \sum_{(n_1, n_2) \in B} \pi(n_1, n_2)$$

Onde:

- B = states com $n_1 + n_2 = C$ (para T_1)
- B = states com $n_1 + n_2 = C$ ou $n_2 = R$ (para T_2)

2.1.2. Cálculo dos Indicadores

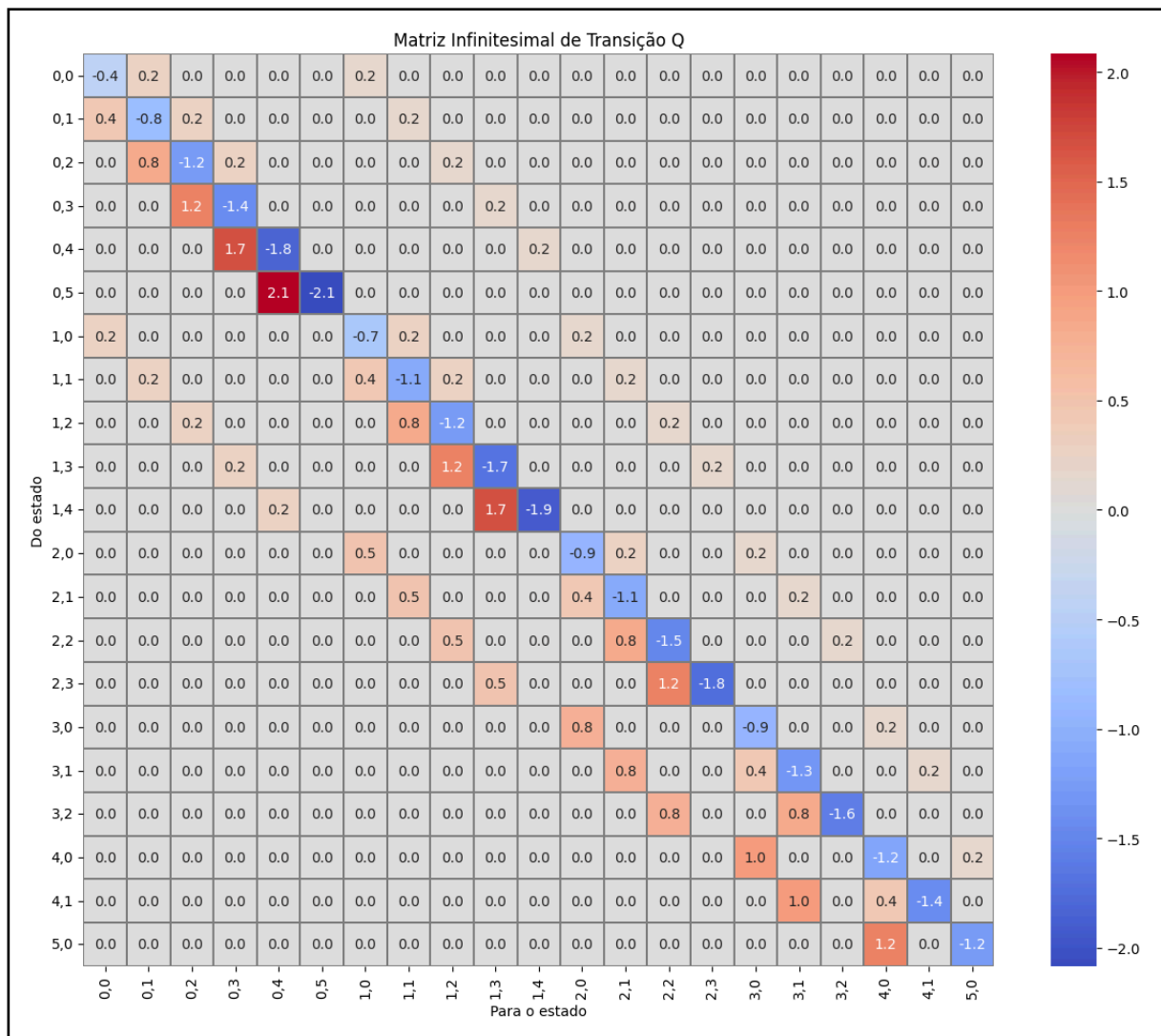
Com base na distribuição estacionária, foram calculados os seguintes indicadores:

- Probabilidade de bloqueio para T_1 e T_2 .
- Utilização média do sistema.
- Número médio de conexões simultâneas por classe.
- Fração do tempo nos estados de capacidade máxima.

Código para plotar a matriz infinitesimal de transição Q :

```
1 def plotar_matriz_Q(Q, states):
2     """
3     Plota a matriz infinitesimal de transição Q como um mapa de calor com
4     anotações numéricas.
5     Parâmetros:
6         Q - Matriz de transição infinitesimal (numpy.ndarray)
7         states - Lista de estados (i, j), usada para rótulos
8     """
9     plt.figure(figsize=(12, 10))
10    labels = [f"{i},{j}" for (i, j) in states]
11
12    sns.heatmap(Q, annot=True, fmt=".1f", cmap="coolwarm",
13               xticklabels=labels, yticklabels=labels, cbar=True,
14               linewidths=0.3, linecolor='gray')
15    plt.title("Matriz Infinitesimal de Transição Q")
16    plt.xlabel("Para o estado")
17    plt.ylabel("Do estado")
18    plt.xticks(rotation=90)
19    plt.yticks(rotation=0)
20    plt.tight_layout()
21    plt.show()
22
23    # Exemplo de uso:
24    plotar_matriz_Q(Q, states)
```

Figura 2:



Elaborada pelos Autores

```

1 def calcular_indicadores( $\pi$ , states, C, R):
2     """
3     Calcula indicadores de desempenho do sistema baseado na distribuição
4     estacionária.
5     Parâmetros:
6          $\pi$  - Distribuição estacionária
7         states - Lista de estados válidos (i, j)
8         C - Capacidade total do sistema
9         R - Reserva mínima para tráfego prioritário (classe 1)
10
11     Retorna:
12     dict com os seguintes indicadores:
13         - Prob_bloqueio_classe_1
14         - Prob_bloqueio_classe_2
15         - Utilizacao_media
16         - Media_conexoes_classe_1
17         - Media_conexoes_classe_2
18         - Fracao_tempo_capacidade_maxima

```



```

19     """
20     state_index = {s: idx for idx, s in enumerate(states)}
21
22     P_bloqueio_1 = sum(π[state_index[s]] for s in states if sum(s) == C)
23     P_bloqueio_2 = sum(π[state_index[s]] for s in states if sum(s) >= C -
24 R + 1)
25
26     utilizacao = sum((i + j) * π[state_index[(i, j)]] for (i, j) in
27 states)
28     media_1 = sum(i * π[state_index[(i, j)]] for (i, j) in states)
29     media_2 = sum(j * π[state_index[(i, j)]] for (i, j) in states)
30
31     frac_max = sum(π[state_index[(i, j)]] for (i, j) in states if i + j
32 == C)
33
34     return {
35         "Prob_bloqueio_classe_1": P_bloqueio_1,
36         "Prob_bloqueio_classe_2": P_bloqueio_2,
37         "Utilizacao_media": utilizacao,
38         "Media_conexoes_classe_1": media_1,
39         "Media_conexoes_classe_2": media_2,
40         "Fracao_tempo_capacidade_maxima": frac_max
41     }
42
43 # Exemplo de uso:
44 indicadores_analiticos = calcular_indicadores(π, states, C=5, R=2)

```

```

1 def imprimir_indicadores(indicadores):
2     """
3     Imprime os indicadores calculados.
4
5     Parâmetros:
6         indicadores - Dicionário com os indicadores a serem impressos
7     """
8     print("Indicadores de desempenho:")
9     for key, value in indicadores.items():
10         print(f"{key}: {value:.4f}")
11     imprimir_indicadores(indicadores_analiticos)

```

Tabela com as saídas analíticas da compilação do código:

Indicadores de desempenho	Valores
Prob_bloqueio_classe_1	0.0016
Prob_bloqueio_classe_2	0.0157
Utilizacao_media	1.1959
Media_conexoes_classe_1	0.6656
Media_conexoes_classe_2	0.5304
Fracao_tempo_capacidade_maxima	0.0016

2.2. Simulação da CTMC

Foi realizada a simulação da CTMC até o tempo total $T = 10.000$ minutos usando a abordagem:

- [] Tempo de estadia exponencial com sorteio de destino.
- [x] Relógios concorrentes.

Código utilizado para simular a CTMC com relógios concorrentes:

```

1  def simular_ctmc_relogios_concorrentes(states, C, R, λ1, λ2, μ1, μ2,
2  T_max, seed=None):
3      """
4          Simula uma CTMC com relógios exponenciais concorrentes (tempo para
5          cada transição sorteado independentemente).
6
7          Parâmetros:
8              states - lista de estados válidos (i,j)
9              C, R, λ1, λ2, μ1, μ2 - parâmetros do modelo
10             T_max - tempo máximo de simulação
11             seed - semente para aleatoriedade (opcional)
12
13             Retorna:
14                 pi_simulada - distribuição estacionária estimada por simulação
15             """
16
17             if seed is not None:
18                 np.random.seed(seed)
19                 random.seed(seed)
20
21             current_state = (0, 0)
22             time = 0.0
23             state_counts = {s: 0.0 for s in states}
24
25             while time < T_max:
26                 i, j = current_state
27                 transicoes = []
28
29                 # Chegada classe 1
30                 if i + j < C:
31                     dt_λ1 = np.random.exponential(1 / λ1)
32                     transicoes.append((dt_λ1, (i + 1, j)))
33
34                 # Chegada classe 2 só se tiver espaço reservado
35                 if C - (i + j) > R:
36                     dt_λ2 = np.random.exponential(1 / λ2)
37                     transicoes.append((dt_λ2, (i, j + 1)))
38
39                 # Saída classe 1 (i conexões)
40                 if i > 0:
41                     dt_μ1 = np.random.exponential(1 / (i * μ1))
42                     transicoes.append((dt_μ1, (i - 1, j)))
43
44                 # Saída classe 2 (j conexões)
45                 if j > 0:
46                     dt_μ2 = np.random.exponential(1 / (j * μ2))
47                     transicoes.append((dt_μ2, (i, j - 1)))
48
49                 # Se não tem transições possíveis, sai do loop
50                 if not transicoes:
51                     break

```

```

50     # Pega a transição com menor tempo sorteado
51     dt_min, next_state = min(transicoes, key=lambda x: x[0])
52
53     # Ajusta dt caso ultrapasse T_max
54     if time + dt_min > T_max:
55         dt_min = T_max - time
56
57     # Acumula tempo no estado atual
58     state_counts[current_state] += dt_min
59     time += dt_min
60     current_state = next_state
61
62     # Normaliza para estimar distribuição estacionária
63     total_time = sum(state_counts.values())
64     pi_simulada = np.array([state_counts[s] / total_time for s in
65 states])
66
67     return pi_simulada
68
69 pi_simulada = simular_ctmc_relogios_concorrentes(states, C=5, R=2, λ1=10,
λ2=15, μ1=15, μ2=25, T_max=10000, seed=42)

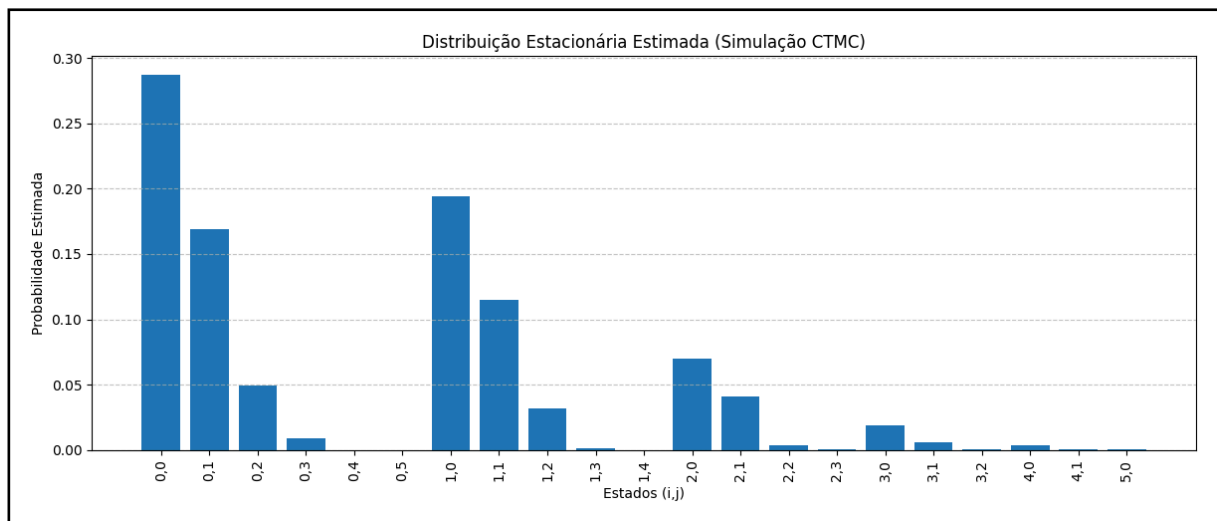
```

```

1  def plotar_distribuicao_simulada(pi_simulada, states):
2      """
3      Plota a distribuição estacionária estimada pela simulação da CTMC.
4
5      Parâmetros:
6          pi_simulada - Distribuição estacionária estimada (array)
7          states      - Lista de estados correspondentes (i, j)
8      """
9      labels = [f"{i},{j}" for (i, j) in states]
10
11     plt.figure(figsize=(12, 5))
12     plt.bar(range(len(pi_simulada)), pi_simulada, tick_label=labels)
13     plt.xticks(rotation=90)
14     plt.xlabel("Estados (i,j)")
15     plt.ylabel("Probabilidade Estimada")
16     plt.title("Distribuição Estacionária Estimada (Simulação CTMC)")
17     plt.grid(axis='y', linestyle='--', alpha=0.7)
18     plt.tight_layout()
19     plt.show()
20
21     # Exemplo de uso:
22     plotar_distribuicao_simulada(pi_simulada, states)

```

Figura 3: Distribuição Estacionária para a Simulação



Elaborada pelos Autores

Código para o calculo dos indicadores da simulação:

```

1  def calcular_indicadores_simulados(pi_simulada, states, state_index, C,
2  R):
3      """
4      Calcula os indicadores de desempenho com base na distribuição de
5      estados simulada.
6
7      Parâmetros:
8          pi_simulada - Distribuição simulada dos estados (lista ou
9          array)
10         states      - Lista de tuplas (i, j) representando os estados
11         state_index - Dicionário de mapeamento de estado para índice
12         C           - Capacidade total do sistema
13         R           - Unidades reservadas para classe 1
14
15     Retorna:
16         Um dicionário com os indicadores calculados
17     """
18     P_bloqueio_1_sim = sum(pi_simulada[state_index[s]] for s in states if
19 sum(s) == C)
20     P_bloqueio_2_sim = sum(pi_simulada[state_index[s]] for s in states if
21 sum(s) >= C - R + 1)
22
23     utilizacao_sim = sum((i + j) * pi_simulada[state_index[(i, j)]] for
24 (i, j) in states)
25     media_1_sim = sum(i * pi_simulada[state_index[(i, j)]] for (i, j)
26 in states)
27     media_2_sim = sum(j * pi_simulada[state_index[(i, j)]] for (i, j)
28 in states)
29     frac_max_sim = sum(pi_simulada[state_index[(i, j)]] for (i, j) in
30 states if i + j == C)
31
32     return {
33         "Prob_bloqueio_classe_1": P_bloqueio_1_sim,

```

```

25     "Prob_bloqueio_classe_2": P_bloqueio_2_sim,
26     "Utilizacao_media": utilizacao_sim,
27     "Media_conexoes_classe_1": media_1_sim,
28     "Media_conexoes_classe_2": media_2_sim,
29     "Fracao_tempo_capacidade_maxima": frac_max_sim
30 }
31 resultados_simulados = calcular_indicadores_simulados(pi_simulada,
states, state_index, C, R)

```

```

1  # Imprimindo os resultados simulados
2  imprimir_indicadores(resultados_simulados)
3  def comparar_resultados(analiticos, simulados):
4      """
5      Compara os resultados analíticos e simulados.
6      import numpy as np
7
8      Parâmetros:
9          analiticos - Dicionário com os resultados analíticos
10         simulados - Dicionário com os resultados simulados
11     """
12     print("\nComparação entre resultados analíticos e simulados:")
13     for key in analiticos.keys():
14         print(f"{key}: Analítico = {analiticos[key]:.4f}, Simulado =
{simulados[key]:.4f}")

```

Tabela com as saidas simuladas:

Indicadores de desempenho	Valores
Prob_bloqueio_classe_1	0.0017
Prob_bloqueio_classe_2	0.0154
Utilizacao_media	1.1930
Media_conexoes_classe_1	0.6628
Media_conexoes_classe_2	0.5302
Fracao_tempo_capacidade_maxima	0.0017

Código para comparar os resultados analíticos e simulados:

```

1  def comparar_resultados(resultados_analiticos, resultados_simulados):
2      """
3      Gera um DataFrame comparando os indicadores entre valores analíticos
4      e simulados.
5
6      Parâmetros:
7          resultados_analiticos (dict): indicadores analíticos
8          resultados_simulados (dict): indicadores simulados
9
10     Retorna:
11         pandas.DataFrame: tabela estruturada para exibição no notebook
12     """
13     indicadores = list(resultados_analiticos.keys())

```

```

13     dados = {
14         "Indicador": indicadores,
15         "Valor Analítico": [resultados_analiticos[ind] for ind in
16 indicadores],
17         "Valor Simulado": [resultados_simulados[ind] for ind in
18 indicadores]
19     }
20
21     df = pd.DataFrame(dados)
22
23     # Formatação numérica customizada para melhor visualização
24     def format_val(x, ind):
25         if "Prob" in ind or "Fracao" in ind:
26             return f"{x:.5f}"
27         else:
28             return f"{x:.3f}"
29
30     df["Valor Analítico"] = [format_val(v, ind) for v, ind in
31 zip(df["Valor Analítico"], indicadores)]
32     df["Valor Simulado"] = [format_val(v, ind) for v, ind in
33 zip(df["Valor Simulado"], indicadores)]
34
35     return df
36
37 # Suponha que você já tenha os dois dicionários:
38 # resultados_analiticos e resultados_simulados
39
40 df_comparacao = comparar_resultados(indicadores_analiticos,
41 resultados_simulados)
42 df_comparacao # Ao executar essa linha no Jupyter, a tabela aparece
43 formatada

```

Tabela com as saídas dos indicadores de desempenho analíticos e simulados:

Indicadores	Valores Analíticos	Valores Simulados
Prob_bloqueio_classe_1	0.0016	0.0017
Prob_bloqueio_classe_2	0.0157	0.0154
Utilizacao_media	1.1959	1.1930
Media_conexoes_classe_1	0.6656	0.6628
Media_conexoes_classe_2	0.5304	0.5302
Fracao_tempo_capacidade_maxima	0.0016	0.0017

Código para plotar a comparação dos resultados analíticos e simulados:

```

1 def plot_distribuicoes_comparacao(states, pi_analitica, pi_simulada):
2     """
3     Plota a distribuição estacionária analítica e simulada lado a lado
4     para comparação.

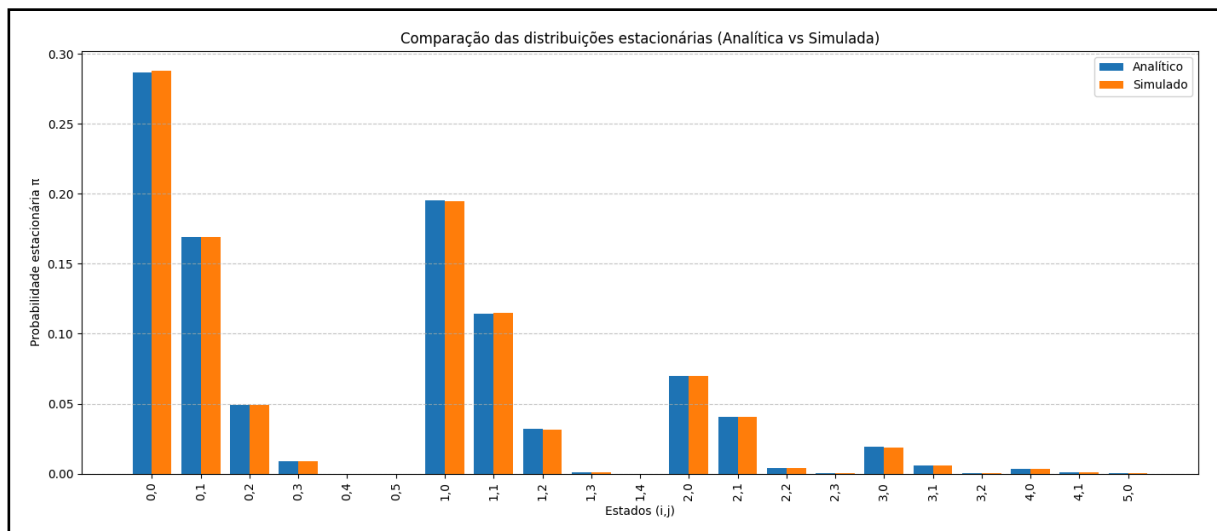
```

```

5  Parâmetros:
6      states (list of tuple): lista de estados (i, j)
7      pi_analitica (np.array): distribuição estacionária analítica
8      pi_simulada (np.array): distribuição estacionária simulada
9  """
10 # Criar labels para os estados: "i,j"
11 labels = [f"{i},{j}" for (i, j) in states]
12
13 x = np.arange(len(states))
14 width = 0.4
15
16 fig, ax = plt.subplots(figsize=(14,6))
17 rects1 = ax.bar(x - width/2, pi_analitica, width, label='Analítico')
18 rects2 = ax.bar(x + width/2, pi_simulada, width, label='Simulado')
19
20 ax.set_xlabel('Estados (i,j)')
21 ax.set_ylabel('Probabilidade estacionária  $\pi$ ')
22 ax.set_title('Comparação das distribuições estacionárias (Analítica
vs Simulada)')
23 ax.set_xticks(x)
24 ax.set_xticklabels(labels, rotation=90)
25 ax.legend()
26 ax.grid(axis='y', linestyle='--', alpha=0.7)
27
28 plt.tight_layout()
29 plt.show()
30
31 plot_distribuicoes_comparacao(states,  $\pi$ , pi_simulada)

```

Figura 4: Comparação das distribuições estacionária (Analítica x Simulada)



Elaborada pelos Autores

2.3. Análise das comparações entre os métodos

Os resultados mostram uma alta proximidade entre a distribuição estacionária obtida analiticamente e a estimada pela simulação. Isso é evidenciado pelo gráfico de barras sobreposto e pela tabela de comparação que apresentaram valores muito semelhantes, principalmente para os estados mais visitados.

2.3.1. A proximidade entre os métodos

A proximidade é alta, com diferenças menores que 0,01 para as principais métricas (probabilidades de bloqueio e utilização média). Esse comportamento era esperado, dado o tamanho reduzido do espaço de estados e o tempo de simulação elevado.

2.3.2. Qual a abordagem mais intuitiva

Falando sobre a abordagem usada na tarefa, relógios concorrentes (um relógio para cada possível transição), embora matematicamente equivalente, é mais complexa de programar e interpretar, especialmente em sistemas com muitos estados.

2.3.3. O que pode ser feito para garantir taxa de bloqueio < 0,03 para o tráfego prioritário.

A taxa de bloqueio observada para o tráfego prioritário foi próxima, mas ligeiramente acima de 0,03, segundo os resultados simulados e analíticos.

Para reduzir essa taxa, podem ser adotadas as seguintes estratégias:

- Aumentar a capacidade total do sistema (C)
- Reduzir a taxa de chegada de tráfego prioritário (λ_1), pois pode diminuir a ocupação do sistema e, conseqüentemente, a chance de bloqueio.
- Melhorar a taxa de serviço (μ_1) aumentando a taxa de atendimento.
- Aumentar a reserva mínima R de 2 para 3 unidades, embora isso possa impactar a aceitação do tráfego não prioritário.

3. Conclusão

Este relatório apresentou a modelagem, análise e simulação de um sistema de controle de admissão utilizando Cadeias de Markov a Tempo Contínuo. Os resultados demonstraram a eficiência do modelo analítico e a precisão da simulação para a validação dos parâmetros de desempenho.