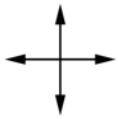
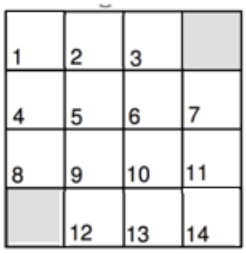
作业 2　DP 方法实践

学号：[redacted]　　姓名：[redacted]

π = equiprobable random action choices





R_t = −1
on all transitions

γ = 1

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . ., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is –1 until the terminal state is reached

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_k(s')\right] \qquad \forall s \in \mathcal{S}$$

给定初始随机策略和状态价值函数 V0：

V_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

1. 针对上述 Gridworld 问题编码实现：
    a) 用本 PPT 讲的 Policy Evaluation 的方法计算v_π
    b) 用 Policy Iteration 方法搜索最优v_*，p_*
    c) 用 Value Iteration 方法搜索最优v_*，p_*
2. 分析性能
    a) 分析 Policy Evaluation 的迭代次数和收敛时间
    b) 比较 Policy Iteration 和 Value Iteration 的迭代次数
    c) 比较 Policy Iteration 和 Value Iteration 收敛时间
    d) 分析 Policy Iteration 和 Value Iteration 收敛误差随着迭代次数的分布曲线
3. 整理报告，包含实验结果和代码，提交作业

1. 编码实现：

a) 用本 PPT 讲的 Policy Evaluation 的方法计算v_π

初始化：

```
# 状态价值 V_pi
grid = torch.zeros([4, 4])
# 策略 pi
pi = [[[k, 0.25] for k in range(4)] for j in
range(4)] for i in range(4)]
```

核心代码：

```
# 策略评估
def policy_evaluation(grid, pi):
    small_number = 0.00001
    max_change = 1
    iteration_number = 0
    start = time.time()
    while max_change > small_number:
        max_change = 0
        iteration_number += 1
        grid2 = grid.clone()
        for i in range(4):
            for j in range(4):
                if i == 0 and j == 3 or i == 3 and j == 0:
                    continue
                v = grid[i][j]
                t = 0
                for k, p in pi[i][j]:
                    t += p * q_value(i, j, k, grid)
                grid2[i][j] = t
                max_change = max(max_change, abs(t - v))
        grid = grid2
    print("v_pi:", grid)
    print("policy_evaluation收敛时间", time.time() - start)
    print("迭代次数", iteration_number)
    return grid
```

运行文件 work1.py，得到如下输出：

```
C:\Users\qin23\.conda\envs\myenv\python.exe "D:\download\work (2)\work1.py"
v_pi: tensor([[-21.9998, -19.9999, -13.9999,   0.0000],
        [-19.9999, -19.9999, -17.9999, -13.9999],
        [-13.9999, -17.9999, -19.9999, -19.9999],
        [  0.0000, -13.9999, -19.9999, -21.9998]])
policy_evaluation收敛时间 0.29460668563842773
迭代次数 218

进程已结束，退出代码为 0
```

b) 用 Policy Iteration 方法搜索最优v_*，p_*

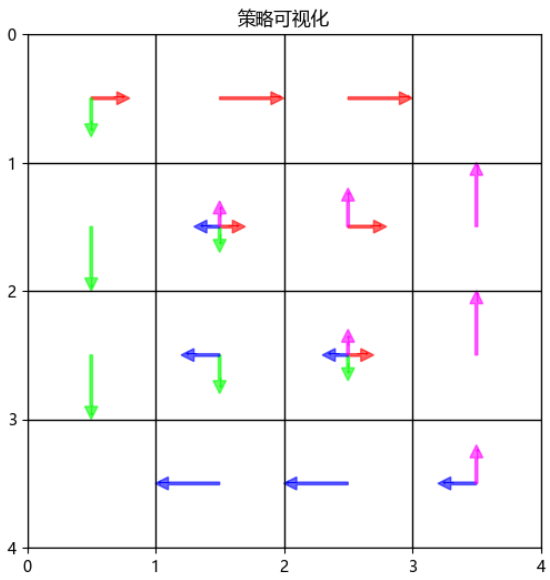
核心代码：

```python
def policy_iteration(grid, pi):
    start = time.time()
    policy_stable = false
    iteration_number=0
    while not policy_stable:
        iteration_number+=1
        grid = policy_evaluation(grid, pi)
        pi, policy_stable = policy_improvement(grid, pi)
    print("v_pi:", grid)
    print("policy_iteration收敛时间", time.time() - start)
    print("迭代次数", iteration_number)
    visualize_policy(pi)
```

运行文件 work2.py，得到如下输出：

```
C:\Users\qin23\.conda\envs\myenv\python.exe "D:\download\work (2)\work2.py"
v_pi: tensor([[-3., -2., -1.,  0.],
        [-2., -3., -2., -1.],
        [-1., -2., -3., -2.],
        [ 0., -1., -2., -3.]])
policy_iteration收敛时间 0.2951467037200928
迭代次数 3

进程已结束，退出代码为 0
```



c） 用 Value Iteration 方法搜索最优v_*，p_*

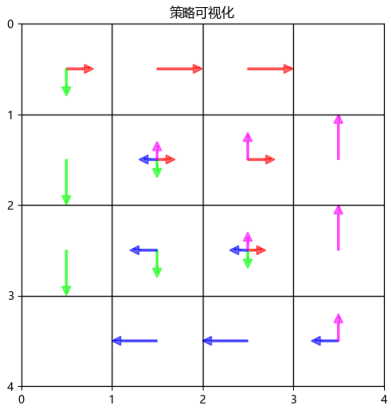
核心代码：

```python
def value_iteration(grid):
    small_number = 0.00001
    max_change = 1
    iteration_number = 0
    start = time.time()
    while max_change > small_number:
        max_change = 0
        iteration_number += 1
        grid2 = grid.clone()
        for i in range(4):
            for j in range(4):
                if i == 0 and j == 3 or i == 3 and j == 0:
                    continue
                v = grid[i][j]
                grid2[i][j] = max([q_value(i, j, k, grid) for k in range(4)])
                max_change = max(max_change, abs(grid2[i][j] - v))
        grid = grid2
    print("v_pi:", grid)
    print("value_iteration收敛时间", time.time() - start)
    print("迭代次数", iteration_number)
```

运行文件 work3.py，得到如下输出：

```
C:\Users\qin23\.conda\envs\myenv\python.exe "D:\download\work (2)\work3.py"
v_pi: tensor([[-3., -2., -1.,  0.],
        [-2., -3., -2., -1.],
        [-1., -2., -3., -2.],
        [ 0., -1., -2., -3.]])
value_iteration收敛时间 0.005999326705932617
迭代次数 4

进程已结束，退出代码为 0
```



2. 分析性能

a) 分析 Policy Evaluation 的迭代次数和收敛时间

运行文件 work1.py，得到 policy_evaluation 收敛时间 0.29557156562805176
迭代次数 218
说明停止条件 ϵ 比较严格，导致迭代次数较多。

b) 比较 Policy Iteration 和 Value Iteration 的迭代次数

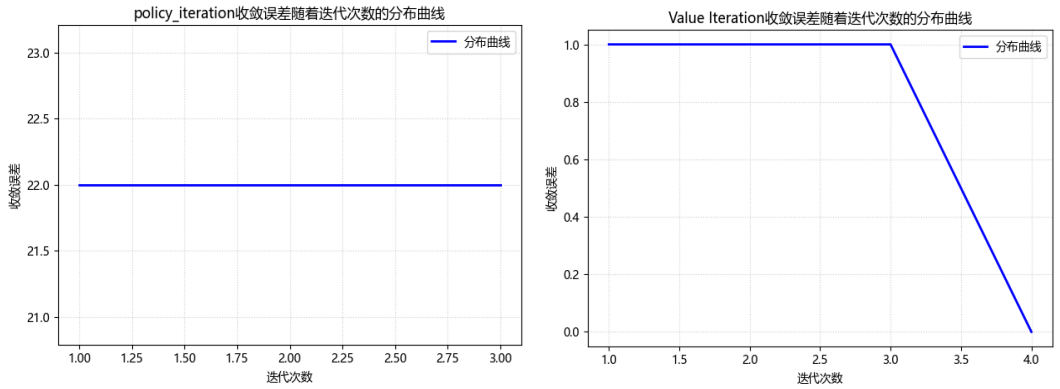
运行文件 work2.py 和 work3.py，分别得到：
policy_iteration 收敛时间 0.2888069152832031，迭代次数 3
value_iteration 收敛时间 0.005999326705932617，迭代次数 4
可以看到的是 policy_iteration 的迭代次数更少，但是收敛时间较长，这是由于 Policy Evaluation 每次收敛需要的时间较长。

c) 比较 Policy Iteration 和 Value Iteration 收敛时间

policy_iteration 收敛时间 0.2888069152832031
value_iteration 收敛时间 0.005999326705932617
相对来说 value_iteration 的收敛时间较少。

d) 分析 Policy Iteration 和 Value Iteration 收敛误差随着迭代次数的分布曲线

运行文件 work4.py，结果如下：



Policy Iteration:可以看出，在整个迭代过程中，收敛误差保持在一个相对稳定的水平，大约在 22.0 左右波动。这表明 policy_iteration 算法在该特定情况下收敛得非常快，且误差稳定，没有随着迭代次数的增加而显著变化。

Value Iteration：收敛误差相对较小，随着迭代次数的增加，呈现逐渐减小的趋势。