

冰湖路径规划

实验指导书

任务2 基于蒙特卡洛的路径规划

目 录

[一、实验信息 1](#_Toc133699359)

[二、实验准备 1](#_Toc133699360)

[1.实验环境 1](#_Toc133699361)

[2.实验数据 4](#_Toc133699362)

[三、实验步骤 4](#_Toc133699363)

[1. 蒙特卡洛策略评估 4](#_Toc133699364)

[2. 蒙特卡洛策略迭代 8](#_Toc133699365)

[四、实验总结 10](#_Toc133699366)

[五、实验思考 11](#_Toc133699367)

[六、课后任务 11](#_Toc133699368)

一、实验信息

|  |  |
| --- | --- |
| 实验名称 | 基于蒙特卡洛的路径规划 |
| 实验等级 | 中阶 |
| 实验时长 | 2课时 |
| 实验内容 | 1. 实现蒙特卡洛策略评估方法 2. 实现蒙特卡洛策略迭代方法 3. 测试最终策略 |
| 实验目标 | 1．掌握基于蒙特卡洛方法的策略评估算法  2．掌握𝝐-贪心策略  3．掌握基于蒙特卡洛方法的策略迭代算法 |

二、实验准备

### 1. 实验环境

|  |  |  |
| --- | --- | --- |
| 序列 | 名称 | 规格/版本 |
| 硬件 | PC计算机或服务器 | 1G以上内存，40G 以上硬盘，有英伟达GPU更佳 |
| 开发语言 | python | 3.8.13 |
| 强化学习环境 | gym | 0.24.1 |
| 数值计算 | numpy | 1.23.0 |

本次实验将基于OpenAI的Gym环境。OpenAI是一家非营利性的人工智能研究公司，公布了非常多的学习资源以及算法资源。其之所以叫作OpenAI，是因为他们把所有开发的算法都进行了开源。OpenAI 的 Gym库是一个环境仿真库，里面包含很多现有的环境。针对不同的场景，我们可以选择不同的环境。

本次实验将基于OpenAI的Gym环境。OpenAI是一家非营利性的人工智能研究公司，公布了非常多的学习资源以及算法资源。其之所以叫作OpenAI，是因为他们把所有开发的算法都进行了开源。OpenAI 的 Gym库是一个环境仿真库，里面包含很多现有的环境。针对不同的场景，我们可以选择不同的环境。

#### 1.1 环境安装

为了下载并安装 OpenAI Gym，我们可以使用以下方法：

|  |
| --- |
| pip install gym # minimal install  pip install gym[all] # full install  pip install gym[atari] # for Atari specific environment installation |

#### 1.2 环境介绍

在本实验中将会用到Gym中的冰湖环境（Frozen Lake），冰湖环境是一个网格环境，大小为 ，见图x-x。每一个方格对应了一个状态，智能体起点状态在左上角，目标状态在右下角，中间还有若干冰洞。在每一个状态都可以采取上、下、左、右 4 个动作。由于智能体在冰面行走，因此每次行走都不一定按预定目标方向移动，而是有一定的概率滑行到附近的其它状态。当掉入冰洞或到达目标状态时结束。每一步行走的奖励是 0，到达目标的奖励是 1。



图x-x 冰湖环境

我们可以通过如下代码查看一条随机轨迹：

|  |
| --- |
| import gym  env = gym.make("FrozenLake-v1") # 创建环境  env.reset()  for t in range(100):  env.render() # 渲染画面  a = env.action\_space.sample() # 随机采样动作  observation, reward, done, \_ = env.step(a) # 环境执行动作，获得转移后的状态、奖励以及环境是否终止的指示  if done:  break  env.render() |

### 2.实验数据

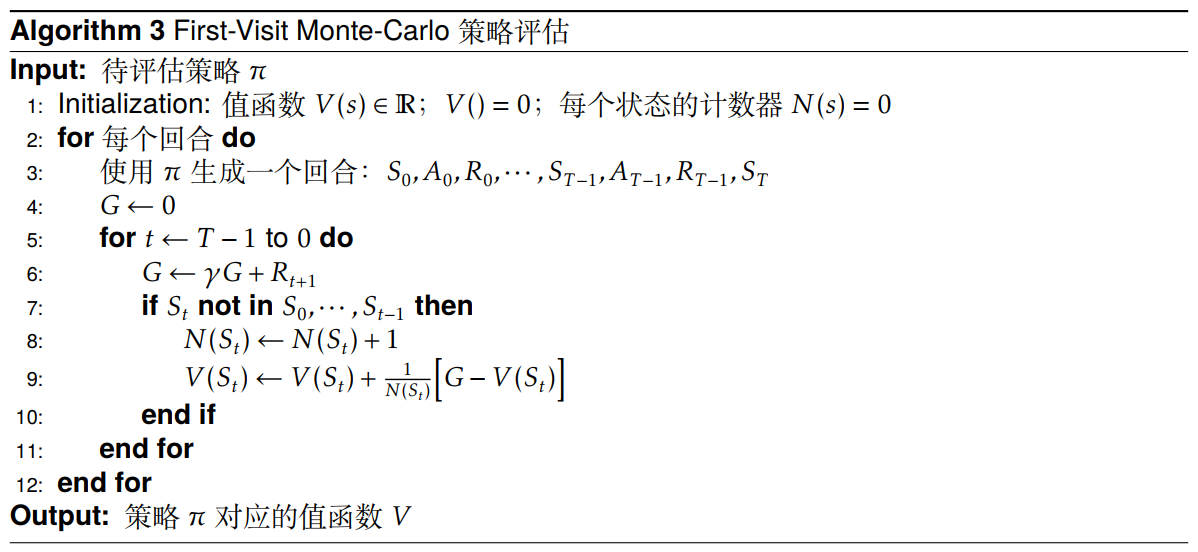
无。

三、实验步骤

### 1. 蒙特卡洛策略评估

在实际应用场景中，我们通常无法获知环境的模型信息，这就使得我们无法使用动态规划方法来寻找最优策略。因此，我们需要智能体与环境进行交互，并根据交互结果进行进一步的策略评估与策略优化。

在本节实验中，我们将使用蒙特卡洛方法对策略进行评估，具体而言，将使用首次访问蒙特卡洛策略评估，只考虑每个回合中对状态s的第一次访问。下面为首次访问蒙特卡洛策略评估伪代码：



基于状态价值函数 𝑉(𝑠) 的贪心策略提升需要知道马尔可夫决策过程的模型信息，即状态转移概率与奖励函数，

而基于动作价值函数 𝑄(𝑠) 的策略提升方法无需知道模型信息，

因此在这里我们不对价值函数进行评估，而是利用蒙特卡罗方法评估动作价值函数。在任务我们需要实现compute\_qpi\_MC函数。这里的参数epsilon对应着ε-贪心策略中的参数，其作用为确保充分探索环境以寻找到最优策略。

#### 1.1 Compute\_qpi\_MC函数实现

|  |
| --- |
| def compute\_qpi\_MC(pi, env, gamma, epsilon, num\_episodes=1000):  """  使用蒙特卡洛方法来估计动作价值函数Q\_pi。  参数：  pi -- 在环境env中使用的确定性策略，是一个大小为状态数的numpy数组，输入状态，输出动作。  env -- OpenAI Gym环境对象。  gamma -- 折扣因子，一个0到1之间的浮点数。  num\_episodes -- 进行采样的回合数。  返回值：  Q -- 动作价值函数Q\_pi的估计，是一个字典，键是状态-动作对，值是该状态-动作对的估计值。  """  Q = np.zeros((env.observation\_space.n, env.action\_space.n), dtype=np.float32)  N = np.zeros((env.observation\_space.n, env.action\_space.n), dtype=np.int64)  for i in range(num\_episodes):  # 生成新的回合  state = env.reset()  episode = []  # 对于该回合中的每个时间步  while True:  # 根据策略选择动作  if np.random.rand() < epsilon:  action = env.action\_space.sample()  else:  action = pi[state]  # 执行动作，获得新状态和回报值  next\_state, reward, done, \_ = env.step(action)  # 记录状态、动作、回报值  episode.append((state, action, reward))  # 如果回合结束，记录最后一个状态、动作和奖励，退出  if done:  episode.append((next\_state, None, reward))  break  # 转换到下一个状态  state = next\_state  # 对于该回合中的每个状态-动作对  visited = set()  G = 0  for j, (state, action, reward) in enumerate(reversed(episode)):  G = gamma \* G + reward  if action == None:  continue  sa = (state, action)  # 如果该状态-动作对没有被访问过，更新N和Q  if sa not in visited:  visited.add(sa)  state = int(state)  action = int(action)  N[state][action] += 1  G = gamma \* G + reward  Q[state][action] += (G - Q[state][action]) / N[state][action]  return Q |

#### 1.2 Compute\_qpi函数的输出

|  |
| --- |
| Qpi = compute\_qpi\_MC(np.ones(16), env, gamma=0.95)  print("Qpi:\n", Qpi)  输出如下：  Qpi:  [[0. 0.06250776 0. 0. ]  [0. 0.04515086 0. 0. ]  [0. 0.08371656 0. 0. ]  [0. 0.03386063 0. 0. ]  [0. 0.09250954 0. 0. ]  [0. 0. 0. 0. ]  [0. 0.20487688 0. 0. ]  [0. 0. 0. 0. ]  [0. 0.22189462 0. 0. ]  [0. 0.5202708 0. 0. ]  [0. 0.6260544 0. 0. ]  [0. 0. 0. 0. ]  [0. 0. 0. 0. ]  [0. 0.8973312 0. 0. ]  [0. 1.8603263 0. 0. ]  [0. 0. 0. 0. ]] |

### 2. 蒙特卡洛策略迭代

在实现compute\_qpi\_MC函数后，我们可以根据其计算所得的动作价值函数，利用贪心算法导出一个提升后的策略，实现策略提升步骤。进而我们可以实现蒙特卡洛策略迭代算法。

#### 2.1 蒙特卡洛策略迭代算法实现

|  |
| --- |
| def policy\_iteration\_MC(env, gamma, eps0=0.5, decay=0.1, num\_episodes=1000):  """  使用蒙特卡洛方法来实现策略迭代。  参数：  env -- OpenAI Gym环境对象。  gamma -- 折扣因子，一个0到1之间的浮点数。  eps0 -- 初始的探索概率。  decay – 衰减速率。  num\_episodes -- 进行采样的回合数。  返回值：  Q -- 动作价值函数Q\_pi的估计，是一个字典，键是状态-动作对，值是该状态-动作对的估计值。  """  pi = np.zeros(env.observation\_space.n)  iteration = 1  while True:  epsilon = eps0/(1+decay\*iteration)  Q = compute\_qpi\_MC(pi, env, gamma, epsilon, num\_episodes)  new\_pi = Q.argmax(axis=1)  if (pi != new\_pi).sum() == 0: # 策略不再改变，作为收敛判定条件  return new\_pi  # print(f"iteration: {iteration}, eps: {epsilon}, change actions: {(pi != new\_pi).sum()}")  pi = new\_pi  iteration = iteration + 1 |

这里的epsilon采用逆时间衰减：将初始的探索概率 eps0 和衰减速率 decay 设置为两个常数，然后让探索概率按照逆时间函数衰减，例如 eps\_t = eps0 / (1 + decay \* t)，其中 t 表示当前迭代的次数。

#### 2.2 策略性能测试

|  |
| --- |
| def test\_pi(env, pi, num\_episodes=100):  """  测试策略。  参数：  env -- OpenAI Gym环境对象。  pi -- 需要测试的策略。  num\_episodes -- 进行测试的回合数。  返回值：  成功到达终点的频率。  """  count = 0  for e in range(num\_episodes):  ob = env.reset()  for t in range(100):  a = pi[ob]  ob, rew, done, \_ = env.step(a)  if done:  count += 1 if rew == 1 else 0  break  return count / num\_episodes  # pi = policy\_iteration\_MC(env, 1, num\_episodes=10000)  # result = test\_pi(env, optimal\_pi)  # print(result) |

四、实验总结

1．在本次实验中，我们假定不知道环境的转移概率与奖励函数，因此需要与环境实际交互以获取对转移概率与奖励函数的估计。

2. 具体来说，采用了蒙特卡洛方法来估计价值函数。而在不知道转移概率时，我们难以通过状态价值函数来进行策略提升步骤，因此在实际操作时，我们利用蒙特卡洛方法对动作价值函数进行评估。

3. 为了对环境进行充分探索，这里利用了ε-贪心策略，并选择特定的衰减策略来对ε进行衰减。

五、实验思考

问题1. 蒙特卡洛方法能和动态规划方法收敛到一样的最优价值函数吗？

问题2. 蒙特卡洛方法相比上一节的动态规划方法有何异同点？

六、课后任务

1. 以上示例代码只具有基本可行的效果，还需要同学们在课后进一步的优化，这里给出几点优化方向：

（1）epsilon的衰减策略：如采用线性衰减、指数衰减。

（2）收敛的判定策略：如固定迭代轮数。

（3）调整参数：如折扣因子、评估时采样的回合数、epsilon的初始值与衰减速率。

2. 每次策略评估阶段都需要收敛后再进行策略提升吗？思考如果不等待收敛就进行策略提升，该如何修改代码。