숙제:

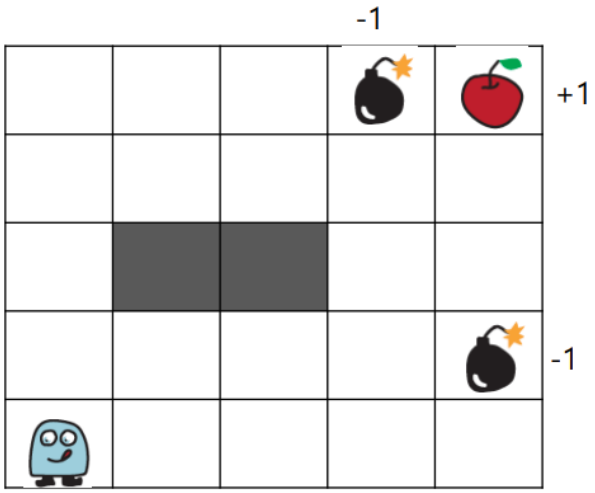
내용 :

Quiz

(Q) Dynamic programming 을 적용하여 5x5 Grid World 에 대한 value function 및 policy 를 구하라.

(1)Policy iteration method

(2)Value iteration method



37

관련 코드 정리

1. Gridworld_ernder.py

코드

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
 	
class Renderer:
  def __init__(self, reward_map, goal_state, wall_state):

```
        self.reward_map = reward_map
        self.goal_state = goal_state
        self.wall_state = wall_state
        self.ys = len(self.reward_map)
        self.xs = len(self.reward_map[0])

        self.ax = None
        self.fig = None
        self.first_flg = True

    def set_figure(self, figsize=None):
        fig = plt.figure(figsize=figsize)
        self.ax = fig.add_subplot(111)
        ax = self.ax
        ax.clear()
        ax.tick_params(labelbottom=False, labelleft=False, labelright=False,
labeltop=False)
        ax.set_xticks(range(self.xs))
        ax.set_yticks(range(self.ys))
        ax.set_xlim(0, self.xs)
        ax.set_ylim(0, self.ys)
        ax.grid(True)

    def render_v(self, v=None, policy=None, print_value=True):
        self.set_figure()

        ys, xs = self.ys, self.xs
        ax = self.ax

        if v is not None:
            color_list = ['red', 'white', 'green']
            cmap = matplotlib.colors.LinearSegmentedColormap.from_list(
                'colormap_name', color_list)
```

```
        # dict -> ndarray
        v_dict = v
        v = np.zeros(self.reward_map.shape)
        for state, value in v_dict.items():
            v[state] = value

        vmax, vmin = v.max(), v.min()
        vmax = max(vmax, abs(vmin))
        vmin = -1 * vmax
        vmax = 1 if vmax < 1 else vmax
        vmin = -1 if vmin > -1 else vmin

        ax.pcolormesh(np.flipud(v), cmap=cmap, vmin=vmin, vmax=vmax)

    for y in range(ys):
        for x in range(xs):
            state = (y, x)
            r = self.reward_map[y, x]
            if r != 0 and r is not None:
                txt = 'R ' + str(r)
                if state == self.goal_state:
                    txt = txt + ' (GOAL)'
                ax.text(x+.1, ys-y-0.9, txt)

            if (v is not None) and state != self.wall_state:
                if print_value:
                    offsets = [(0.4, -0.15), (-0.15, -0.3)]
                    key = 0
                    if v.shape[0] > 7: key = 1
                    offset = offsets[key]
                    ax.text(x+offset[0], ys-y+offset[1], "{:12.2f}".format(v[y, x]))

            if policy is not None and state != self.wall_state:
                actions = policy[state]
```

```
                        max_actions = [kv[0] for kv in actions.items() if kv[1] ==
max(actions.values())]

                        arrows = ["↑", "↓", "←", "→"]
                        offsets = [(0, 0.1), (0, -0.1), (-0.1, 0), (0.1, 0)]
                        for action in max_actions:
                            arrow = arrows[action]
                            offset = offsets[action]
                            if state == self.goal_state:
                                continue
                            ax.text(x+0.45+offset[0], ys-y-0.5+offset[1], arrow)

                    if state == self.wall_state:
                        ax.add_patch(plt.Rectangle((x,ys-y-1), 1, 1, fc=(0.4, 0.4, 0.4, 1.)))
        plt.show()

    def render_q(self, q, show_greedy_policy=True):
        self.set_figure()

        ys, xs = self.ys, self.xs
        ax = self.ax
        action_space = [0, 1, 2, 3]

        qmax, qmin = max(q.values()), min(q.values())
        qmax = max(qmax, abs(qmin))
        qmin = -1 * qmax
        qmax = 1 if qmax < 1 else qmax
        qmin = -1 if qmin > -1 else qmin



        color_list = ['red', 'white', 'green']
        cmap = matplotlib.colors.LinearSegmentedColormap.from_list(
            'colormap_name', color_list)
```

```
        for y in range(ys):
            for x in range(xs):
                for action in action_space:
                    state = (y, x)
                    r = self.reward_map[y, x]
                    if r != 0 and r is not None:
                        txt = 'R ' + str(r)
                        if state == self.goal_state:
                            txt = txt + ' (GOAL)'
                        ax.text(x+.05, ys-y-0.95, txt)

                    if state == self.goal_state:
                        continue

                    tx, ty = x, ys-y-1

                    action_map = {
                        0: ((0.5+tx, 0.5+ty), (tx+1, ty+1), (tx, ty+1)),
                        1: ((tx, ty), (tx+1, ty), (tx+0.5, ty+0.5)),
                        2: ((tx, ty), (tx+0.5, ty+0.5), (tx, ty+1)),
                        3: ((0.5+tx, 0.5+ty), (tx+1, ty), (tx+1, ty+1)),
                    }
                    offset_map = {
                        0: (0.1, 0.8),
                        1: (0.1, 0.1),
                        2: (-0.2, 0.4),
                        3: (0.4, 0.4),
                    }
                    if state == self.wall_state:
                        ax.add_patch(plt.Rectangle((tx, ty), 1, 1, fc=(0.4, 0.4, 0.4, 1.)))
                    elif state in self.goal_state:
                        ax.add_patch(plt.Rectangle((tx, ty), 1, 1, fc=(0., 1., 0., 1.)))
                    else:
```

```
                        tq = q[(state, action)]
                        color_scale = 0.5 + (tq / qmax) / 2    # normalize: 0.0-1.0

                        poly = plt.Polygon(action_map[action],fc=cmap(color_scale))
                        ax.add_patch(poly)

                        offset= offset_map[action]
                        ax.text(tx+offset[0], ty+offset[1], "{:12.2f}".format(tq))
        plt.show()

        if show_greedy_policy:
            policy = {}
            for y in range(self.ys):
                for x in range(self.xs):
                    state = (y, x)
                    qs = [q[state, action] for action in range(4)]   # action_size
                    max_action = np.argmax(qs)
                    probs = {0:0.0, 1:0.0, 2:0.0, 3:0.0}
                    probs[max_action] = 1
                    policy[state] = probs
            self.render_v(None, policy)
```

실행화면

2. Gridworld.py

코드

```
import numpy as np
import gridworld_render as render_helper



class GridWorld:
    def __init__(self):
```

```
        self.action_space = [0, 1, 2, 3]  # 행동 공간(가능한 행동들)
        self.action_meaning = {  # 행동의 의미
            0: "UP",
            1: "DOWN",
            2: "LEFT",
            3: "RIGHT",
        }

        self.reward_map = np.array(  # 보상 맵(각 좌표의 보상 값) 5*5에 대한 정보
            [[0, 0, 0, -1.0, 1.0],
             [0, 0, 0,      0,    0],
             [0, None, None, 0, 0],
             [0, 0, 0, 0, -1.0],
             [0, 0, 0, 0, 0]]
        )
        self.goal_state = (0, 4)      # 목표 상태(좌표)
        self.wall_state = {(2, 1), (2, 2)}      # 벽 상태(좌표)
        self.start_state = (4, 0)    # 시작 상태(좌표)
        self.agent_state = self.start_state    # 에이전트 초기 상태(좌표)

    @property
    def height(self):
        return len(self.reward_map)

    @property
    def width(self):
        return len(self.reward_map[0])

    @property
    def shape(self):
        return self.reward_map.shape

    def actions(self):
        return self.action_space
```

```
def states(self):
    for h in range(self.height):
        for w in range(self.width):
            yield (h, w)

def next_state(self, state, action):
    # 이동 위치 계산
    action_move_map = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    move = action_move_map[action]
    next_state = (state[0] + move[0], state[1] + move[1])
    ny, nx = next_state

    # 이동한 위치가 그리드 월드의 테두리 밖이나 벽인가?
    if nx < 0 or nx >= self.width or ny < 0 or ny >= self.height:
        next_state = state
    elif next_state == self.wall_state:
        next_state = state

    return next_state  # 다음 상태 반환

def reward(self, state, action, next_state):
    r = self.reward_map[next_state]
    return 0.0 if r is None else r

def reset(self):
    self.agent_state = self.start_state
    return self.agent_state

def step(self, action):
    state = self.agent_state
    next_state = self.next_state(state, action)
    reward = self.reward(state, action, next_state)
    done = (next_state == self.goal_state)
```

```
        self.agent_state = next_state
        return next_state, reward, done

    def is_wall(self, state):
        return self.reward_map[state] is None

    def render_v(self, v=None, policy=None, print_value=True):
        renderer = render_helper.Renderer(self.reward_map, self.goal_state,
                                          self.wall_state)
        renderer.render_v(v, policy, print_value)

    def render_q(self, q=None, print_value=True):
        renderer = render_helper.Renderer(self.reward_map, self.goal_state,
                                          self.wall_state)
        renderer.render_q(q, print_value)
```

실행화면

3. policeyEval.py

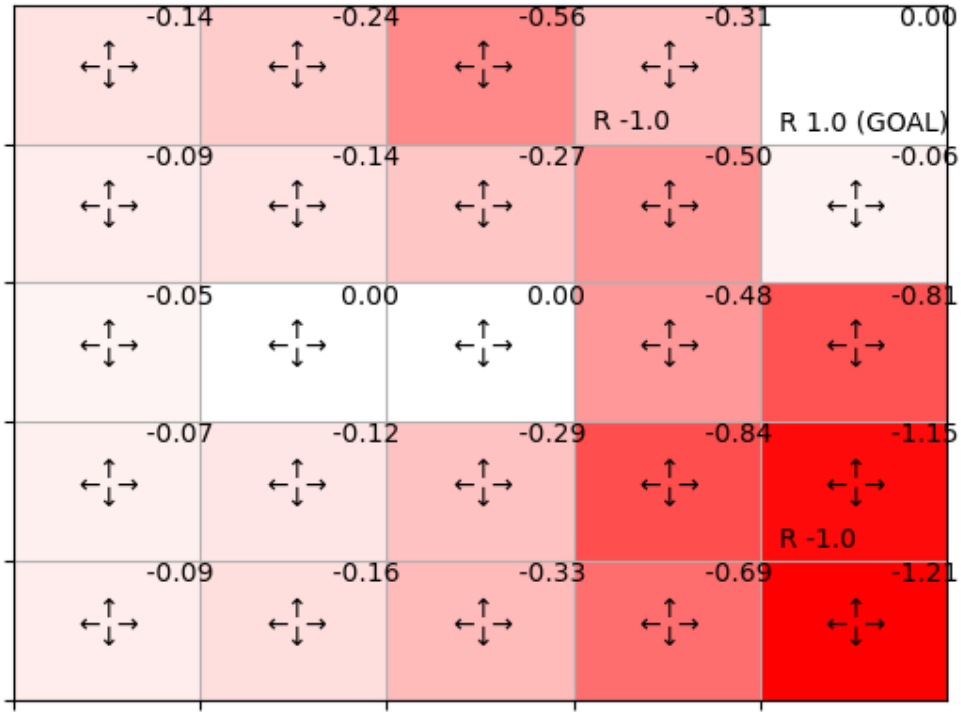
코드

```
from collections import defaultdict
from gridworld import GridWorld
#from homework.gridworld import GridWorld

def eval_onestep(pi, V, env, gamma = 0.9):
    for state in env.states():
        if state == env.goal_state:
            V[state] = 0
            continue

        if env.reward_map[state] is None:
```

```
                continue

            action_probs = pi[state]
            new_V = 0

            for action,action_prob in action_probs.items():
                next_state = env.next_state(state, action)
                r = env.reward(state, action, next_state)

                if r is None:
                    r = 0.0

                new_V += action_prob * ( r+ gamma * V[next_state])

            V[state] = new_V
        return V

def policyEval(pi, V, env, gamma, threshold = 0.001):
        while True:
            old_V = V.copy()
            V = eval_onestep(pi, V, env, gamma)

            delta = 0
            for state in V.keys():
                t = abs(V[state] - old_V[state])
                if delta < t:
                    delta = t

            if delta < threshold:
                break
        return V

if __name__ == '__main__':
        env = GridWorld()
```

```
gamma = 0.9

pi = defaultdict(lambda: {0: 0.25, 1:0.25, 2:0.25, 3:0.25})
V = defaultdict(lambda: 0)

V = policyEval(pi, V, env, gamma)

env.render_v(V, pi)
```

실행화면



4. policyIter.py

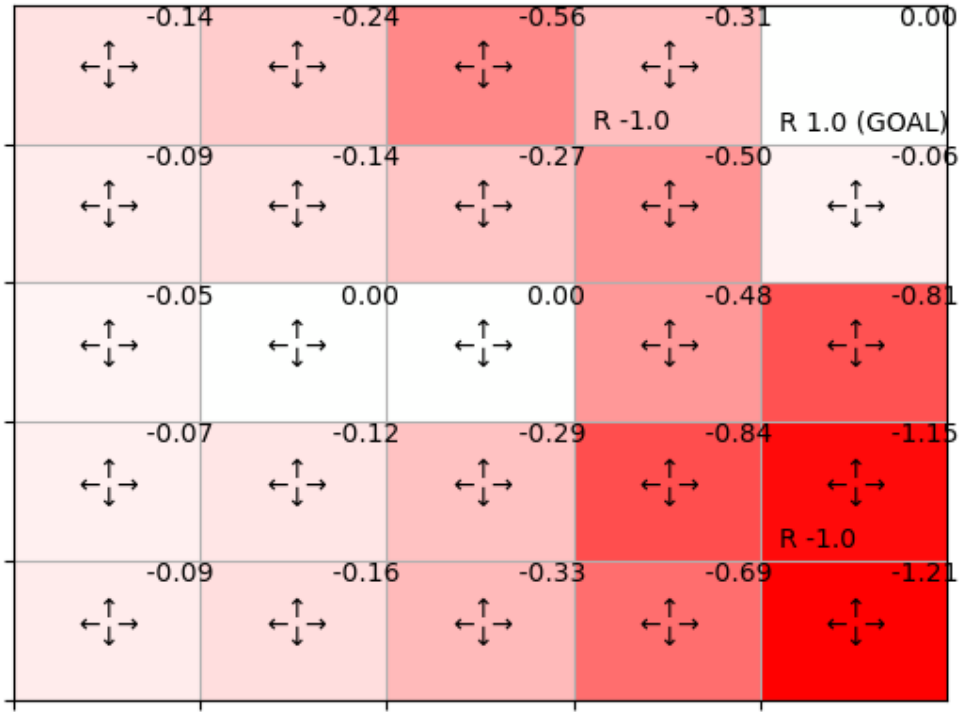
코드

```
from collections import defaultdict
from gridworld import GridWorld
```

```
from policyEval import policyEval


def argmax(d):
    """d (dict)"""
    max_value = max(d.values())
    max_key = -1
    for key, value in d.items():
        if value == max_value:
            max_key= key


    return max_key

def greedyPolicy(V, env, gamma):
    pi = {}

    for state in env.states():
        if env.is_wall(state):
            continue

        action_values = {}

        for action in env.actions():
            next_state = env.next_state(state, action)
            r = env.reward(state, action, next_state)
            value = r + gamma * V[next_state]
            action_values[action] = value

        max_action = argmax(action_values)
        action_probs = {0: 0, 1: 0, 2: 0, 3:0}
        action_probs[max_action] = 1.0
        pi[state] = action_probs
    return pi
```

```
def policyIter(env, gamma, threshold=0.001, is_render = True):
    pi = defaultdict(lambda: {0: 0.25, 1: 0.25, 2: 0.25, 3: 0.25})
    V = defaultdict(lambda: 0)

    while True:
        V = policyEval(pi, V, env, gamma, threshold)
        new_pi = greedyPolicy(V, env, gamma)

        if is_render:
            env.render_v(V, pi)

        if new_pi == pi:
            break
        pi = new_pi
    return pi



if __name__ == '__main__':
    env = GridWorld()
    gamma = 0.9

    pi = policyIter(env, gamma)
```

실행화면



5. vaslueIter.py

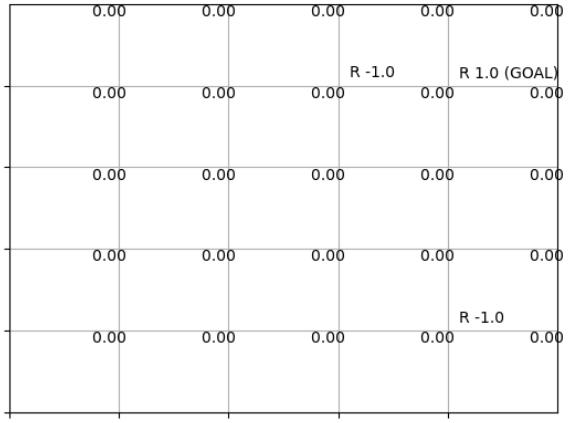
코드

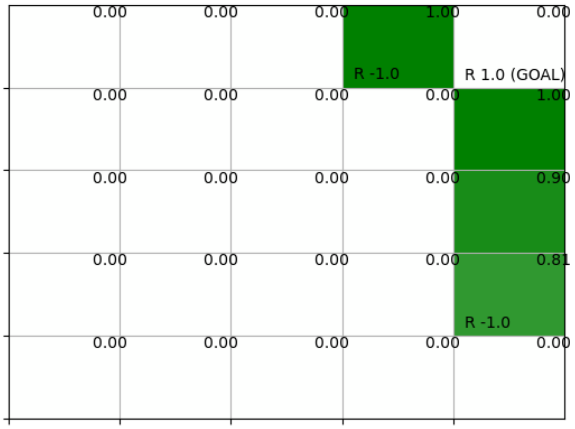
from collections import defaultdict
from gridworld import GridWorld
from policyIter import greedyPolicy, policyIter
 	
def value_iter_onestep(V, env, gamma):
  for state in env.states():
  if state == env.goal_state:
  V[state] = 0
  continue
 	
  action_values = []

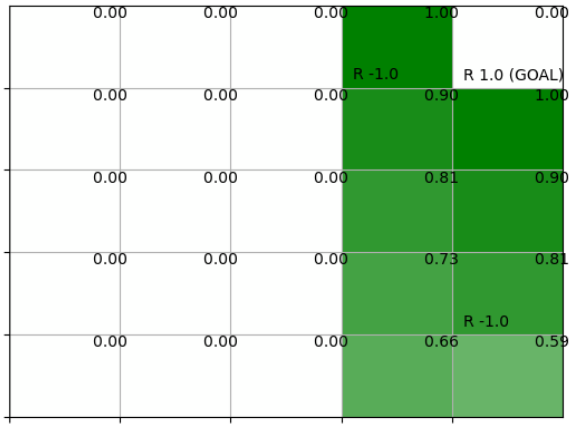
```
            for action in env.actions():
                next_state = env.next_state(state, action)
                r = env.reward(state, action, next_state)
                value = r + gamma * V[next_state]
                action_values.append(value)

            V[state] = max(action_values)
        return V

def value_iter(V, env, gamma, threshold = 0.001, is_render = True):
        while True:
            if is_render:
                env.render_v(V)
            old_V = V.copy()
            V = value_iter_onestep(V, env, gamma)

            delta = 0

            for state in V.keys():
                t = abs(V[state] - old_V[state])
                if delta < t:
                    delta = t

            if delta < threshold:
                break
        return V



if __name__ == '__main__':
        V = defaultdict(lambda: 0)
        env = GridWorld()
        gamma = 0.9
```

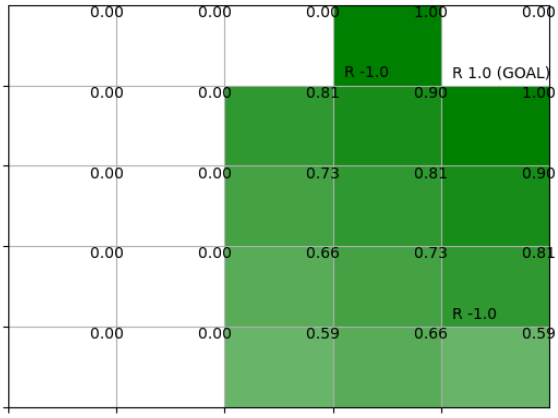
```
V = value_iter(V, env, gamma)

pi = policyIter(env, gamma)
env.render_v(V, pi)
```

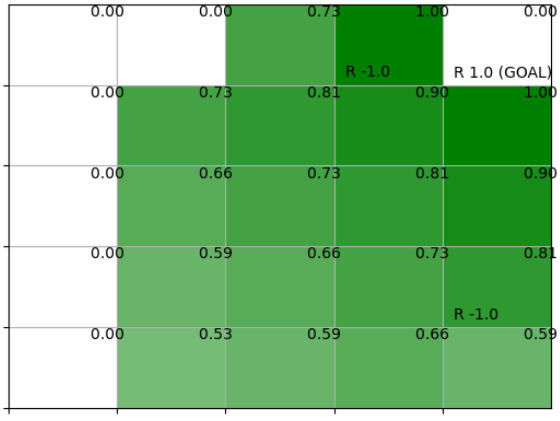
실행화면

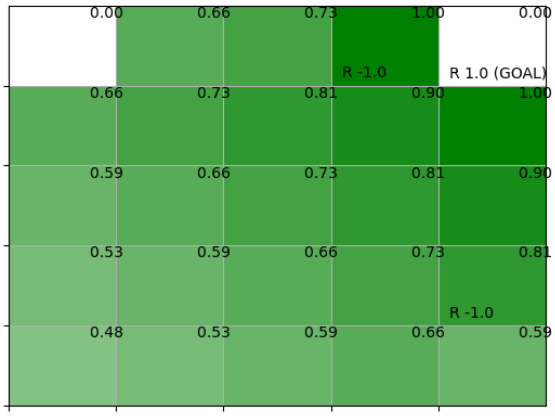


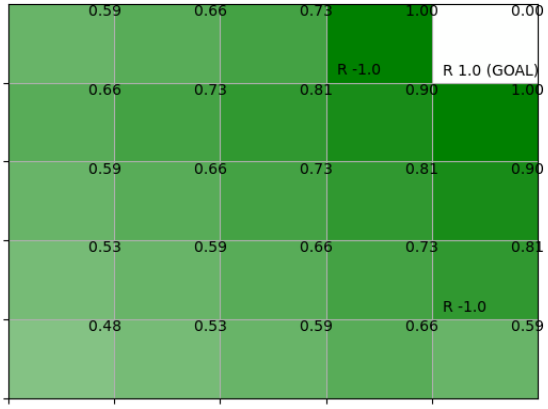



0.00 0.00 0.00 1.00 0.00
R -1.0
R 1.0 (GOAL)
0.00 0.00 0.00 0.90 1.00
0.00 0.00 0.00 0.81 0.90
0.00 0.00 0.00 0.73 0.81
R -1.0
0.00 0.00 0.00 0.66 0.59


0.00 0.00 0.00 1.00 0.00
R -1.0
R 1.0 (GOAL)
0.00 0.00 0.81 0.90 1.00
0.00 0.00 0.73 0.81 0.90
0.00 0.00 0.66 0.73 0.81
R -1.0
0.00 0.00 0.59 0.66 0.59


0.00 0.00 0.73 1.00 0.00
R -1.0
R 1.0 (GOAL)
0.00 0.73 0.81 0.90 1.00
0.00 0.66 0.73 0.81 0.90
0.00 0.59 0.66 0.73 0.81
R -1.0
0.00 0.53 0.59 0.66 0.59


0.00 0.66 0.73 1.00 0.00
R -1.0
R 1.0 (GOAL)
0.66 0.73 0.81 0.90 1.00
0.59 0.66 0.73 0.81 0.90
0.53 0.59 0.66 0.73 0.81
R -1.0
0.48 0.53 0.59 0.66 0.59


0.59 0.66 0.73 1.00 0.00
R -1.0
R 1.0 (GOAL)
0.66 0.73 0.81 0.90 1.00
0.59 0.66 0.73 0.81 0.90
0.53 0.59 0.66 0.73 0.81
R -1.0
0.48 0.53 0.59 0.66 0.59