# 实验报告

1120210526 马煜

目录

[实验报告 1](#_Toc145958571)

[实验名称： 1](#_Toc145958572)

[实验要求： 1](#_Toc145958573)

[实验目的： 1](#_Toc145958574)

[实验过程： 2](#_Toc145958575)

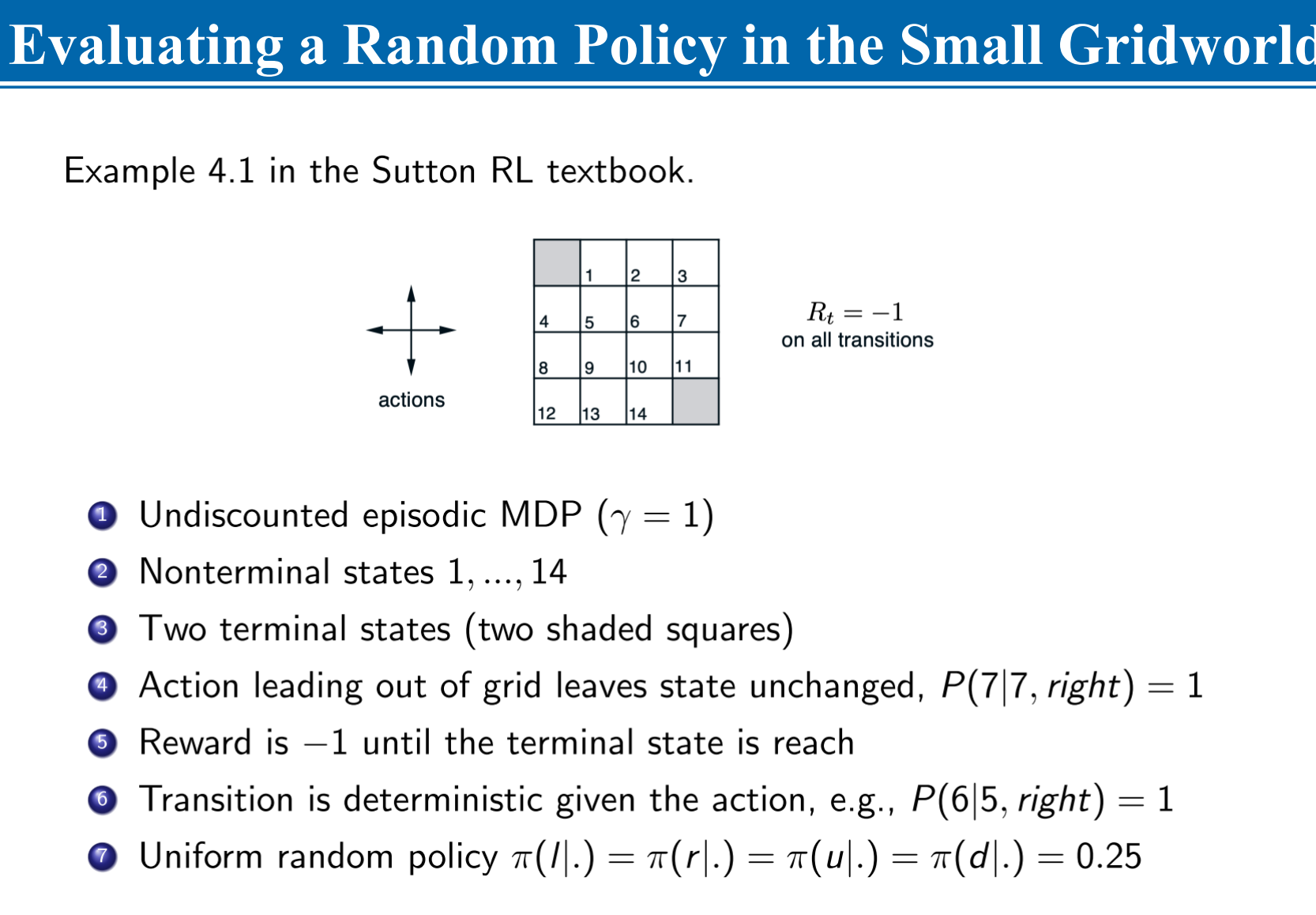
[实验结果 5](#_Toc145958576)

[实验思考和总结： 6](#_Toc145958577)

## 实验名称：

小网格环境下策略评估及马尔科夫策略实验

## 实验要求：



## 实验目的：

* 在小网格中实现对随机策略的评估，了解策略评估是一个不停迭代的过程
* 学习马尔科夫决策过程（MDP）中的基本概念，包括状态、动作、奖励等，并了解如何使用MDP框架来建模和解决强化学习问题
* 学习使用策略评估使用的价值函数，了解公式中的各部分意义，了解价值函数在强化学习中的关键作用

## 实验过程：

实验代码如下：

MAX\_iterations = 100000  *# Maximum number of iterations*

MAX\_num = 5  *# Decimal places to keep*

MAX\_delta = 0.0000001  *# Maximum error*

def **chess\_score**(*chess\_number*):

    global score

    global newscore

    endpos1 = [[0, 0], [*chess\_number* - 1, *chess\_number* - 1]]

    score = [[0.0 for \_ in range(*chess\_number*)] for \_ in range(*chess\_number*)]

    newscore = [[0.0 for \_ in range(*chess\_number*)] for \_ in range(*chess\_number*)]

    global iteration

    for iteration in range(MAX\_iterations):

        global delta

        delta = 0.0

        if iteration <= 10 or int(iteration) % 10 == 0:  *# To make the results clearer, print every 10 iterations after the first 10*

**print**("Iteration", iteration, ":")

            for row in score:

**print**(row)

        for i in range(*chess\_number*):

            for j in range(*chess\_number*):

                R=4

                if [i, j] in endpos1:

                    score[i][j] = 0.0

                else:

                    total = 0

                    count = 0

                    for x in [-1, 1]:

                        ni = i + x

                        if 0 <= ni < *chess\_number*:

                            total += score[ni][j]

                            count += 1

                            if [ni, j] in endpos1:

                                R = 3

                    for y in [-1, 1]:

                        nj = j + y

                        if 0 <= nj < *chess\_number*:

                            total += score[i][nj]

                            count += 1

                            if [i, nj] in endpos1:

                                R = 3

                    newscore[i][j] = **round**(((total + (4 - count) \* score[i][j]) - R) / 4, MAX\_num)

                    delta = **max**(delta, **abs**(score[i][j] - newscore[i][j]))

        score = [row[:] for row in newscore]

        if delta < MAX\_delta:

            break

    optimal\_policy = [['' for \_ in range(*chess\_number*)] for \_ in range(*chess\_number*)]

    for i in range(*chess\_number*):

        for j in range(*chess\_number*):

            if [i, j] in endpos1:

                optimal\_policy[i][j] = 'End'

            else:

                best\_actions = []

                max\_neighbor = float('-inf')

                for x in [-1, 1]:

                    ni = i + x

                    if 0 <= ni < *chess\_number*:

                        neighbor\_value = score[ni][j]

                        if neighbor\_value > max\_neighbor:

                            max\_neighbor = neighbor\_value

                            best\_actions = ['Up'] if x == -1 else ['Down']

                        elif neighbor\_value == max\_neighbor:

                            best\_actions.**append**('Up' if x == -1 else 'Down')

                for y in [-1, 1]:

                    nj = j + y

                    if 0 <= nj < *chess\_number*:

                        neighbor\_value = score[i][nj]

                        if neighbor\_value > max\_neighbor:

                            max\_neighbor = neighbor\_value

                            best\_actions = ['Left'] if y == -1 else ['Right']

                        elif neighbor\_value == max\_neighbor:

                            best\_actions.**append**('Left' if y == -1 else 'Right')

                optimal\_policy[i][j] = '/'.**join**(best\_actions)

**print**("Final result after", int(iteration + 1), "iterations:")

    for row in score:

**print**([**round**(element, 3) for element in row])

**print**("Optimal Policy:")

    for row in optimal\_policy:

**print**(row)

**print**("Calculation error:", delta)

if \_\_name\_\_ == "\_\_main\_\_":

**chess\_score**(4)

在函数的开头，先初始化网格，确定终点位置，确定价值矩阵和临时的价值矩阵，并且初始化为零。

global score

    global newscore

    endpos1 = [[0, 0], [*chess\_number* - 1, *chess\_number* - 1]]

    score = [[0.0 for \_ in range(*chess\_number*)] for \_ in range(*chess\_number*)]

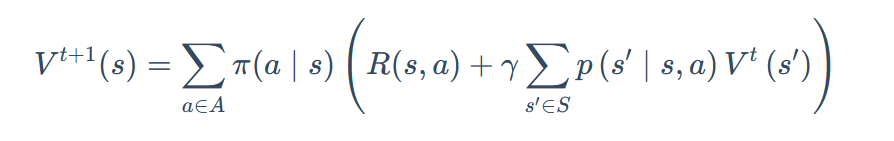
    newscore = [[0.0 for \_ in range(*chess\_number*)] for \_ in range(*chess\_number*)]

    global iteration

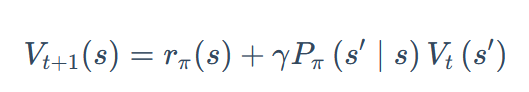
    for iteration in range(MAX\_iterations):

        global delta

        delta = 0.0

每个循环中遍历更新该点的未来价值和奖励函数，奖励函数通过R=4来初始化，当该点下一步为终点时，将R改为3。更新遵循价值函数：

当策略确定为随机策略即向各个方向都有25%的概率时，

价值函数可以更新为：

for i in range(*chess\_number*):

            for j in range(*chess\_number*):

                R=4

                if [i, j] in endpos1:

                    score[i][j] = 0.0

                else:

                    total = 0

                    count = 0

                    for x in [-1, 1]:

                        ni = i + x

                        if 0 <= ni < *chess\_number*:

                            total += score[ni][j]

                            count += 1

                            if [ni, j] in endpos1:

                                R = 3

                    for y in [-1, 1]:

                        nj = j + y

                        if 0 <= nj < *chess\_number*:

                            total += score[i][nj]

                            count += 1

                            if [i, nj] in endpos1:

                                R = 3

                    newscore[i][j] = **round**(((total + (4 - count) \* score[i][j]) - R) / 4, MAX\_num)

此后是对策略评估和最优策略的可视化部分的代码，不再赘述

## 实验结果

Iteration 230 :

[0.0, -12.99996, -18.99994, -20.99994]

[-12.99996, -16.99995, -18.99994, -18.99994]

[-18.99994, -18.99994, -16.99995, -12.99996]

[-20.99994, -18.99994, -12.99996, 0.0]

Final result after 234 iterations:

[0.0, -13.0, -19.0, -21.0]

[-13.0, -17.0, -19.0, -19.0]

[-19.0, -19.0, -17.0, -13.0]

[-21.0, -19.0, -13.0, 0.0]

Optimal Policy:

['End', 'Left', 'Left', 'Down/Left']

['Up', 'Up/Left', 'Down/Left', 'Down']

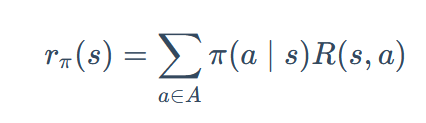
['Up', 'Up/Right', 'Down/Right', 'Down']

['Up/Right', 'Right', 'Right', 'End']

Calculation error: 0.0

可以看到，在经历230次迭代后，结果就近乎收敛，得到了最终的策略评估以及最优策略。

## 实验思考和总结：

在最一开始的一个版本中，我没有将到达终点的那一个行为的回报设置为0，导致评估结果的大小有一点偏差。这是因为我忘记了r(S)的定义为：没有深刻理解和p的相互关系