Assignment4

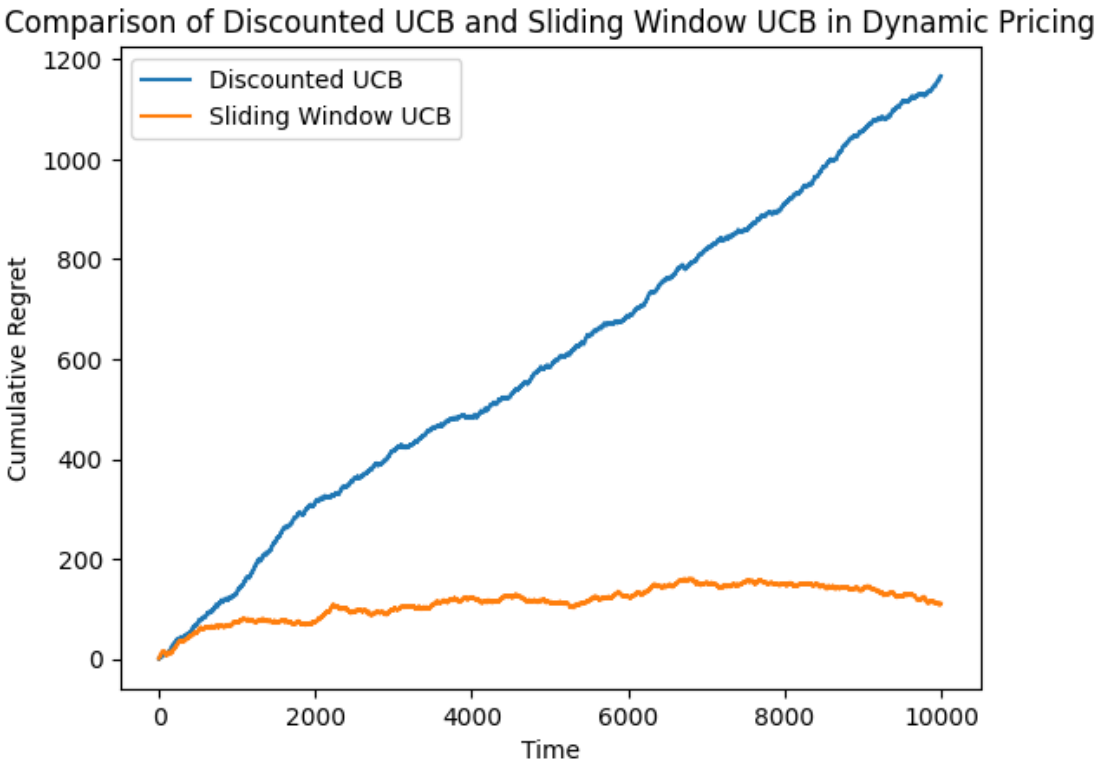
March 26, 2025

0.1 Code Assignment 1

[ ]:
```python
# Deepseek assignment 1
import numpy as np
import matplotlib.pyplot as plt

class DynamicPricingEnv:
    def __init__(self, prices, change_points, reward_probabilities):
        self.prices = prices
        self.change_points = change_points
        self.reward_probabilities = reward_probabilities
        self.current_phase = 0
        self.t = 0

    def step(self, price_index):
        if self.t in self.change_points:
            self.current_phase = min(self.current_phase + 1, len(self.
 ↪reward_probabilities) - 1)
        self.t += 1
        return np.random.rand() < self.reward_probabilities[self.
 ↪current_phase][price_index]

    def reset(self):
        self.current_phase = 0
        self.t = 0

class DiscountedUCB:
    def __init__(self, n_arms, gamma=0.99):
        self.n_arms = n_arms
        self.gamma = gamma
        self.counts = np.zeros(n_arms)
        self.values = np.zeros(n_arms)

    def select_arm(self):
        for arm in range(self.n_arms):
            if self.counts[arm] == 0:
                return arm
        total = np.sum(self.counts)
```

```
        ucb_values = (self.values / self.counts) + np.sqrt((2 * np.log(total)) /
 ↪ self.counts)
        return np.argmax(ucb_values)

    def update(self, arm, reward):
        self.counts *= self.gamma
        self.values *= self.gamma
        self.counts[arm] += 1
        self.values[arm] += reward

class SlidingWindowUCB:
    def __init__(self, n_arms, window_size=50):
        self.n_arms = n_arms
        self.window_size = window_size
        self.rewards = [[] for _ in range(n_arms)]

    def select_arm(self):
        for arm in range(self.n_arms):
            if len(self.rewards[arm]) == 0:
                return arm
        total_pulls = sum(len(rewards) for rewards in self.rewards)
        ucb_values = []
        for arm in range(self.n_arms):
            cnt = len(self.rewards[arm])
            avg = sum(self.rewards[arm]) / cnt
            ucb = avg + np.sqrt((2 * np.log(total_pulls)) / cnt)
            ucb_values.append(ucb)
        return np.argmax(ucb_values)

    def update(self, arm, reward):
        self.rewards[arm].append(reward)
        if len(self.rewards[arm]) > self.window_size:
            self.rewards[arm].pop(0)

# Experiment Setup
prices = [5, 10, 15, 20]
change_points = [2000, 4000, 6000, 8000]
reward_probabilities = [
    [0.3, 0.5, 0.2, 0.1],  # Phase 1
    [0.2, 0.6, 0.3, 0.15], # Phase 2
    [0.1, 0.4, 0.5, 0.3],  # Phase 3
    [0.25, 0.35, 0.3, 0.2], # Phase 4
    [0.15, 0.5, 0.25, 0.4]  # Phase 5
]

env = DynamicPricingEnv(prices, change_points, reward_probabilities)
d_ucb = DiscountedUCB(len(prices))
```

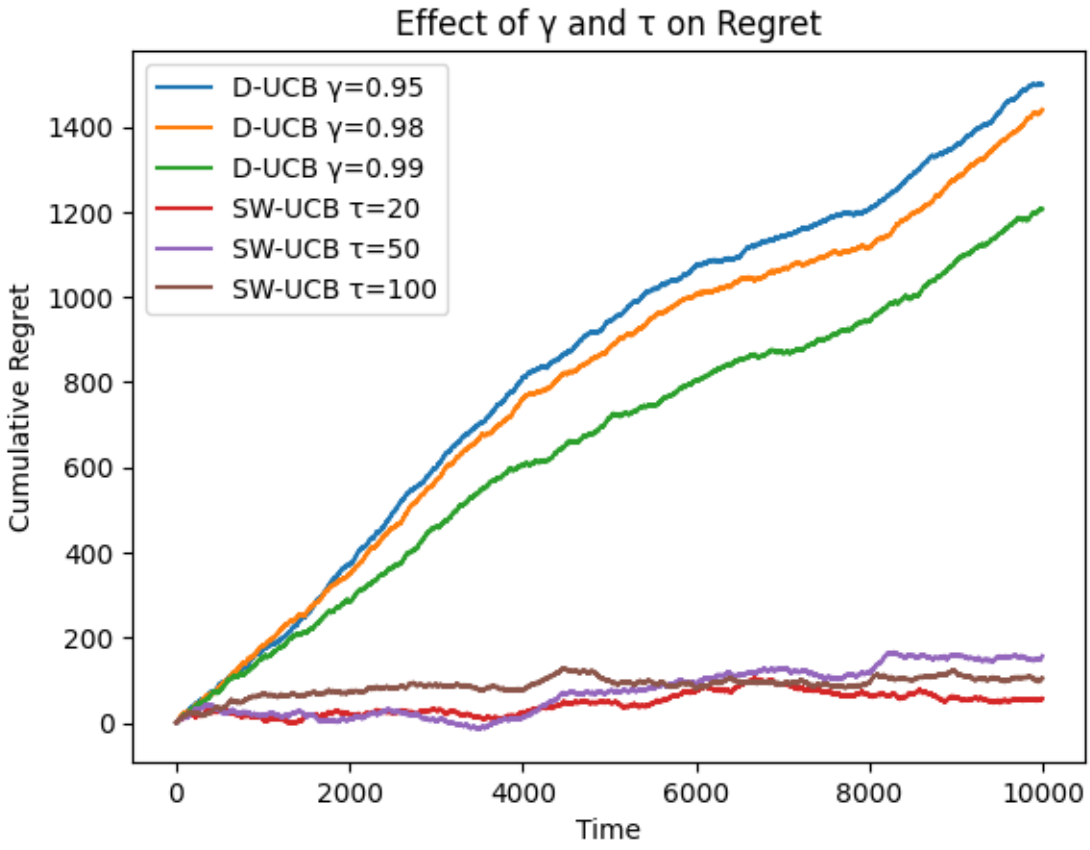
```
sw_ucb = SlidingWindowUCB(len(prices))

T = 10000  # Total time steps
regrets_d = []
regrets_sw = []

for t in range(T):
    # Discounted UCB
    arm_d = d_ucb.select_arm()
    reward_d = env.step(arm_d)
    d_ucb.update(arm_d, reward_d)
    regrets_d.append(max(reward_probabilities[env.current_phase]) - reward_d)

    # Sliding Window UCB
    arm_sw = sw_ucb.select_arm()
    reward_sw = env.step(arm_sw)
    sw_ucb.update(arm_sw, reward_sw)
    regrets_sw.append(max(reward_probabilities[env.current_phase]) - reward_sw)

# Plot Regrets
plt.plot(np.cumsum(regrets_d), label='Discounted UCB')
plt.plot(np.cumsum(regrets_sw), label='Sliding Window UCB')
plt.xlabel("Time")
plt.ylabel("Cumulative Regret")
plt.legend()
plt.title("Comparison of Discounted UCB and Sliding Window UCB in Dynamic␣
 ↪Pricing")
plt.show()
```



0.2 Assignment 4.2

[8]:
```
# ===== Q2 =====
# Q2: Test different values of gamma (D-UCB) and tau (SW-UCB)
gammas = [0.95, 0.98, 0.99]
taus = [20, 50, 100]

for gamma in gammas:
    d_ucb = DiscountedUCB(len(prices), gamma=gamma)
    env.reset()
    regrets = []
    for t in range(T):
        arm = d_ucb.select_arm()
        reward = env.step(arm)
        d_ucb.update(arm, reward)
        regrets.append(max(reward_probabilities[env.current_phase]) - reward)
    plt.plot(np.cumsum(regrets), label=f'D-UCB  ={gamma}')

for tau in taus:
    sw_ucb = SlidingWindowUCB(len(prices), window_size=tau)
    env.reset()
    regrets = []
```

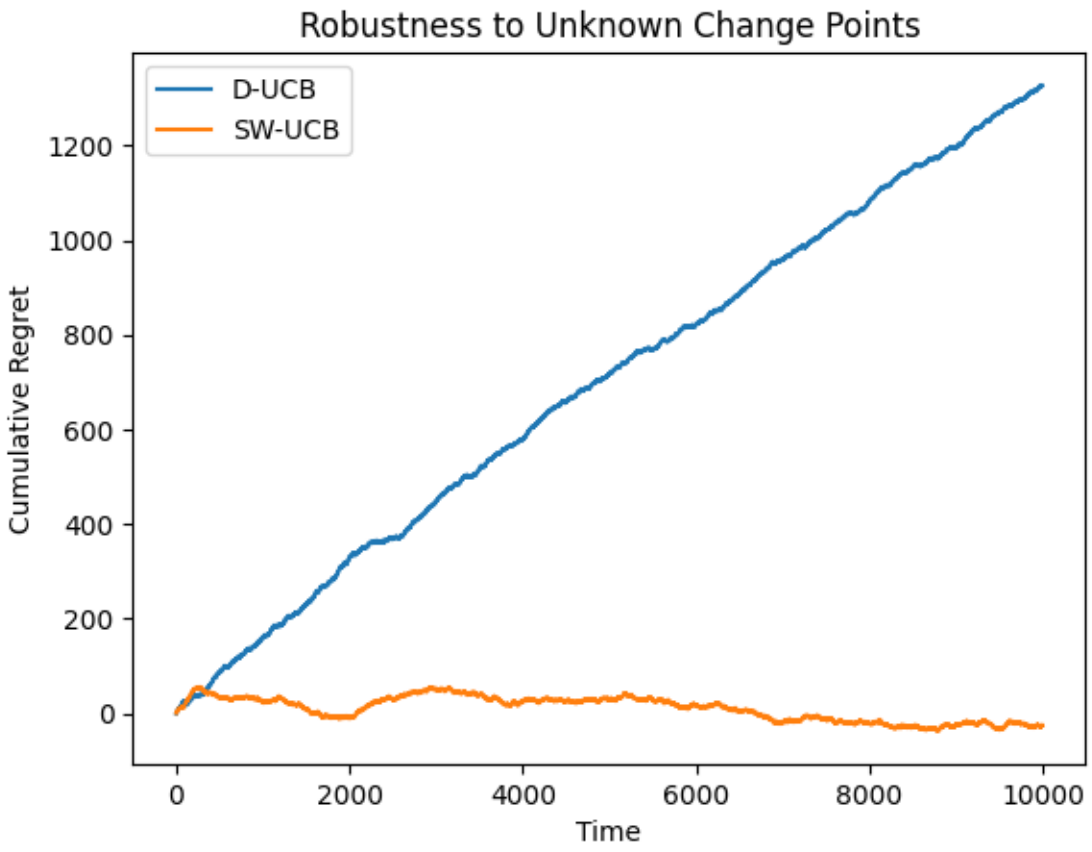
```
    for t in range(T):
        arm = sw_ucb.select_arm()
        reward = env.step(arm)
        sw_ucb.update(arm, reward)
        regrets.append(max(reward_probabilities[env.current_phase]) - reward)
    plt.plot(np.cumsum(regrets), label=f'SW-UCB  ={tau}')

plt.xlabel("Time")
plt.ylabel("Cumulative Regret")
plt.title("Effect of   and   on Regret")
plt.legend()
plt.show()
```



0.3 Assignment 4.3

[9]:
```
# ===== Q3 =====
# Q3: Irregular (unknown) change points
import random
change_points = sorted(random.sample(range(500, 9500), 4))
```

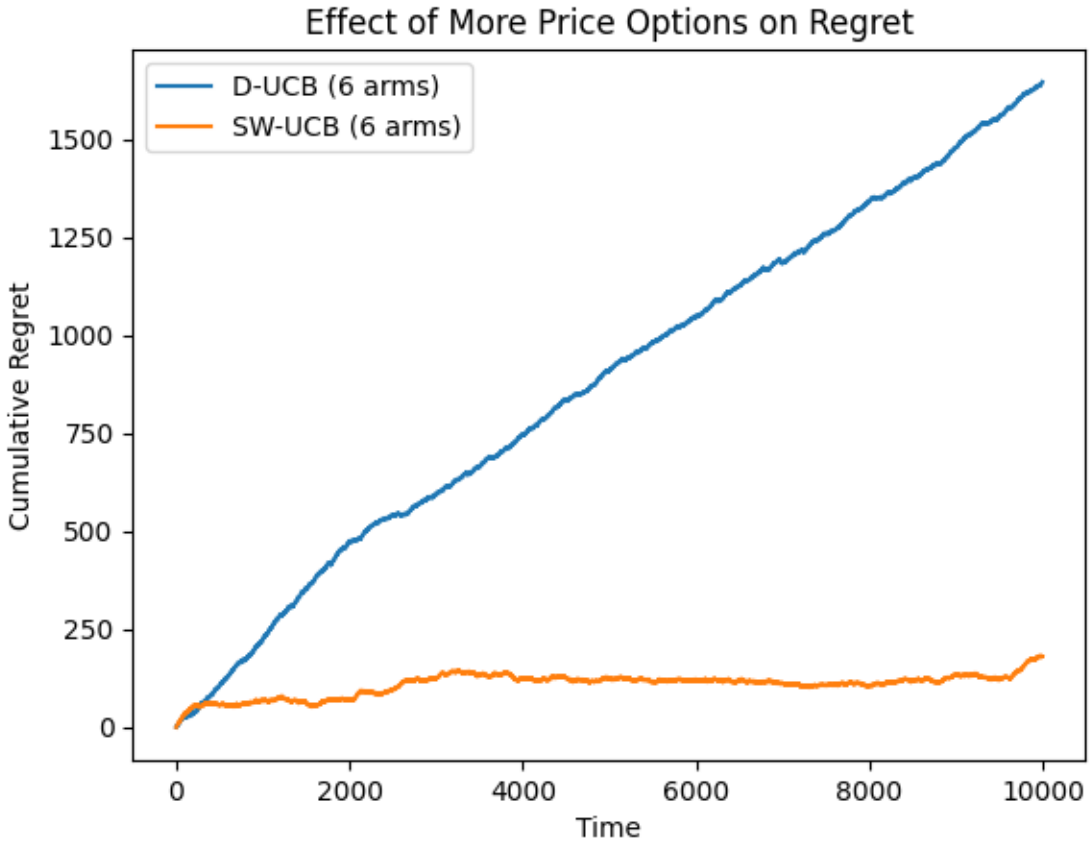
```
env = DynamicPricingEnv(prices, change_points, reward_probabilities)
sw_ucb = SlidingWindowUCB(len(prices), window_size=50)
d_ucb = DiscountedUCB(len(prices), gamma=0.99)

regrets_sw, regrets_d = [], []
for t in range(T):
    # SW-UCB
    arm_sw = sw_ucb.select_arm()
    reward_sw = env.step(arm_sw)
    sw_ucb.update(arm_sw, reward_sw)
    regrets_sw.append(max(reward_probabilities[env.current_phase]) - reward_sw)

    # D-UCB
    arm_d = d_ucb.select_arm()
    reward_d = env.step(arm_d)
    d_ucb.update(arm_d, reward_d)
    regrets_d.append(max(reward_probabilities[env.current_phase]) - reward_d)

plt.plot(np.cumsum(regrets_d), label="D-UCB")
plt.plot(np.cumsum(regrets_sw), label="SW-UCB")
plt.title("Robustness to Unknown Change Points")
plt.xlabel("Time")
plt.ylabel("Cumulative Regret")
plt.legend()
plt.show()
```



0.4 Assignment 4.4

[10]:
```
# ===== Q4 =====
# Q4: Increase number of arms
prices = [5, 10, 15, 20, 25, 30]
reward_probabilities = [
    [0.3, 0.5, 0.2, 0.1, 0.05, 0.02],
    [0.2, 0.6, 0.3, 0.15, 0.1, 0.03],
    [0.1, 0.4, 0.5, 0.3, 0.2, 0.1],
    [0.25, 0.35, 0.3, 0.2, 0.15, 0.12],
    [0.15, 0.5, 0.25, 0.4, 0.3, 0.18]
]
env = DynamicPricingEnv(prices, change_points, reward_probabilities)
d_ucb = DiscountedUCB(len(prices), gamma=0.99)
sw_ucb = SlidingWindowUCB(len(prices), window_size=50)

regrets_sw, regrets_d = [], []
for t in range(T):
    arm_sw = sw_ucb.select_arm()
```

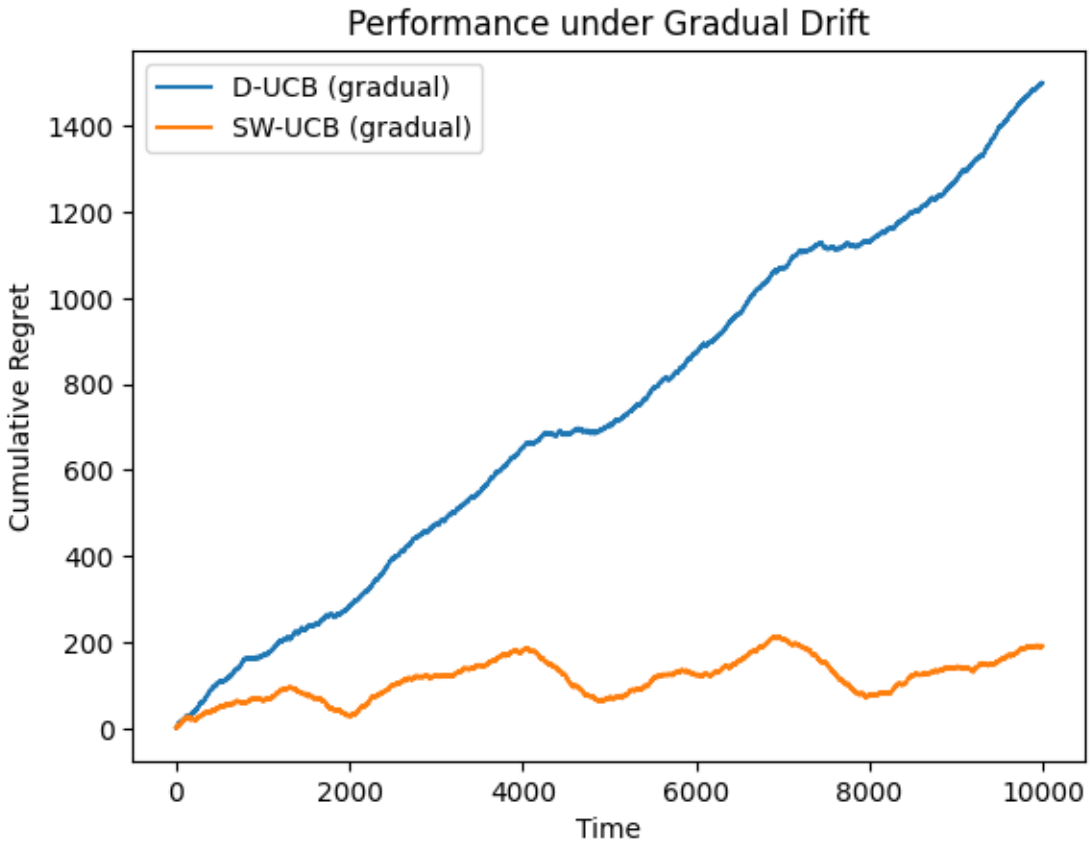
```
    reward_sw = env.step(arm_sw)
    sw_ucb.update(arm_sw, reward_sw)
    regrets_sw.append(max(reward_probabilities[env.current_phase]) - reward_sw)

    arm_d = d_ucb.select_arm()
    reward_d = env.step(arm_d)
    d_ucb.update(arm_d, reward_d)
    regrets_d.append(max(reward_probabilities[env.current_phase]) - reward_d)

plt.plot(np.cumsum(regrets_d), label="D-UCB (6 arms)")
plt.plot(np.cumsum(regrets_sw), label="SW-UCB (6 arms)")
plt.title("Effect of More Price Options on Regret")
plt.xlabel("Time")
plt.ylabel("Cumulative Regret")
plt.legend()
plt.show()
```



0.5 Assignment 4.5

[11]:
```python
# ===== Q5 =====
# Q5: Gradually changing reward probabilities
class GradualEnv:
    def __init__(self, prices):
        self.prices = prices
        self.t = 0

    def step(self, price_index):
        probs = self.get_probs(self.t)
        self.t += 1
        return np.random.rand() < probs[price_index]

    def get_probs(self, t):
        return [
            0.3 + 0.2 * np.sin(2 * np.pi * t / 5000),
            0.5 + 0.1 * np.cos(2 * np.pi * t / 3000),
            0.2 + 0.1 * np.sin(2 * np.pi * t / 2000),
            0.1 + 0.1 * np.cos(2 * np.pi * t / 2500)
        ]

    def reset(self):
        self.t = 0

env = GradualEnv(prices=[5, 10, 15, 20])
d_ucb = DiscountedUCB(4, gamma=0.99)
sw_ucb = SlidingWindowUCB(4, window_size=50)

regrets_d, regrets_sw = [], []
for t in range(T):
    probs = env.get_probs(t)

    arm_d = d_ucb.select_arm()
    reward_d = env.step(arm_d)
    d_ucb.update(arm_d, reward_d)
    regrets_d.append(max(probs) - reward_d)

    arm_sw = sw_ucb.select_arm()
    reward_sw = env.step(arm_sw)
    sw_ucb.update(arm_sw, reward_sw)
    regrets_sw.append(max(probs) - reward_sw)

plt.plot(np.cumsum(regrets_d), label="D-UCB (gradual)")
plt.plot(np.cumsum(regrets_sw), label="SW-UCB (gradual)")
plt.title("Performance under Gradual Drift")
plt.xlabel("Time")
```

```
plt.ylabel("Cumulative Regret")
plt.legend()
plt.show()
```



[13]:
```
!jupyter nbconvert --to pdf Assignment4.ipynb
```

```
[NbConvertApp] Converting notebook Assignment4.ipynb to pdf
[NbConvertApp] Support files will be in Assignment4_files\
[NbConvertApp] Making directory .\Assignment4_files
[NbConvertApp] Writing 41481 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | b had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 212257 bytes to Assignment4.pdf
```