Jack's Capitalism Optimization

Exercise 4.9: Original Problem

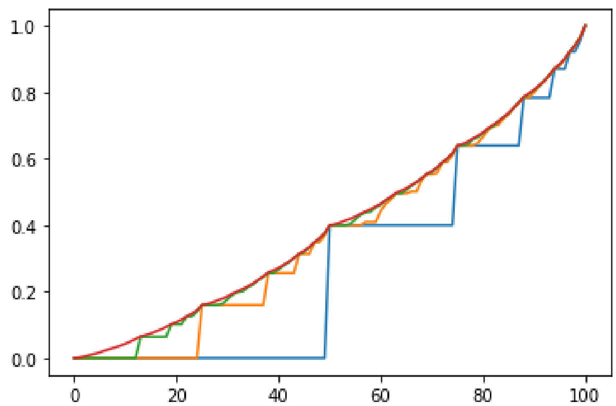

Figure 1: Value

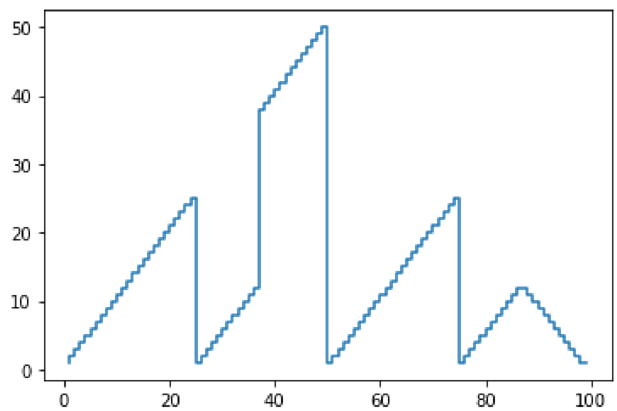

Figure 2: Policy

Exercise 4.9: ph = 0.25

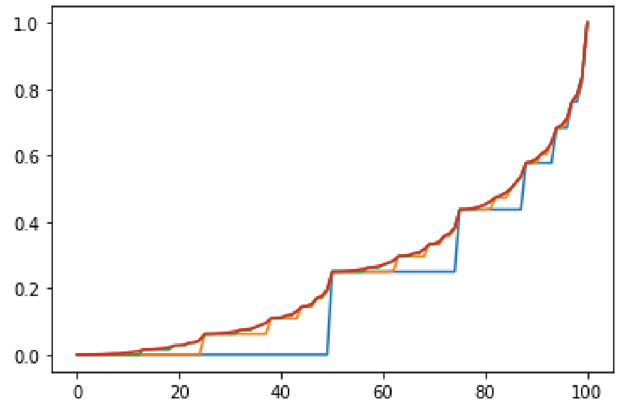

Figure 3: Value

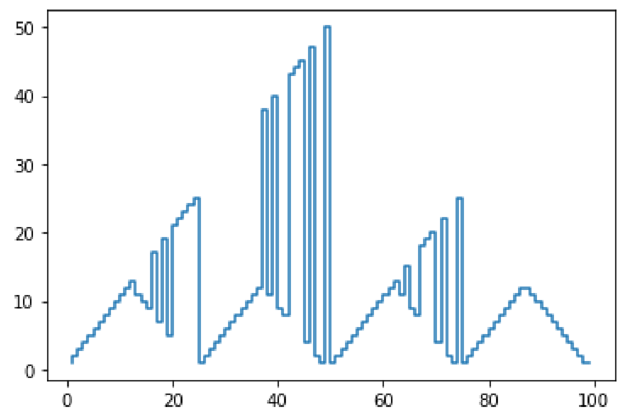

Figure 4: Policy

Exercise 4.9: ph = 0.55

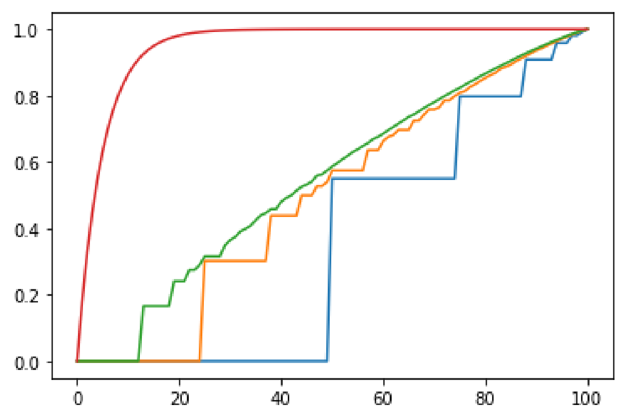

Figure 5: Value

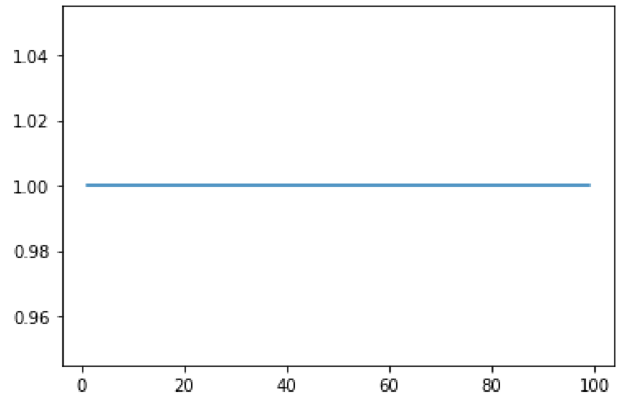

Figure 6: Policy

Appendix

Python Code for Problem 1

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 23 09:06:23 2023

@author: justi
"""

import numpy as np
import matplotlib.pyplot as plt
import random

# Value Iteration for the Gambler's problem

# %% Gambler function

class Gambler:
    def __init__(self, ph):
        self.ph = ph
        self.S = np.arange(1, 100)
        self.V = np.zeros(101)
        self.V[0] = 0
        self.V[100] = 1
        self.Vs = []
        self.pi = None
        self.sweep_count = None

    def valueIteration(self):
        self.sweep_count = 0
        while True:
            delta = 0
            for s in self.S:
                v = self.V[s]
                self.V[s] = np.max([self.V_eval(s, a) for a in self.A(s)])
                delta = np.maximum(delta, abs(v - self.V[s]))
            if self.sweep_count < 3:
                self.Vs.append(self.V.copy())
            self.sweep_count += 1
            if delta < 1E-10:
                break
        print('Sweeps needed:', self.sweep_count)
        self.Vs.append(self.V.copy())
        self.pi = [self.A(s)[np.argmax([self.V_eval(s, a) for a in self.A(s)])] for s in
    self.S]
        plt.figure()
        plt.plot(self.Vs[0])
        plt.plot(self.Vs[1])
        plt.plot(self.Vs[2])
        plt.plot(self.Vs[3])
        plt.figure()
        plt.step(self.S, self.pi)

    def A(self, s):
        return np.arange(1, np.minimum(s, 100 - s) + 1)

    def V_eval(self, s, a):
        return 1 * self.V[s + a] * self.ph + 1 * self.V[s - a] * (1 - self.ph)

Orig = Gambler(0.4)
Orig.valueIteration()

Low = Gambler(0.25)
Low.valueIteration()

High = Gambler(0.55)
High.valueIteration()
```

Python Code for Problem 2

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 18 10:14:27 2023

@author: justi
"""

import numpy as np
import matplotlib.pyplot as plt
import random
import math

states = [(x,y) for x in range(2) for y in range(5)]
policy = np.zeros((2,5), dtype=np.int8)
values = np.zeros((2,5), dtype=np.float16)
reward = np.array([500, 600, 700, 800, 1000])
prob = np.array([1/4, 1/4, 1/6, 1/6, 1/6])
gamma = 0.97

def sprimes(state, action):
    bought = state[0]
    price = state[1]

    if bought == 0:
        if action == 0:
            sprimes = [(0,y) for y in range(5)]
        elif action == 1:
            sprimes = [(1,y) for y in range(5)]
    elif bought == 1:
        sprimes = [(1,y) for y in range(5)]

    return sprimes

def probs(sprime, state, action):
    bought = state[0]
    price =  state[1]
    bought2 = sprime[0]

    if bought + action == bought2:
        probs = prob[price]
    else:
        probs = 0

    return probs

def rewards(state, action):
    bought = state[0]
    price = state[1]

    if bought == 0:
        if action == 0:
            rewards = -60
        elif action == 1:
            rewards = reward[price]
    elif bought == 1:
        rewards = 0

    return rewards

def possibleActions(state):
    bought = state[0]
    price = state[1]

    if bought == 0:
        actions = [0,1]
    else:
        actions = [0]

    return actions

```

```python

def evalVal(state, action):
    value = 0
    psa = sprimes(state, action)

    r = rewards(state,action)
    for sp in range(len(psa)):
        p = probs(psa[sp], state, action)
        nv = values[psa[sp]]
        value +=  p *( r + gamma * nv )
    return value

theta = 1E-5
maxIte = 10000
ite = 0
print("Init Values")
print(values)
print("Init Policy")
print(policy)

def policyEvaluation():
    while True:
        delta = 0
        for s in range(len(states)):
            b, p = states[s]
            v = values[b,p]
            #breakpoint()
            values[b,p] = evalVal(states[s], policy[b,p])
            delta = max(delta, abs(v - values[b,p]))
        if delta < theta:
            break

def policyImprovement():
    vvalues = {a: evalVal(s,a) for a in possibleActions(s)}
    bestActions = [a for a,value in vvalues.items() if value == np.max(list(vvalues.values()))]
    #print(bestActions)
    policy[s] = np.random.choice(bestActions)

while True:
    print(f"ite: {ite}")
    print(f"policy: {policy[0]}")
    ite += 1
    policyEvaluation()
    print(f"values: {values[0]}")

    policy_stable = True
    for s in range(len(states)):
        b, p = states[s]
        old = policy[b,p].copy()
        vvalues = {a: evalVal([b,p],a) for a in possibleActions([b,p])}
        bestActions = [a for a,value in vvalues.items() if value == np.max(list(vvalues.values()))]
        #print(bestActions)
        policy[b,p] = np.random.choice(bestActions)

        if old != policy[b,p]:
            policy_stable = False
    if policy_stable:
        break


print("Final Values")
print(values)
print("Final Policy")
print(policy)
```

Python Code for Problem 3

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 18 10:14:27 2023

@author: justi
"""

import numpy as np
import matplotlib.pyplot as plt
import random
import math

states = np.arange(0,4)
policy = np.array([0,0,0,1])
values = np.zeros(4, dtype=np.float16)
prob = np.array([[0, 7/8, 1/16, 1/16],
                 [0, 3/4, 1/8 , 1/8 ],
                 [0,   0, 1/2 , 1/2 ],
                 [0,   0,    0, 1   ]])
gamma = 0.95

rewards = np.array([[0, -1000, -3000,     0],
                    [0,     0, -2000, -6000]])

def possibleActions(state):
    if state == 2:
        actions = [0,1]
    elif state == 3:
        actions = [1]
    else:
        actions = [0]
    return actions

def sprimes(state, action):
    if state == 2 and action == 1:
        statePrimes = [1]
    elif state == 3 and action == 1:
        statePrimes = [0]
    else:
        statePrimes = [0,1,2,3]

    return statePrimes

"""
def evalVal(state, action):
    value = 0
    r = rewards[action, state]
    for i in range(4):
        p = prob[state, i]
        value += p * (r + gamma * values[i])

    return value
"""

def evalVal(state, action):
    value = 0
    psa = sprimes(state, action)

    r = rewards[action, state]
    for sp in range(len(psa)):
        p = prob[state, psa[sp]]
        nv = values[psa[sp]]
        value +=  p *( r + gamma * nv )
    return value

theta = 1E-2
maxIte = 10000
ite = 0
print("Init Values")
print(values)
```

```
print("Init Policy")
print(policy)

def policyEvaluation():
    while True:
        print(values)
        delta = 0
        for s in range(len(states)):
            v = values[s]
            values[s] = evalVal(states[s], policy[s])
            delta = max(delta, abs(v - values[s]))
        if delta < theta:
            break

#policyEvaluation()

while True:
    print(f"ite: {ite}")
    print(f"policy: {policy}")
    ite += 1
    policyEvaluation()
    print(f"values: {values}")

    policy_stable = True
    for s in range(len(states)):
        old = policy[s].copy()
        vvalues = {a: evalVal(s,a) for a in possibleActions(s)}
        bestActions = [a for a,value in vvalues.items() if value == np.max(list(vvalues.values()))]
        policy[s] = np.random.choice(bestActions)

        if old != policy[s]:
            policy_stable = False
    if policy_stable:
        break

print("Final Values")
print(values)
print("Final Policy")
print(policy)
```

Python Code for Problem 4

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 23 23:12:10 2023

@author: justi
"""

import numpy as np
import matplotlib.pyplot as plt
import random
import math

states = np.arange(3)
policy = np.array([2,2,2])
values = np.zeros(3, dtype=np.float16)

gamma = 0.95

actions = np.array([0,1,2])

"""
def evalVal(state, action):
    value = 0
    r = rewards[action, state]
    for i in range(4):
        p = prob[state, i]
        value += p * (r + gamma * values[i])

    return value
"""

def evalVal(state, action):
    value = 0
    sprime = state + action
    cost = action * 5 #+ 10*np.sign(action)

    for i in range(3):
        nv = sprime-i

        if nv >= 0 and nv <= 2:
            r = -4 * nv
            nvalue = values[nv]
        elif nv == -1:
            r = -8
            nvalue = 0
        elif nv == -2:
            r = -40
            nvalue = 0
        elif nv > 2:
            r = -8
            nvalue = values[2]

        value +=  -cost + 1/3 *( r + gamma * nvalue )
    return value

theta = 1E-2
maxIte = 10000
ite = 0
print("Init Values")
print(values)
print("Init Policy")
print(policy)

def policyEvaluation():
    while True:
        print(values)
        delta = 0
        for s in range(len(states)):
            v = values[s]
            values[s] = evalVal(states[s], policy[s])
```

```
            delta = max(delta, abs(v - values[s]))
        if delta < theta:
            break

#policyEvaluation()

while True:
    print(f"ite: {ite}")
    print(f"policy: {policy}")
    ite += 1
    policyEvaluation()
    print(f"values: {values}")

    policy_stable = True
    for s in range(len(states)):
        old = policy[s].copy()
        vvalues = {a: evalVal(s,a) for a in [0,1,2]}
        bestActions = [a for a,value in vvalues.items() if value == np.max(list(vvalues.
    values()))]
        policy[s] = np.random.choice(bestActions)

        if old != policy[s]:
            policy_stable = False
    if policy_stable:
        break

print("Final Values")
print(values)
print("Final Policy")
print(policy)
```

JUSTINE JOHN A. SERDONCILLO
IE 5571 REINFORCEMENT LEARNING
& DYNAMIC PROGRAMMING

OCTOBER 23, 2027
HOMEWORK #3

EXERCISE 4.9 (PROGRAMMING EXERCISE)
GAMBLER'S PROBLEM for $P_h = 0.25$ & $P_h = 0.55$
stable if $\theta \to 0$?

EXERCISE 5.6 WHAT IS THE EQUATION ANALOGOUS TO (5.6) FOR ACTION VALUES $Q(s,a)$ INSTEAD OF STATE VALUES $V(s)$, AGAIN GIVEN RETURNS GENERATED USING B?

not sure but now instead of

$$V(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

$\mathcal{T}(s)$ now becomes $\mathcal{T}(s,a)$

$$\boxed{Q(s,a) = \frac{\sum_{t \in \mathcal{T}(s,a)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s,a)} \rho_{t:T(t)-1}}}$$

EXERCISE 5.8 THE RESULTS W/ EX. 5.5 & SHOWN IN FIG 5.4 USED A 1st VISIT MC METHOD. SUPPOSE THAT INSTEAD AN EVERY-VISIT MC METHOD WAS USED ON THE SAME PROBLEM, WOULD THE VARIANCE OF THE ESTIMATOR STILL BE INFINITE? WHY OR WHY NOT?

---

estimator will still be infinite because reward is at terminal still.

& the visits to the state will still run to infinity

expected reward at every state = 1

PROBLEM 3 A MANUFACTURER RELIES ON ONE KEY MACHINE. DUE TO HEAVY USE, THE MACHINE DETERIORATES RAPIDLY. AT THE END OF EACH WEEK A THOROUGH INSPECTION IS DONE THAT CLASSIFIES THE MACHINE INTO ONE OF FOUR POSSIBLE STATES.

0 - GOOD AS NEW
1 - OPERABLE MINOR DETERIORATION
2 - OPERABLE MAJOR DETERIORATION
3 - INOPERABLE

WITHOUT ANY REPAIRS THE STATE OF THE MACHINE EVOLVES AS A MARKOV CHAIN W/ A TRANSITION MATRIX

$$\Pi = \begin{bmatrix} 0 & 7/8 & 1/16 & 1/16 \\ 0 & 3/4 & 1/8 & 1/8 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

IF IN STATE 3, MANUFACTURER REPLACES MACHINE & COSTS 6000$

IF IN STATE 1 & 2 $1000/week in 1

$3000/week in 2

repair $2000

USING DISCOUNT FACTOR OF $\alpha = 0.95$ FIND OPTIMAL POLICY VIA POLICY ITERATION

Stages is month

infinite problem hence discount

States: {0, 1, 2, 3}

actions: {continue, repair}

Rewards $\begin{cases} 0 & -1000 & -3000 & \times \\ 0 & 0 & -2000 & -6000 \end{cases}$

Dynamics of repair

convert to a solvable linear system for Policy Evaluation

initialize w/ {continue, continue, continue, repair}

only thing possible to change

To do policy iteration start w/ policy evaluation first

from code this evaluates to [-5188, -5832 -5772, 0]

using policy improvement we get

{continue continue repair repair}

& new values = [-2892, -3480 0, 0]

Final policy	
0	continue
1	continue
2	repair
3	repair

PROBLEM 4 CONSIDER AN INFINITE-PERIOD INVENTORY SYSTEM W/ A SINGLE PRODUCT WHERE, AT THE BEGINNING OF EACH PERIOD, A DESISION IS TO BE MADE ABOUT HOW MANY ITEMS TO PRODUCE DURING THAT PERIOD. THE SETUP COST IS $10 & THE UNIT PRODUCTION COST IS $5. THE HOLDING COST FOR EACH ITEM NOT SOLD DURING IS $4, AND A MAXIMUM OF 2 ITEMS CAN BE STORED. DURING EACH PERIOD, DEMAND IS 0,1,2 ITEMS EACH WITH PROBABILITY 1/3. IF DEMAND EXCEEDS THE SUPPLY AVAILABLE DURING THAT PERIOD, THOSE SALES ARE LOST AND A SHORTAGE COST IS INCURRED $\begin{cases} 1 \text{ UNIT: } \$8 \\ 2 \text{ UNITS: } \$32 \end{cases}$

$\alpha = 0.95$. Get optimal policy using policy iteration

$x_{k+1} = x_k - d_k + \text{prod}$
012

STATES: # of items in inventory $\{2,1,0\}$

ACTIONS: PRODUCE $\{0,1,2\}$

use code

initialize value function = $(0,0,0)$

initialize policy: produce $(2,2,2)$

policy evaluation:

not sure about code but final policy is

$$\boxed{\begin{array}{c} \{0,1,2\} \\ \text{produce } 0,0,0 \end{array}}$$

b) using policy iteration, see code attached.

Started w/ $V = \{0, 0, 0, 0, 0\}$

& π = (wait, wait, wait, wait, wait)

1st iteration policy evaluation: [-2476, -2476, -1651, -1651, -1651]

policy improvement [sell, sell, sell, sell, sell]

2nd iteration policy evaluation: [625 750 583.5 666.5 833.5]

policy improvement = [wait wait sell sell sell]

<u>Converged</u>

best policy:	$500	wait
	$600	wait
	$700	sell
	$800	sell
	$1000	sell