Miniprojeto: Cadeias de Markov a Tempo Contínuo

Controle de Admissão com CTMC

Jéssica Gomes Carrico e Leonardo Ludvig Silva

IFSC – Engenharia de Telecomunicações

Maio de 2025

Contexto do Projeto

- Sistema com duas classes de tráfego: T1 (prioritário) e T2 (não prioritário).
- Cada sessão ocupa 1 unidade de capacidade.
- Capacidade total: $C = 5$
- Reserva mínima para T1: $R = 2$

Parâmetros do Sistema

- Chegada de T1: $\lambda_1 = 10$ req/min
- Chegada de T2: $\lambda_2 = 15$ req/min
- Atendimento T1: $\mu_1 = 15$ req/min
- Atendimento T2: $\mu_2 = 25$ req/min

Conjunto de Estados Válidos

- Estados da forma (n_1, n_2) com $n_1 + n_2 \leq C$ e $n_2 \leq C - R$

$$
\begin{aligned}
&(0,0),(0,1),(0,2),(0,3),\\
&(1,0),(1,1),(1,2),(1,3),\\
&(2,0),(2,1),(2,2),(2,3),\\
&(3,0),(3,1),(3,2),\\
&(4,0),(4,1),\\
&(5,0).
\end{aligned}
$$

Figura: Estados possíveis

Matriz de Transição Q

- $Q(i,j) =$
  - λ_1 se chegada T1 é válida
  - λ_2 se chegada T2 é válida
  - $n_1\mu_1$ se saída T1
  - $n_2\mu_2$ se saída T2
- Diagonal: $Q(i,i) = -\sum_{j\neq i} Q(i,j)$

Matriz de Transição Q

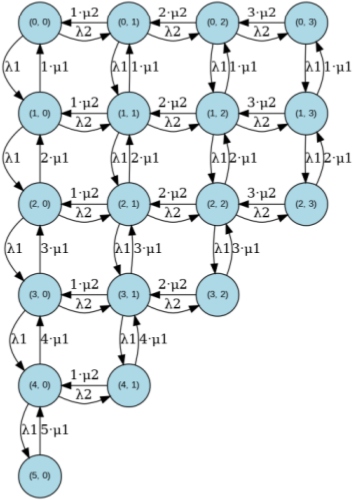


Figura: Matriz Q

Matriz de Transição Q - Código Parte 1

```
# -----------------------------
# Parmetros do sistema
# -----------------------------
lambda1 = 10 # chegada T1
lambda2 = 15 # chegada T2
mi1 = 15 # atendimento T1
mi2 = 25 # atendimento T2
C = 5 # capacidade total
R = 2 # reserva para T1
```

```
# -----------------------------
# Lista de estados vlidos
# -----------------------------
states = [
    (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)
]
```

Matriz de Transição Q - Código Parte 2

```
index_map = {state: i for i, state
     in enumerate(states)}
n = len(states)
Q = np.zeros((n, n))

for i, (n1, n2) in enumerate(
    states):
    # Entrada de T1
    next_state = (n1 + 1, n2)
    if next_state in index_map:
        Q[i][index_map[next_state]]
             = lambda1

    # Entrada de T2
    next_state = (n1, n2 + 1)
    if next_state in index_map:
        Q[i][index_map[next_state]]
             = lambda2
```

```
    # Sada de T1
    if n1 > 0:
        next_state = (n1 - 1, n2)
        Q[i][index_map[next_state]]
             = n1 * mi1

    # Sada de T2
    if n2 > 0:
        next_state = (n1, n2 - 1)
        Q[i][index_map[next_state]]
             = n2 * mi2
```

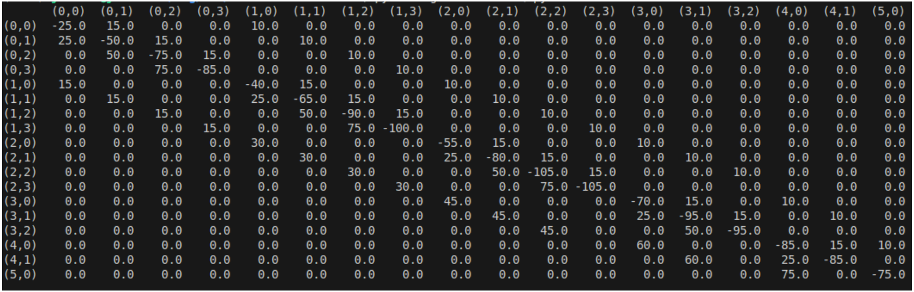
Matriz de Transição Q - Código Parte 3

```
    # Diagonal principal
    Q[i][i] = -np.sum(Q[i])

# -----------------------------
# Exibir matriz Q com pandas
# -----------------------------
state_labels = [f"({n1},{n2})" for
     (n1, n2) in states]
Q_df = pd.DataFrame(Q, index=
    state_labels, columns=
    state_labels)
```

```
# Mostrar matriz no console
print(Q_df.round(2))
```

Matriz de Transição Q Gerada



	(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)
(0,0)	-25.0	15.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(0,1)	25.0	-50.0	15.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(0,2)	0.0	50.0	-75.0	15.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(0,3)	0.0	0.0	75.0	-85.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(1,0)	15.0	0.0	0.0	0.0	-40.0	15.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(1,1)	0.0	15.0	0.0	0.0	25.0	-65.0	15.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(1,2)	0.0	0.0	15.0	0.0	0.0	50.0	-90.0	15.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(1,3)	0.0	0.0	0.0	15.0	0.0	0.0	75.0	-100.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0
(2,0)	0.0	0.0	0.0	0.0	30.0	0.0	0.0	0.0	-55.0	15.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0
(2,1)	0.0	0.0	0.0	0.0	0.0	30.0	0.0	0.0	25.0	-80.0	15.0	0.0	0.0	10.0	0.0	0.0	0.0	0.0
(2,2)	0.0	0.0	0.0	0.0	0.0	0.0	30.0	0.0	0.0	50.0	-105.0	15.0	0.0	0.0	10.0	0.0	0.0	0.0
(2,3)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	30.0	0.0	0.0	75.0	-105.0	0.0	0.0	0.0	0.0	0.0	0.0
(3,0)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	45.0	0.0	0.0	0.0	-70.0	15.0	0.0	10.0	0.0	0.0
(3,1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	45.0	0.0	0.0	25.0	-95.0	15.0	0.0	10.0	0.0
(3,2)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	45.0	0.0	0.0	50.0	-95.0	0.0	0.0	0.0
(4,0)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	60.0	0.0	0.0	-85.0	15.0	10.0
(4,1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	60.0	0.0	25.0	-85.0	0.0
(5,0)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	75.0	0.0	-75.0

Figura: Matriz Q gerada

Distribuição Estacionária π

- Sistema linear: $Q^T \cdot \pi^T = 0$
- Normalização: $\sum_i \pi_i = 1$
- Solução obtida com `scipy.linalg.null_space`

Distribuição Estacionária π - Código Parte 1

```
# -----------------------------
# Calcular distribuio estacionria
# -----------------------------
# Transpor Q para resolver Q   = 0
Q_t = Q.T
ns = null_space(Q_t)
```

```
# Normalizar  (soma = 1)
pi = ns[:, 0]
pi = pi / np.sum(pi)
```

Distribuição Estacionária π - Código Parte 2

```
# ----------------------------
# Imprimir resultados
# ----------------------------
print("\nDistribuio estacionria :")
for i, prob in enumerate(pi):
    print(f"[{i}] Estado {states[i]} = {prob:.6f}")
```

Distribuição Estacionária Gerada

```
📌 Distribuição estacionária π:
π[0]  Estado (0, 0) = 0.283094
π[1]  Estado (0, 1) = 0.169857
π[2]  Estado (0, 2) = 0.050957
π[3]  Estado (0, 3) = 0.010191
π[4]  Estado (1, 0) = 0.188730
π[5]  Estado (1, 1) = 0.113238
π[6]  Estado (1, 2) = 0.033971
π[7]  Estado (1, 3) = 0.006794
π[8]  Estado (2, 0) = 0.062910
π[9]  Estado (2, 1) = 0.037746
π[10]  Estado (2, 2) = 0.011324
π[11]  Estado (2, 3) = 0.002265
π[12]  Estado (3, 0) = 0.013980
π[13]  Estado (3, 1) = 0.008388
π[14]  Estado (3, 2) = 0.002516
π[15]  Estado (4, 0) = 0.002330
π[16]  Estado (4, 1) = 0.001398
π[17]  Estado (5, 0) = 0.000311
```

Figura: Distribuição Estacionária Gerada

Soma das Probabilidades de Rejeição

```
# Conjunto dos ndices que
    satisfazem n2=3
# ou n1+n2=5
indices_cond = [i for i, (n1, n2)
    in
              enumerate(states)
              if (n2 == 3) or (n1
                  + n2 == 5)]

# Soma das probabilidades  para
    esses estados
soma_prob = np.sum(pi[indices_cond
    ])
```

```
print(f"\nSoma das probabilidades
    para estados \
com n2=3 ou n1+n2=5: {soma_prob:.6
    f}")
```

```
Soma das probabilidades para estados com n2=3 ou n1+n2=5: 0.023475
```

Figura: Resultado de Probabilidade de bloqueio - Código

Indicadores de Desempenho

- Utilização média: $\sum_i (n_1 + n_2)\pi_i$
- Conexões médias T1: $\sum_i n_1\pi_i$
- Conexões médias T2: $\sum_i n_2\pi_i$
- Tempo em capacidade máxima: estados com $n_1 + n_2 = C$

Indicadores de Desempenho - Código

```
# ----------------------------
# Indicadores de desempenho
# ----------------------------
utilizacao_total = 0
conexoes_t1 = 0
conexoes_t2 = 0
tempo_cap_max = 0

for i, (n1, n2) in enumerate(
    states):
    prob = pi[i]
    utilizacao_total += (n1 + n2) *
        prob
    conexoes_t1 += n1 * prob
    conexoes_t2 += n2 * prob
    if n1 + n2 == C:
        tempo_cap_max += prob
```

```
print(f"\nUtilizao mdia do sistema
    : {utilizacao_total:.6f}")
print(f"Nmero mdio de conexes T1:
    {conexoes_t1:.6f}")
print(f"Nmero mdio de conexes T2:
    {conexoes_t2:.6f}")
print(f"Frao do tempo em
    capacidade mxima \
(n1+n2={C}): {tempo_cap_max:.6f}")
```

Simulação da CTMC

- Duração: 10.000 minutos
- Amostragem do tempo exponencial
- Escolha do próximo estado por sorteio com probabilidade proporcional
- Estimativa empírica $\hat{\pi}$ com tempo acumulado em cada estado

Simulação da CTMC - Parte 1

```
# ------------------------------
# Simulao da CTMC
# ------------------------------
T_total = 10000 # Tempo total de
    simulao (minutos)
tempo = 0.0
estado_atual = (0, 0)
index_atual = index_map[
    estado_atual]
tempo_estado = np.zeros(n)
```

Simulação da CTMC - Parte 2

```
while tempo < T_total:
    taxas = Q[index_atual].copy()
    taxas[index_atual] = 0 #
         Remover diagonal
    taxa_total = -Q[index_atual][
         index_atual]

    if taxa_total == 0:
        break # Estado absorvente (
             no  o caso aqui)

    # Tempo de permanncia ~
         exponencial(taxa_total)
    delta_t = np.random.exponential
         (scale=1/taxa_total)

    # Acumular tempo no estado
         atual
    tempo_estado[index_atual] +=
         delta_t
```

```
    # Probabilidades normalizadas
         de transio
    probs = taxas / taxa_total
    proximo_index = np.random.
         choice(n, p=probs)

    # Atualizar o estado
    index_atual = proximo_index
```

Simulação da CTMC - Parte 3

```
# ----------------------------
# Estimar distribuio
# ----------------------------
pi_hat = tempo_estado / np.sum(
    tempo_estado)

print("\nEstimativa de  por
    simulao ():")
for i, prob in enumerate(pi_hat):
    print(f"[{i}] Estado {states[i
        ]} = {prob:.6f}")
```

Resultado Simulação

```
📌 Estimativa de π por simulação (π̂):
π[0]  Estado (0, 0) = 0.283210
π[1]  Estado (0, 1) = 0.170210
π[2]  Estado (0, 2) = 0.050517
π[3]  Estado (0, 3) = 0.010056
π[4]  Estado (1, 0) = 0.188008
π[5]  Estado (1, 1) = 0.112526
π[6]  Estado (1, 2) = 0.033802
π[7]  Estado (1, 3) = 0.006657
π[8]  Estado (2, 0) = 0.064341
π[9]  Estado (2, 1) = 0.037894
π[10]  Estado (2, 2) = 0.011382
π[11]  Estado (2, 3) = 0.002308
π[12]  Estado (3, 0) = 0.014181
π[13]  Estado (3, 1) = 0.008466
π[14]  Estado (3, 2) = 0.002562
π[15]  Estado (4, 0) = 0.002184
π[16]  Estado (4, 1) = 0.001400
π[17]  Estado (5, 0) = 0.000295
```

Figura: Distribuição estácionaria na simulação

Comparação π vs $\hat{\pi}$

- Erro absoluto médio entre π e $\hat{\pi}$
- Comparação por estado mostra a precisão da simulação

Comparação entre (analítico) e (simulado)

```
# -----------------------------
# Comparar  (analtico) vs  (
    simulado)
# -----------------------------
erro_abs = np.abs(pi - pi_hat)
erro_medio = np.mean(erro_abs)

print(f"\nErro absoluto mdio entre
      e : {erro_medio:.6f}")
```

```
# Mostrar diferenas por estado
print("\nDiferena entre  e  por
    estado:")
for i, (p_analitico, p_simulado)
    in enumerate(zip(pi, pi_hat)):
   print(f"Estado {states[i]}: | -
        | = {abs(p_analitico -
        p_simulado):.6f}")
```

Comparação entre (analítico) e (simulado)

```
Erro absoluto médio entre π e π̂: 0.000275

Diferença entre π e π̂ por estado:
Estado (0, 0): |π - π̂| = 0.000116
Estado (0, 1): |π - π̂| = 0.000353
Estado (0, 2): |π - π̂| = 0.000440
Estado (0, 3): |π - π̂| = 0.000135
Estado (1, 0): |π - π̂| = 0.000721
Estado (1, 1): |π - π̂| = 0.000712
Estado (1, 2): |π - π̂| = 0.000169
Estado (1, 3): |π - π̂| = 0.000137
Estado (2, 0): |π - π̂| = 0.001431
Estado (2, 1): |π - π̂| = 0.000148
Estado (2, 2): |π - π̂| = 0.000058
Estado (2, 3): |π - π̂| = 0.000044
Estado (3, 0): |π - π̂| = 0.000201
Estado (3, 1): |π - π̂| = 0.000078
Estado (3, 2): |π - π̂| = 0.000045
Estado (4, 0): |π - π̂| = 0.000146
Estado (4, 1): |π - π̂| = 0.000002
Estado (5, 0): |π - π̂| = 0.000015
```

Figura: Diferença entre e por estado:

Discussões Finais

- Simulação confirma resultados teóricos
- Possível reduzir rejeição de T1:
  - Aumentar C
  - Aumentar R
  - Reduzir λ_1