Instituto Tecnológico de Aeronáutica - ITA
CT-213 - Inteligência Artificial aplicada à Robótica Móvel
Aluno: Ulisses Lopes da Silva

Relatório do Laboratório 10 - Programação Dinâmica

---

1 Breve Explicação em Alto Nível da Implementação

O objetivo deste laboratório foi implementar e analisar algoritmos clássicos de *programação dinâmica* para a resolução de *Processos Decisórios de Markov* (MDP), especificamente aplicados a um ambiente do tipo *Grid World*. Esses algoritmos — *avaliação de política* (*policy evaluation*), *iteração de política* (*policy iteration*) e *iteração de valor* (*value iteration*) — são fundamentais no campo de *Aprendizado por Reforço* (*Reinforcement Learning*) e permitem encontrar políticas ótimas quando se conhece completamente o modelo probabilístico do ambiente, ou seja, as funções de transição e recompensa do MDP.

No problema proposto, o *Grid World* é um tabuleiro bidimensional composto por células que podem ser livres, obstáculos ou o estado objetivo. O agente inicia em uma célula qualquer e pode executar uma dentre cinco ações possíveis: `STOP`, `UP`, `RIGHT`, `DOWN` e `LEFT`. A ação `STOP` é executada com certeza (probabilidade 1), enquanto as demais têm uma probabilidade de acerto p_c e, caso falhem, resultam em qualquer outra ação com igual probabilidade. A recompensa atribuída a cada movimento é -1 para estados não objetivos, e 0 ao atingir a célula objetivo. O objetivo do agente é, portanto, alcançar o estado objetivo com o menor custo acumulado possível, considerando o fator de desconto γ.

A função `policy_evaluation` foi implementada de forma síncrona e baseia-se na equação de Bellman de expectativa. Para cada estado do ambiente, calcula-se iterativamente o valor esperado de seguir a política atual. A recompensa $r(s, a)$ é somada uma única vez por ação, garantindo aderência correta à formulação teórica. Estados inválidos (obstáculos ou o estado objetivo) são ignorados durante a atualização. O critério de parada utilizado envolve um número máximo de iterações e um limiar de convergência definido pela variação máxima entre os valores sucessivos dos estados.

A função `value_iteration` segue a equação de otimalidade de Bellman. Ao invés de fixar uma política, essa abordagem busca diretamente a função valor ótima, assumindo que o agente toma sempre a melhor ação disponível. Para cada estado, calcula-se o valor máximo dentre todas as ações possíveis, considerando a soma da recompensa imediata e o valor esperado descontado dos estados sucessores. Após a convergência da função valor, a política ótima pode ser obtida de maneira gulosa a partir dessa função, utilizando a função auxiliar `greedy_policy`.

A função `policy_iteration`, por sua vez, alterna entre etapas de avaliação parcial da política e sua melhoria. Durante a avaliação, são realizadas algumas iterações de `policy_evaluation`,

sem necessidade de convergência total. Em seguida, a política é atualizada para uma versão gulosa com respeito à função valor corrente. Esse ciclo se repete até que a política deixe de sofrer alterações significativas. Para comparar políticas sucessivas, foi utilizado o critério `np.allclose`, o que oferece robustez numérica ao processo.

A função `greedy_policy` constrói uma política estocástica gulosa com base na função valor atual. Caso existam múltiplas ações ótimas com valores similares (dentro de uma tolerância ϵ), todas são consideradas com igual probabilidade, evitando viés na escolha da ação.

Dessa forma, os algoritmos desenvolvidos permitem resolver eficientemente o problema do agente no *Grid World*, sendo capazes de encontrar tanto a função valor quanto a política ótima associada, garantindo desempenho ótimo esperado em ambientes totalmente observáveis e estocásticos.

1.0.1 Avaliação de Política

Dado uma política fixa $\pi(a|s)$, este algoritmo calcula iterativamente a função valor de estado $v_\pi(s)$ até a convergência, utilizando a equação de expectativa de Bellman:

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v_\pi(s') \right)$$

A função `policy_evaluation` implementa a equação de expectativa de Bellman. Ela realiza varreduras síncronas sobre todos os estados da grade, atualizando o valor de cada estado com base nos valores da iteração anterior. O processo se repete até que a máxima mudança no valor de qualquer estado (Δ) caia abaixo de um limiar de tolerância ϵ.

1.0.2 Iteração de Valor

Este algoritmo calcula diretamente a função valor ótima $v_*(s)$ sem a necessidade de uma política explícita. Ele aplica iterativamente a equação de otimalidade de Bellman:

$$v_{k+1}(s) = \max_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v_k(s') \right)$$

Após a convergência de $v_k(s)$ para $v_*(s)$, a política ótima $\pi_*(s)$ é extraída de forma gulosa (*greedy*).

A função `value_iteration` segue uma estrutura muito semelhante à do algoritmo anterior, mas, em vez de calcular a média ponderada pela política, aplica o operador de maximização sobre os valores-ação (Q-values) para cada estado, convergindo para a função valor ótima.

1.0.3 Iteração de Política

Este algoritmo alterna entre duas fases: avaliação e melhoria da política. Ele começa com uma política inicial, avalia seu valor (geralmente de forma parcial) e, em seguida, melhora a política

tornando-a gulosa em relação à função valor recém-calculada. A política de melhoria π' é obtida por:

$$\pi'(s) = \arg\max_{a \in A} \left(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v_\pi(s') \right)$$

Este processo é repetido até que a política não mude mais, garantindo a convergência para a política ótima π_*.

A função `policy_iteration` foi implementada de modo que, a cada iteração principal, ela chama a função `policy_evaluation` por um número fixo e predeterminado de vezes (`evaluations_per_policy`) para obter uma estimativa do valor da política atual. Em seguida, a função `greedy_policy` é usada para extrair uma nova política melhorada. A convergência do algoritmo é verificada comparando a política antiga com a nova através da função `numpy.allclose`, que é um método para comparações de arrays de ponto flutuante.

2 Tabelas Comprovando Funcionamento do Código

2.1 Caso $p_c = 1,0$ e $\gamma = 1,0$

Avaliação de Política

Função valor:

```
[ -384.09, -382.73, -381.19,    *    , -339.93, -339.93]
[ -380.45, -377.91, -374.65,    *    , -334.92, -334.93]
[ -374.34, -368.82, -359.85, -344.88, -324.92, -324.93]
[ -368.76, -358.18, -346.03,    *    , -289.95, -309.94]
[    *   , -344.12, -315.05, -250.02, -229.99,    *   ]
[ -359.12, -354.12,    *   , -200.01, -145.00,    0.00]
```

Política:

```
[ SURDL , SURDL , SURDL ,    *    , SURDL , SURDL ]
[ SURDL , SURDL , SURDL ,    *    , SURDL , SURDL ]
[ SURDL , SURDL , SURDL , SURDL , SURDL , SURDL ]
[ SURDL , SURDL , SURDL ,    *    , SURDL , SURDL ]
[    *  , SURDL , SURDL , SURDL , SURDL ,    *   ]
[ SURDL , SURDL ,    *  , SURDL , SURDL ,    S   ]
```

Iteração de Valor

Função valor:

```
[  -10.00,   -9.00,   -8.00,    *    ,   -6.00,   -7.00]
[   -9.00,   -8.00,   -7.00,    *    ,   -5.00,   -6.00]
[   -8.00,   -7.00,   -6.00,   -5.00,   -4.00,   -5.00]
[   -7.00,   -6.00,   -5.00,    *    ,   -3.00,   -4.00]
```

```
[    *    ,    -5.00,    -4.00,    -3.00,    -2.00,    *    ]
[    -7.00,    -6.00,    *    ,    -2.00,    -1.00,     0.00]
```

Política:

```
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    RD   ,    R    ,    D    ,    DL   ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    RD   ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,   SURD  ]
```

Iteração de Política

Função valor:

```
[   -10.00,    -9.00,    -8.00,    *    ,    -6.00,    -7.00]
[    -9.00,    -8.00,    -7.00,    *    ,    -5.00,    -6.00]
[    -8.00,    -7.00,    -6.00,    -5.00,    -4.00,    -5.00]
[    -7.00,    -6.00,    -5.00,    *    ,    -3.00,    -4.00]
[    *    ,    -5.00,    -4.00,    -3.00,    -2.00,    *    ]
[    -7.00,    -6.00,    *    ,    -2.00,    -1.00,     0.00]
```

Política:

```
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    D    ,    *    ,    D    ,    DL   ]
[    RD   ,    RD   ,    RD   ,    R    ,    D    ,    DL   ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    RD   ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,   SURD  ]
```

2.2 Caso $p_c = 0,8$ e $\gamma = 0,98$

Avaliação de Política

Função valor:

```
[   -47.19,   -47.11,   -47.01,    *    ,   -45.13,   -45.15]
[   -46.97,   -46.81,   -46.60,    *    ,   -44.58,   -44.65]
[   -46.58,   -46.21,   -45.62,   -44.79,   -43.40,   -43.63]
[   -46.20,   -45.41,   -44.42,    *    ,   -39.87,   -42.17]
[    *    ,   -44.31,   -41.64,   -35.28,   -32.96,    *    ]
[   -45.73,   -45.28,    *    ,   -29.68,   -21.88,     0.00]
```

Política:

```
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ]
[  SURDL  ,  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ]
[    *    ,  SURDL  ,  SURDL  ,  SURDL  ,  SURDL  ,    *    ]
[  SURDL  ,  SURDL  ,    *    ,  SURDL  ,  SURDL  ,    S    ]
```

Iteração de Valor

Função valor:

```
[   -11.65,   -10.78,    -9.86,    *    ,    -7.79,    -8.53]
[   -10.72,    -9.78,    -8.78,    *    ,    -6.67,    -7.52]
[    -9.72,    -8.70,    -7.59,    -6.61,    -5.44,    -6.42]
[    -8.70,    -7.58,    -6.43,    *    ,    -4.09,    -5.30]
[    *    ,    -6.43,    -5.17,    -3.87,    -2.76,    *    ]
[    -8.63,    -7.58,    *    ,    -2.69,    -1.40,     0.00]
```

Política:

```
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    RD   ,    D    ,    D    ,    R    ,    D    ,    D    ]
[    R    ,    RD   ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    D    ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,    S    ]
```

Iteração de Política

Função valor:

```
[   -11.66,   -10.79,    -9.87,    *    ,    -7.80,    -8.53]
[   -10.73,    -9.79,    -8.78,    *    ,    -6.67,    -7.52]
[    -9.73,    -8.70,    -7.60,    -6.61,    -5.44,    -6.42]
[    -8.70,    -7.58,    -6.43,    *    ,    -4.09,    -5.30]
[    *    ,    -6.43,    -5.18,    -3.87,    -2.76,    *    ]
[    -8.64,    -7.58,    *    ,    -2.69,    -1.40,     0.00]
```

Política:

```
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    D    ,    D    ,    D    ,    *    ,    D    ,    D    ]
[    R    ,    D    ,    D    ,    R    ,    D    ,    D    ]
[    R    ,    D    ,    D    ,    *    ,    D    ,    L    ]
[    *    ,    R    ,    R    ,    D    ,    D    ,    *    ]
[    R    ,    U    ,    *    ,    R    ,    R    ,    S    ]
```

3 Discussão dos Resultados

Caso 1: $p_c = 1.0$**,** $\gamma = 1.0$

No primeiro caso, como as ações são executadas com perfeição e o fator de desconto é unitário, espera-se que os algoritmos produzam políticas que minimizem estritamente o número de passos até o estado objetivo. Isso significa que, para cada estado, a política ótima deve corresponder a um caminho mais curto (em número de ações) até o objetivo.

A **avaliação de política aleatória**, como esperado, produziu uma função valor com valores altamente negativos, refletindo uma política ineficiente que leva muitas etapas para alcançar o objetivo (ou sequer o alcança consistentemente). Como a política é aleatória, os caminhos tomados não otimizam o custo acumulado, e os valores dos estados são muito baixos (por exemplo, até -384), evidenciando longos caminhos esperados.

Por outro lado, tanto a **iteração de valor** quanto a **iteração de política** resultaram em políticas e funções valor significativamente melhores. A função valor mostra uma redução linear nos custos à medida que se aproxima do objetivo, com valores variando de -10 até 0, refletindo trajetórias mínimas. As políticas produzidas são consistentes com o comportamento ótimo esperado, apresentando direções diretas (`D`, `R`, `DL`, etc.) em direção ao objetivo. Notavelmente, as políticas finais obtidas por ambos os métodos são equivalentes, como esperado pela teoria, pois ambos convergem para a política ótima.

Caso 2: $p_c = 0.8$**,** $\gamma = 0.98$

Neste cenário, a execução das ações é estocástica, com apenas 80% de chance de serem executadas corretamente. Além disso, o fator de desconto $\gamma = 0.98$ penaliza levemente ações futuras, o que prioriza trajetórias mais curtas, mas com menor severidade que no caso $\gamma = 1.0$.

Na **avaliação da política aleatória**, a função valor é novamente muito baixa (ex: -47), o que é coerente com uma política que não tenta ativamente alcançar o objetivo. O impacto da estocasticidade é visível na suavidade da função valor — os valores decrescem de maneira mais gradual, refletindo a maior incerteza na transição entre estados.

Tanto a **iteração de valor** quanto a **iteração de política** resultaram em políticas que, apesar de ainda eficientes, apresentaram valores levemente mais negativos do que no cenário determinístico. Por exemplo, a célula mais distante do objetivo tem valor próximo de -11.6, enquanto no caso determinístico o valor era -10. Isso reflete o aumento no custo esperado devido à incerteza nas ações e ao fator de desconto menor que 1. A política ótima, neste caso, ainda se aproxima das direções mais diretas, mas com ajustes sutis em função dos riscos e obstáculos.

De modo geral, os resultados obtidos estão de acordo com as expectativas vistas na teoria em sala. A avaliação de política, como esperado, não produz boas funções valor quando a política fornecida é aleatória. Já a iteração de valor e a iteração de política convergem para a mesma política ótima, conforme previsto, sendo que ambas são robustas em cenários determinísticos e estocásticos.

Além disso, é importante destacar que a estocasticidade das ações e o fator de desconto menor impõem um aumento no custo esperado total, como refletido pelos valores mais negativos da função valor no segundo cenário. Ainda assim, os algoritmos são capazes de lidar com essa incerteza e produzir soluções apropriadas.

Esses resultados confirmam a eficácia dos métodos de programação dinâmica na solução de MDPs quando o modelo do ambiente é completamente conhecido.