UNIVERSIDADE FEDERAL DE MINAS GERAIS

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Daniel Terra Gomes

Relatório da Lista de Exercícios 1: Processos de Decisão de Markov e Programação Dinâmica

Belo Horizonte, MG

11 de setembro de 2025

"Behind me lies a farm.
I wonder if there is bread above the hearth
and if I will ever return."
(Pantheon, League of Legends)

Lista de ilustrações

Lista de trechos de código

Sumário

1 Implementação e Análise de Algoritmos de Programação Dinâmica

Este relatório detalha a implementação e análise dos algoritmos de **Iteração de Política** (Policy Iteration) e **Iteração de Valor** (Value Iteration), aplicados ao ambiente *Frozen Lake*. O objetivo é demonstrar a aplicação prática dos conceitos teóricos de Processos de Decisão de Markov (MDPs) e Programação Dinâmica, validando os resultados obtidos através de uma análise comparativa entre ambientes determinísticos e estocásticos.

1.1 O Ambiente Frozen Lake

O *Frozen Lake* é um ambiente clássico de grade (gridworld) onde um agente deve navegar de um estado inicial (S) a um estado objetivo (G) sobre um lago congelado. O caminho pode conter buracos (H), que terminam o episódio com falha, conforme ilustrado na Figura 1.

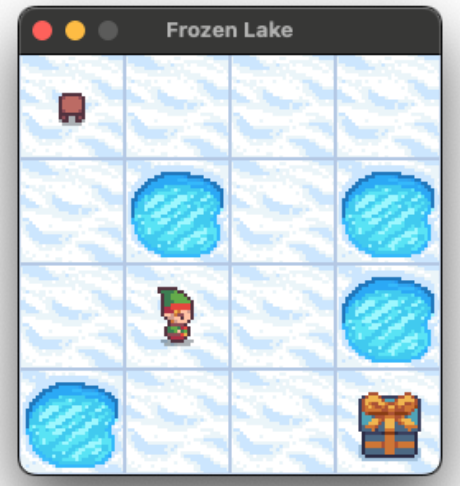

Figura 1 – Ambiente Frozen Lake.

A principal característica explorada neste trabalho é a propriedade `is_slippery`, que define a natureza do ambiente como:

- **Determinístico (`is_slippery=False`):** as ações do agente têm um resultado certo e previsível. A política ótima tende a ser o caminho mais curto até o objetivo.

- **Estocástico (`is_slippery=True`):** há incerteza no resultado das ações. Uma ação tem apenas 1/3 de chance de sucesso, com 2/3 de chance de resultar em um movimento perpendicular. Isso força o agente a adotar uma política mais cautelosa, levando o risco em consideração.

As Figuras 3 e 2 ilustram a execução de uma política em ambos os cenários. No ambiente determinístico (esquerda), o agente segue um caminho direto. No estocástico (direita), a trajetória é mais errática devido aos "escorregões".

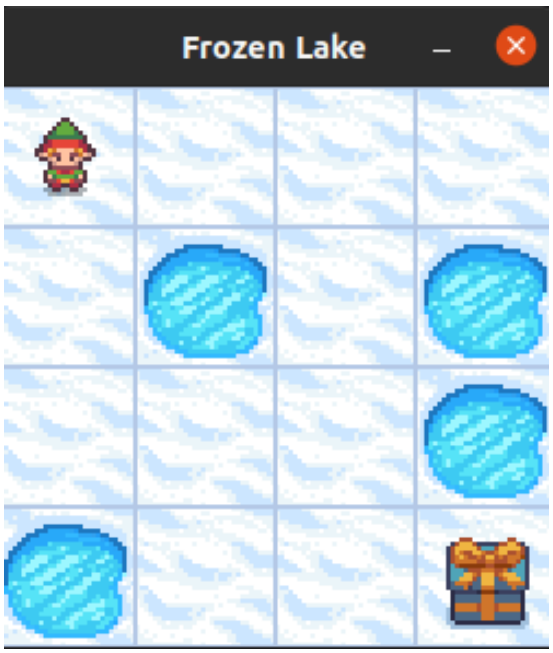

Figura 2 – Execução em ambiente determinístico.

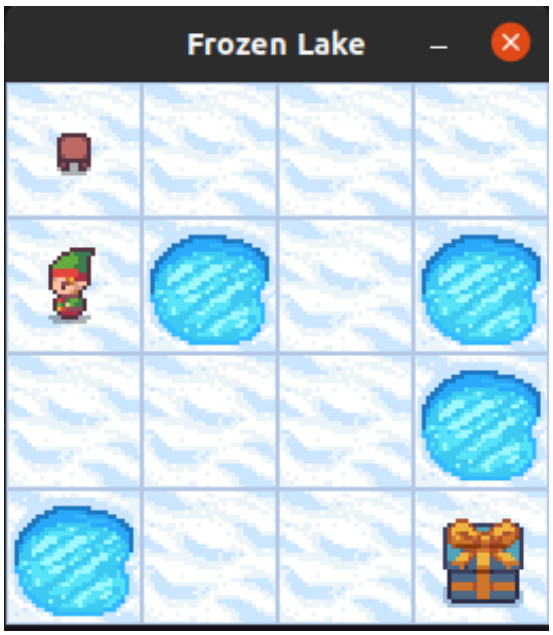

Figura 3 – Execução em ambiente estocástico.

1.2 Iteração de Política (Policy Iteration)

Nesta seção, veremos o processo para implementação da iteração de política.

1.2.1 Fundamentação Teórica

A Iteração de Política é um algoritmo de Programação Dinâmica que encontra a política ótima (π_*) alternando iterativamente entre dois processos: Avaliação de Política (Policy Evaluation) e Melhora da Política (Policy Improvement), conforme ilustrado na Figura 4.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(\textit{terminal}) \doteq 0$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Figura 4 – Pseudocódigo do algoritmo de Iteração de Política, conforme apresentado em aula.

1.2.2 Implementação

A implementação seguiu a estrutura teórica, dividindo o algoritmo em suas duas funções principais.

1.2.2.1 Avaliação de Política

A função `evaluate_policy` (Vista no Trecho de Código 1.1) implementa o passo de Avaliação. Ela calcula a função de valor (v_π) para uma política fixa π, aplicando iterativamente a equação de Bellman de expectativa até a convergência, conforme a Equação 1.1.

$$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')] \tag{1.1}$$

```
1  def evaluate_policy(env, policy, V, gamma, theta):
2      while True:
3          delta = 0
4          for state in range(env.observation_space.n):
5              old_v = V[state]
6              action = policy[state]
7              V[state] = compute_expected_value(env, V, state, action,
   gamma)
8              delta = max(delta, abs(old_v - V[state]))
9          if delta < theta:
10             break
```

Lista de código 1.1 – Implementação da Avaliação de Política.

1.2.2.2 Melhora da Política

A função `improve_policy` (Vista no Trecho de Código 1.2) implementa a melhora da política. Para cada estado, ela age de forma gulosa (*greedy*) em relação à função de valor calculada, selecionando a ação que maximiza o retorno esperado, conforme a Equação 1.2.

$$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \tag{1.2}$$

```
def improve_policy(env, policy, V, gamma):
    policy_stable = True
    for state in range(env.observation_space.n):
        old_action = policy[state]
        action_values = [compute_expected_value(env, V, state, action,
   gamma)
                         for action in range(env.action_space.n)]
        policy[state] = np.argmax(action_values)
        if old_action != policy[state]:
            policy_stable = False
    return policy_stable
```

Lista de código 1.2 – Implementação da Melhora da Política.

1.3 Iteração de Valor (Value Iteration)

Nesta seção, veremos o processo para implementação da Iteração de Valor.

1.3.1 Fundamentação Teórica

A Iteração de Valor é um algoritmo mais direto que combina os passos de avaliação e melhora em uma única atualização. Ele aplica diretamente a equação de Bellman para encontrar a função de valor ótima (v_*), como visto na Figura 5.

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Figura 5 – Pseudocódigo do algoritmo de Iteração de Valor, conforme apresentado em aula.

1.3.2 Implementação

A função `value_iteration` (Vista no Trecho de Código 1.3) implementa este processo. Em cada iteração, o valor de cada estado é atualizado com o valor máximo que pode ser obtido ao tomar a melhor ação possível a partir daquele estado.

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \tag{1.3}$$

Após a convergência de V, a política ótima é extraída de forma gulosa.

```
1  def value_iteration(env, gamma, theta):
2      V = init_value_iteration(env)
3      while True:
4          delta = 0
5          for state in range(env.observation_space.n):
6              old_v = V[state]
7              action_values = [compute_expected_value(env, V, state,
       action, gamma)
8                               for action in range(env.action_space.n)]
9              V[state] = np.max(action_values)
10             delta = max(delta, abs(old_v - V[state]))
11         if delta < theta:
12             break
13     policy = generate_policy(env, V, gamma)
14     return V, policy
```

Lista de código 1.3 – Implementação da Iteração de Valor.

1.4 Resultados e Análise Comparativa

Nesta seção, apresentaremos os resultados obtidos durante a simulação e as respostas para as perguntas do Notebook `ps1.ipynb`.

1.4.1 Análise da Função de Valor Ótima (v_*)

A Figura 6 compara as funções de valor ótimas (v_*) para os ambientes determinístico e estocástico.

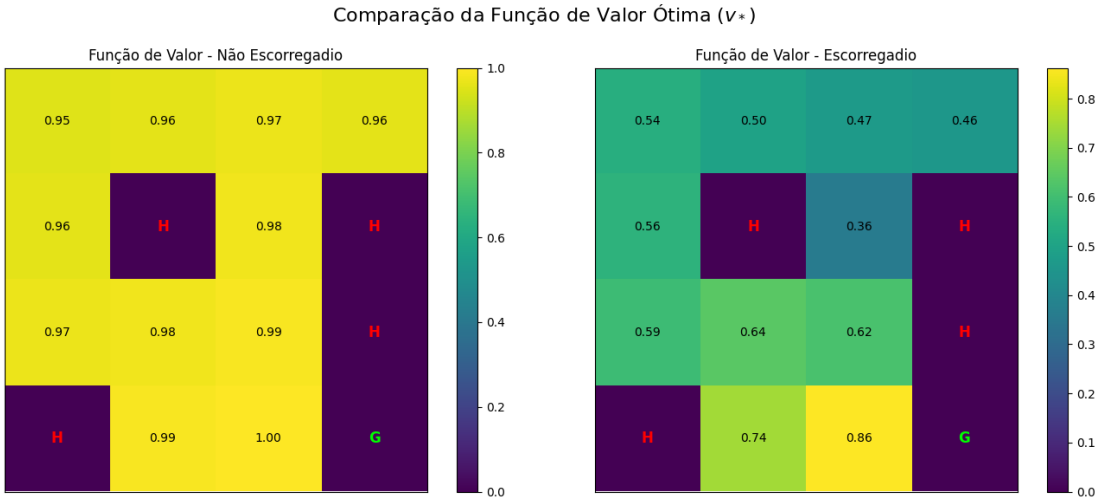

Figura 6 – Comparação da Função de Valor Ótima (v_*). No cenário determinístico (esquerda), os valores são altos, refletindo a certeza de alcançar o objetivo. No cenário estocástico (direita), os valores são significativamente menores devido à incerteza e ao risco de falha.

No ambiente **Não Escorregadio**, os valores são altos e decrescem suavemente a partir do objetivo (G), refletindo o desconto de $\gamma = 0.99$ a cada passo do caminho mais curto. Já no ambiente **Escorregadio**, os valores são muito menores. A incerteza inerente ao ambiente diminui drasticamente o retorno esperado de cada estado, pois sempre há uma probabilidade de escorregar para um buraco ou para um estado menos vantajoso.

1.4.2 Validação Cruzada: v_* vs. v_{π_*}

A Figura 7 valida a implementação ao comparar o desempenho de diferentes políticas no ambiente escorregadio.

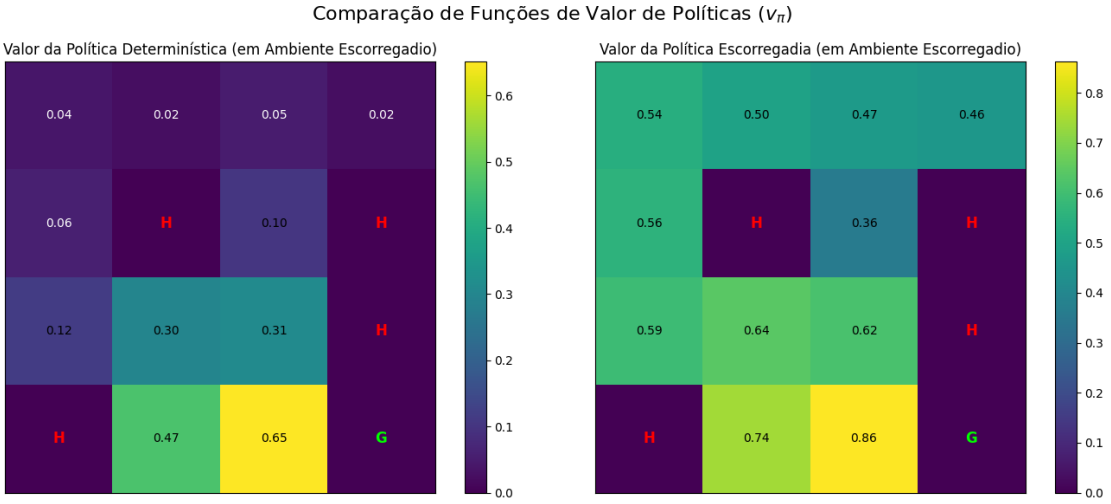

Figura 7 – Comparação de Funções de Valor de Políticas (v_π) no ambiente escorregadio. A política determinística (esquerda) tem um desempenho muito baixo, enquanto a política ótima para o ambiente escorregadio (direita) atinge a função de valor ótima.

O gráfico da esquerda mostra o valor de se executar a política "ingênua"(ótima para o mundo determinístico) no ambiente escorregadio. Como esperado, os valores são baixíssimos, pois essa política não considera o risco e leva o agente a falhar frequentemente.

O gráfico da direita mostra o valor da política ótima encontrada para o ambiente escorregadio. É crucial notar que este gráfico é **idêntico** ao gráfico da direita da Figura 6. Isso valida a teoria de que a função de valor da política ótima (v_{π_*}) é a função de valor ótima (v_*), confirmando a correta implementação de ambas as soluções.

1.4.3 Análise de Desempenho dos Agentes

Finalmente, a Figura 8 compara o desempenho médio de 10 episódios para dois agentes no ambiente escorregadio.

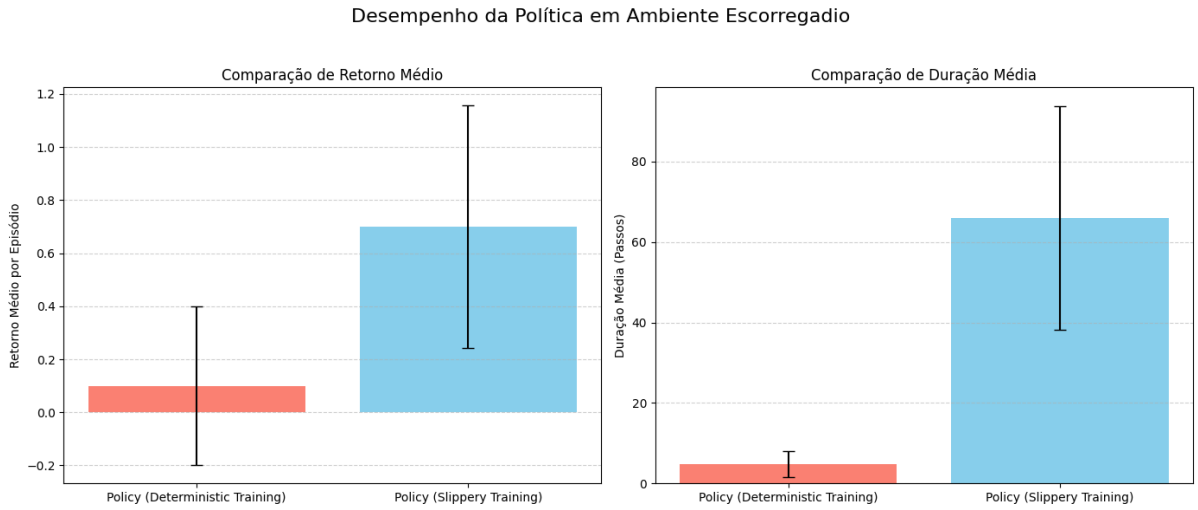

Figura 8 – Comparação de desempenho entre o Agente 1 (política determinística) e o Agente 2 (política estocástica) no ambiente escorregadio.

- **Agente 1 (Política Determinística):** tem um retorno médio próximo de zero e episódios curtos. Isso ocorre porque sua política ingênua o leva a cair rapidamente nos buracos na maioria das vezes.
- **Agente 2 (Política Estocástica):** atinge um retorno médio significativamente maior (cerca de 0,7), indicando que consegue chegar ao objetivo com frequência. No entanto, a alta variância (barra de erro) e a maior duração média dos episódios mostram que os caminhos para o sucesso são mais longos e menos diretos, refletindo sua estratégia cautelosa para evitar os riscos do ambiente.

1.4.4 Perguntas

1.4.4.1 15. Explique quais fatores levaram às diferenças observadas entre as políticas obtidas no ambiente determinístico e no ambiente escorregadio.

1. **Natureza do Ambiente:** Determinismo vs. Estocasticidade O fator principal é a propriedade escorregadia (is_slippery) do ambiente.

   Ambiente Determinístico não escorregadio (is_slippery=False): neste cenário, a ação escolhida pelo agente leva ao resultado esperado com 100% de certeza. Se o agente decide ir para a "direita", ele se move para a direita. A política ótima, portanto, traça o caminho mais curto e direto para o objetivo, evitando apenas os buracos de forma precisa. Não há necessidade de se preocupar com movimentos não intencionais, conforme podemos identificar no gráfico da Figura 6.

   Ambiente Escorregadio (is_slippery=True): nesse caso, o ambiente é estocástico. Uma ação escolhida tem apenas uma chance de levar ao resultado pretendido, com

as demais possibilidades sendo movimentos perpendiculares. Isso introduz um risco significativo: uma ação aparentemente segura pode levar o agente a um buraco.

2. **Estratégia de Mitigação de Risco:** devido à incerteza, a política ótima no ambiente escorregadio muda de uma estratégia de "caminho mais curto"para uma de mitigação de risco. Na Política Determinística, a política gerada é "gananciosa"e otimista. Ela assume que o controle do agente sobre o ambiente é absoluto e, por isso, se aproxima das bordas e dos buracos sem hesitação para encurtar o caminho. Enquanto, na Política Estocástica, a política gerada é visivelmente mais conservadora. Ela aprende que ações executadas perto de buracos têm uma alta probabilidade de falha, mesmo que a ação "correta"aponte para longe do perigo. Por exemplo, no estado 5 (segunda linha, segunda coluna), a política determinística pode seguramente ir para a direita. No entanto, na versão escorregadia, essa ação pode resultar em um movimento para baixo, levando ao buraco no estado 9. A política ótima, então, aprende a evitar essas zonas de risco, preferindo caminhos mais longos, porém mais seguros, que mantêm o agente longe das "bordas"dos buracos e até mesmo se chocando contra a parede para se aproveitar do ambiente escorregadio e escorregar para o estado desejado, conforme identifico pelos gráficos da Figura 7 e visto na Figura 3.

3. **Impacto na Função de Valor:** os algoritmos de Policy e Value Iteration buscam maximizar o retorno esperado, conforme vimos nas seções 1.2 e 1.3. No ambiente determinístico, o valor de um estado é alto se ele estiver em um caminho curto para a recompensa. Enquanto que no ambiente estocástico, o valor de um estado (V(s)) passa a ser descontado não apenas pelo fator gamma, mas também pela probabilidade de falha. Estados adjacentes a buracos terão valores inerentemente mais baixos, pois o retorno esperado a partir deles é menor devido à chance de cair e receber recompensa zero. O algoritmo então favorece ações que levam a estados com valores esperados mais altos e mais seguros.

1.4.4.2 16. Quais estratégias poderiam ser adotadas para tornar o comportamento do agente menos conservador quando treinado no ambiente escorregadio?

Para induzir um comportamento menos conservador (ou seja, que aceite mais riscos em troca de caminhos potencialmente mais curtos), não estamos buscando um algoritmo "melhor", mas sim alterando a definição do problema ou o objetivo do agente.

Estratégias para Reduzir o Comportamento Conservador:

1. **Modificação da Função de Recompensa (Reward Shaping):** a principal razão para o conservadorismo é que o custo de cair em um buraco (fim do episódio, recompensa zero) é muito maior do que o benefício de economizar alguns passos. Podemos incentivar caminhos mais curtos introduzindo uma pequena penalidade a

cada passo. Em nosso problema, em vez de uma recompensa de 0 para cada transição, poderíamos usar uma recompensa de -0,01, por exemplo. Para usar um exemplo dado em aula, seria como um chão quente, e o agente precisaria encontrar o caminho mais curto para não queimar os pés; uma pressão para que o agente chegue ao objetivo o mais rápido possível para minimizar a penalidade acumulada, forçando-o a considerar caminhos mais curtos e, consequentemente, mais arriscados.

2. **Ajuste do Fator de Desconto γ (Gamma):** o parâmetro γ (fator de desconto) determina a importância de recompensas futuras no cálculo do ganho total. Um γ próximo de 1 torna o agente "cauteloso"(com visão de longo prazo), pois as recompensas distantes são pouco descontadas, levando-o a valorizar o ganho cumulativo total. Por outro lado, um γ menor o tornaria mais "impaciente"e "míope", focando em ganhos de curto prazo, já que as recompensas futuras perdem seu valor rapidamente.

2 Conclusão

Este trabalho demonstrou com sucesso a implementação dos algoritmos de Iteração de Política e Iteração de Valor para resolver o ambiente *Frozen Lake*. A análise comparativa entre os cenários determinístico e estocástico evidenciou como a incerteza ambiental impacta fundamentalmente a política ótima e a função de valor. Os resultados práticos obtidos alinharam-se perfeitamente com a teoria da Programação Dinâmica, validando as implementações e proporcionando uma compreensão aprofundada dos conceitos de MDPs.