Optimal Control and Reinforcement Learning

骆皓青

22112094

Institute of Intelligent Transportation System

1. Dynamic Programming

这两小题都用到了递归，见 **method shortest_path(self, x, y)**，其中在第二小题多了一个 heuristic 的 rollout policy 即 **method nearest_neighbor(self, x, y)**。

另外需要说明的一点是在建模的过程中，我们将 B 置成了（0，0），斜向下为 x 轴，斜向上为 y 轴，因此 A 点的坐标为（3，3），res 与 res_prime 分别表示 exactly dp 与 rollout 算法的结果。前者中，输出路径算是一个比较 dirty 的工作，于是换了另一种方式，即将（x，y）到（0，0）的所有 optimality 都打印出来，对应相减与实际的 reward 比较，如果差值小于等于 reward，那么该路段是最优路径的一部分，最后将所有的潜在路段连接起来就是最优路径。

最短路长均为 40，路径也一致，均为（3，3）>（2，3）>（1，3）>（1，2）>（0，2）>（0，1）>（0，0）。

2. Model Predictive Control

这道题我一直怀疑自己做错了，关键点在于 terminal state 为零与 action bounds 无法同时满足，也就导致了在用 scipy.optimize 解最小值的时候经常会无解。随着 L（题目里面是 N）的增大这种现象会缓解，但是当 N=4 的时候前几步仍旧无法同时满足 terminal state 为零与 action bounds。

基于此，我们在 action bounds 方面做了一定的妥协，即满足第一步（实际会采取的控制），放宽后几步的动作限制，见 code 里面的 **bounds = [b, un, un]**，对于 action sequence 的求解见 **method solve_mpc(self)**。

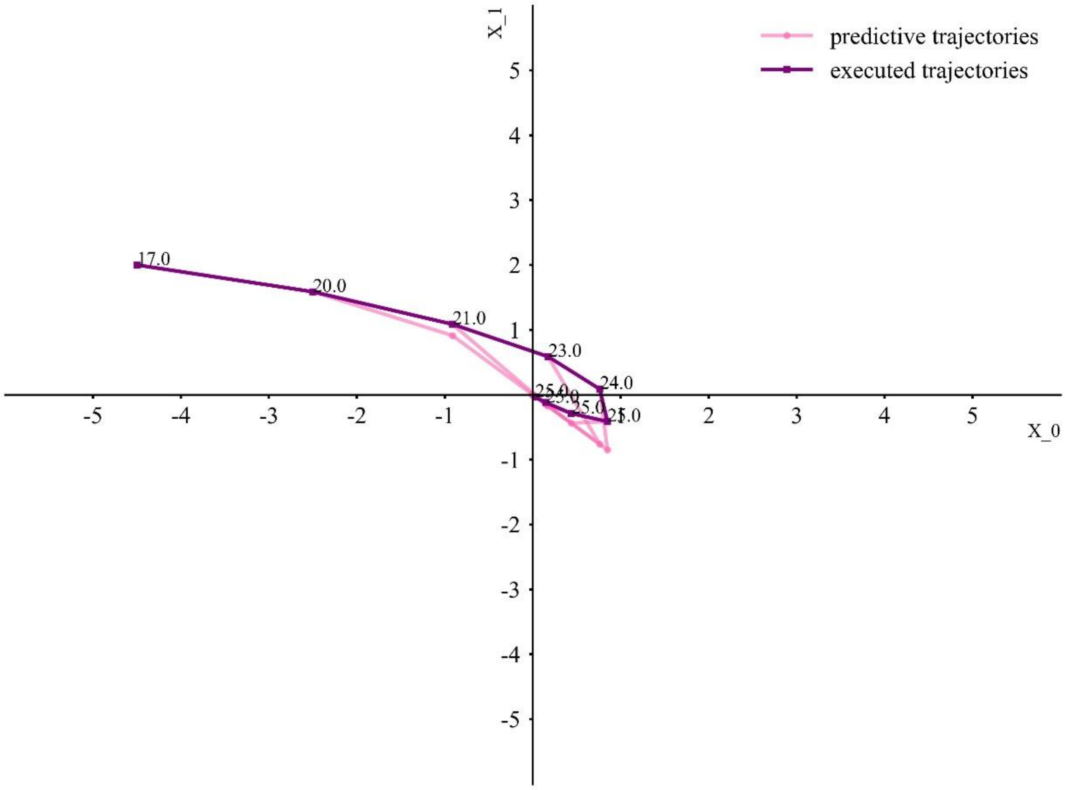

图 1. N=3 初始状态（-4.5，2）预测控制与实际控制轨迹图

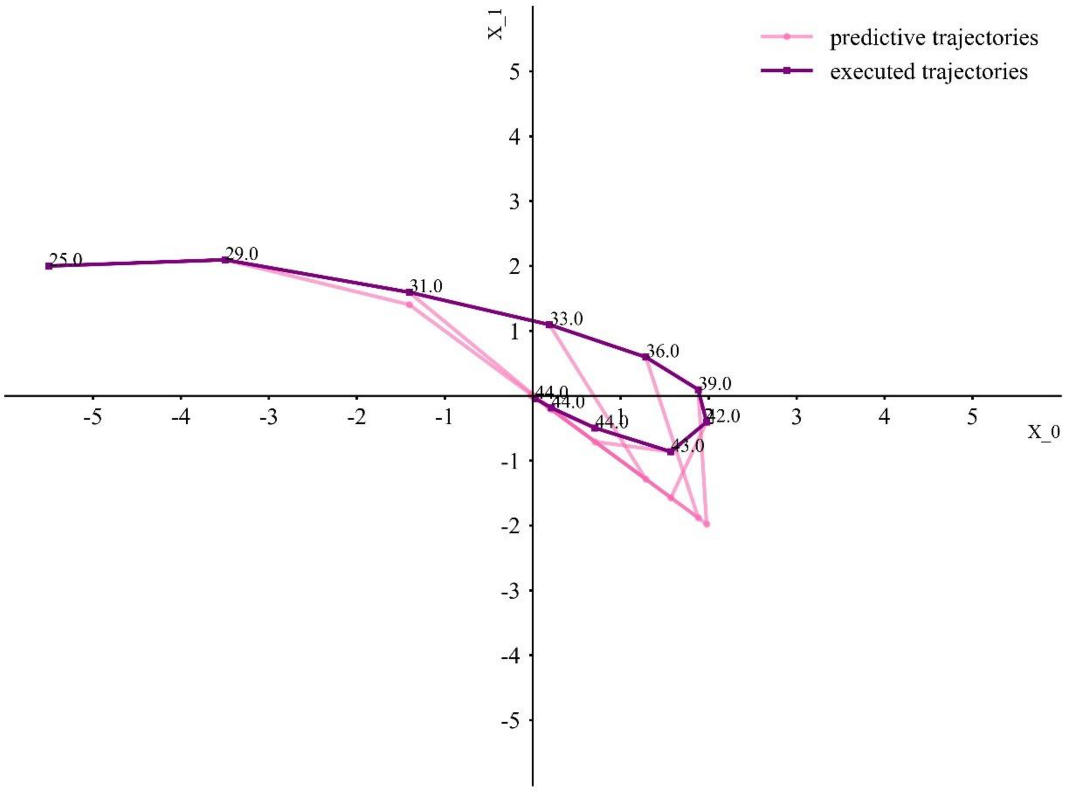

图 2. N=3 初始状态（-5.5，2）预测控制与实际控制轨迹图

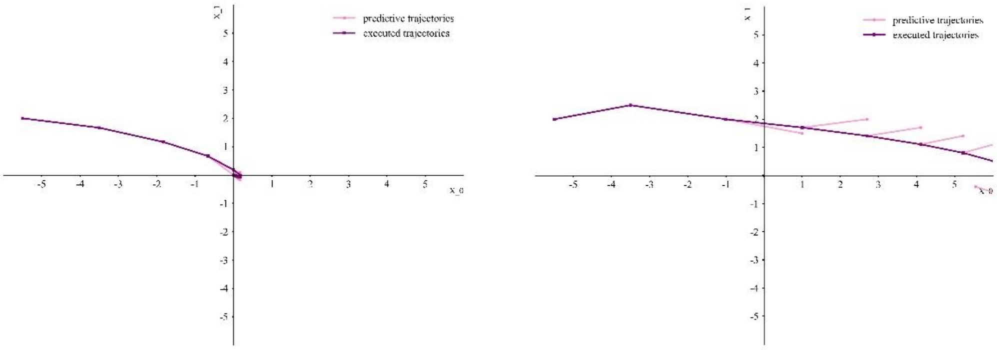

图 3. N=2 与 N=4 初始状态（-5.5，2）预测控制与实际控制轨迹图

图 1 与图 2 是 N=3 在不同初始状态下的预测控制与实际控制的示意图，图里信息量较大因此不再说明。图三是 N=2 与 N=4，初始状态为（-5.5，2）的状态下预测控制与实际控制的示意图，可以得到的 insight 有

- N 较小的情况下，控制策略比较激进，会出现无法收敛至次优的情况，且 cost 比较大。
- N 较大的情况下，控制策略比较平滑、高效（长远考虑），在轨迹上能够迅速收敛至次优，且 cost（31.19，迭代 10 次）小于 N=3 的 cost（44.00，迭代 10 次）。
- 关于 N 的选择，理论上 N 越大越到，其控制性能是单调递增的，但是鉴于当 N 到达一定的限度的时候其性能提升不大以及计算量的大幅增大，我们应该选择一个合适的 N。

3. Reinforcement Learning

3.1.1 Analytical Method

首先是编程的思路，从 terminal state 开始，并往$J^*(x_0)$开始迭代：

$$J^*(x_{T-1}) = {x_{T-1}}^2 + {u_{T-1}}^2 + J^*(x_T)$$

$$= {x_{T-1}}^2 + {u_{T-1}}^2$$

取$u_{T-1} = 0$

$$J^*(x_{T-2}) = {x_{T-2}}^2 + {u_{T-2}}^2 + {x_{T-1}}^2$$

$$= {x_{T-2}}^2 + {u_{T-2}}^2 + (x_{T-2} + u_{T-2})^2$$

我们同样会解出$J^*(x_{T-2}) = t_{T-2}{x_{T-2}}^2$并代入到$J^*(x_{T-3})$中

…

$$J^*(x_0) = {x_0}^2 + {u_0}^2 + t_1(x_0 + u_0)^2$$

根据以上 notation，我们可以得到$u_i^* = -\frac{t_{i+1}}{1+t_{i+1}}x_i$

因此整个 DP 的逻辑是，从 terminal 到 initial state 计算出t_i序列（backward），再根据x_0计算出最优控制序列u_i^*(forward)，且最终的 cost 为$J^*(x_0)$，将x_0和u_0^*代入即可。

3.1.2 Value Iteration

我们假设针对x_i的 state value 为 $k{x_i}^2$

$$J^*(x_i) = min\ {x_i}^2 + {u_i}^2 + k(x_i + u_i)^2$$

右边取$u_i = -\frac{kx_i}{1+k}$，得到：

$$k = \frac{k}{1+k} + 1$$

推出$k^* = \frac{1+\sqrt{5}}{2}$，负的值舍去，因此

$$u_i^* = -\frac{k^* x_i}{1+k^*}$$

3.2 Policy Iteration

这其实是一个最简化版的 Q-learning，阉割了神经网络（包括它系数的 backward）、replay_buffer（其实有点类似于 on policy 算法，也就是说用当前策略采样 n 个 episode 然后对 critic 和 actor 进行更新）那更别说现在普遍存在的 Double 和 Dueling 等技巧。

为了验证 exact dynamic programming, value iteration 和 policy iteration 的正确性，我们尝试了一个初始状态 4，并计算了各个方法的累积代价分别为 25.89、25.89 以及 32.11。鉴于 policy iteration 在 policy mapping 的时候引入了一些有利于探索的扰动，这个误差是可以接受的。

```
Under exact dynamic programming, the cumulative cost is 25.88854341066731
Under value iteration, the cumulative cost is 25.88854381999832
```

```
optimal policy mapping: 0.9967560229700363
optimal action value mapping:
 [[ 0.49812699 -0.49976406]
 [-0.49976406  0.50139056]]
Under the optimal policy the trajectories are
 [4, -0.013082106324792342, 0.004180732962268126, -0.002963114768185212, 0.003514781958980892, 0.0025914222233763386, -0.0006659954600313003]
The cumulative cost is 32.105446928038546
```