The code for the algorithms can be found in the `RL-book/Assignment13/assignment13_code.py` file.

We have implemented both the Tabular versions and the Function Approximation version in each case.

In the tabular case, we notice that all our solutions yield the same optimal policy.

As for the value functions (derived from the Q value functions), we get that our solutions are quite similar to the true value function.

We created a class to define the state. This class which we called `StateSnakeAndLadder` only has one attribute: it is the position.

```
Testing Tabular Versions
MDP Value Iteration Optimal Value Function and Optimal Policy
--------------
{InventoryState(on_hand=2, on_order=0): -29.991890076067463,
 InventoryState(on_hand=1, on_order=1): -28.991890076067467,
 InventoryState(on_hand=1, on_order=0): -28.660950216301437,
 InventoryState(on_hand=0, on_order=2): -27.991890076067463,
 InventoryState(on_hand=0, on_order=1): -27.66095021630144,
 InventoryState(on_hand=0, on_order=0): -34.89484576629397}
For State InventoryState(on_hand=0, on_order=0):
  Do Action 1 with Probability 1.000
For State InventoryState(on_hand=0, on_order=1):
  Do Action 1 with Probability 1.000
For State InventoryState(on_hand=0, on_order=2):
  Do Action 0 with Probability 1.000
For State InventoryState(on_hand=1, on_order=0):
  Do Action 1 with Probability 1.000
For State InventoryState(on_hand=1, on_order=1):
  Do Action 0 with Probability 1.000
For State InventoryState(on_hand=2, on_order=0):
  Do Action 0 with Probability 1.000


Solving Problem 1
MC Control Algorithm
{InventoryState(on_hand=0, on_order=0): -35.53865711436859,
InventoryState(on_hand=0, on_order=1): -27.931812642376766,
InventoryState(on_hand=0, on_order=2): -28.37923187981179,
InventoryState(on_hand=1, on_order=0): -28.94998799196226,
InventoryState(on_hand=1, on_order=1): -29.371288577266103,
InventoryState(on_hand=2, on_order=0): -30.389138052171038}
Solving Problem 2
Sarse Control Algorithm
{InventoryState(on_hand=0, on_order=0): -36.19919667140732,
InventoryState(on_hand=0, on_order=1): -29.028949532957707,
InventoryState(on_hand=0, on_order=2): -29.424212080576947,
InventoryState(on_hand=1, on_order=0): -29.974676076398165,
InventoryState(on_hand=1, on_order=1): -30.57861407845046,
InventoryState(on_hand=2, on_order=0): -31.208958432637914}
Solving Problem 3
```

```
Tabular Q-Learning Control Algorithm
{InventoryState(on_hand=0, on_order=0): -34.94239946200996,
InventoryState(on_hand=0, on_order=1): -27.758349159510125,
InventoryState(on_hand=0, on_order=2): -27.816660153477148,
InventoryState(on_hand=1, on_order=0): -28.690177791518888,
InventoryState(on_hand=1, on_order=1): -29.120850025524785,
InventoryState(on_hand=2, on_order=0): -30.04577043137654}
```

CME241: Assignment 13 : Problem 4:

Let us model this problem as a MDP.

Since there are no transaction costs associated with buying or selling shares and with borrowing cash, everything is as if we had to choose at the end of each day:

- Sell the quantity of stock from the previous day
- choose to increase or decrease the quantity of cash borrowing
- Choose to purchase a certain quantity of stock.

The state is observed after we sell our stock from the previous day. It consists of several elements:

- the time $t \in [1, T]$
- the net cash in the bank after selling stock: $x_t \in \mathbb{R}_+$
- the money we owe to people to whom we borrowed: $l_t \in \mathbb{R}_+$
- the quantity of withdrawal we were not able to fulfil in the day: $b_t \in \mathbb{R}_+$
- the price of a stock $s_t \in \mathbb{R}_+$

(all quantities are continuous)

Let c_t be the cash at the start of each day, we always need to keep enough cash to pay for the regulator's fine: hence: $c_t \geq K \cot\left(\frac{\pi c_{min}}{2C}\right)$ where $c_{min} = K \cot\left(\frac{\pi c_{min}}{2C}\right)$

Let y_t be the decrease or increase in liability: it is positive when we borrow and negative when we pay liability back: meaning that $y_t \geq -l_t$: we cannot repay more than what we owe.

Besides, our net cash cannot go below c_{min} $x_t + y_t \geq c_{min}$

This implies that $y_t \geq \max(-l_t, c_{min} - x_t)$

Let z_t be the number of shares of stock to purchase: net cash cannot fall below c_{min}. Hence:

$x_t + y_t - z_t s_t \geq c_{min}$ (we choose to purchase stock after borrowing).

Hence $0 \leq z_t \leq \frac{x_t + y_t - c_{min}}{s_t}$

In this MDP, we choose the Action as the pair (y_t, z_t) with the action space defined for $t < T$ by the constraints we wrote above for y_t and z_t.

Let us now specify state transitions. We need to define variables for where randomness comes in the model.

- Let deposits on day t be $d_t \in \mathbb{R}_+$
- For withdrawal requests: $w_t = f(t, b_{t-1})$ for $2 \leq t \leq T$ where f describes the usual random withdrawal requests as well as the previous day's unfulfilled requests.
- The value of a share of stock is denoted by $s_t = g(t, s_{t-1})$ where g describes the stochastic movement of the stock price from one day to the next.

We now have $x_{t+1} = z_t \cdot g(t+1, s_t) + \max\left(x_t + y_t - z_t s_t - K \cot\left(\frac{\pi \min(x_t + y_t - z_t s_t, C)}{2C}\right) + d_{t+1} - f(t+1, b_t), 0\right)$

$l_{t+1} = (l_t + y_t)(1 + R)$ and $s_{t+1} = g(t+1, s_t)$

last, we have that:

$$b_{t+1} = \max\left(-\left(x_t + y_t - z_t S_t - K \cdot \cot\left(\frac{\pi \cdot \min(x_t + y_t - z_t S_t, C)}{2C}\right) + d_{t+1} - f(t+1, b_t)\right), 0\right)$$

As for the reward, it is 0 for day $0 \le t \le T-1$ and on day T it is $U(x_T - l_T)$

The way we count on solving this problem with reinforcement learning is the following.
The sources of randomness here are deposits, withdrawals as well as daily stock price movements.
We will thus generate historical data about it.
From this, we will be able to build simulation episodes, that is to say lists of successive states and rewards: we will be able using our historical data to predict the future movements of some of the factors like (or their probabilities) future deposits, withdrawals and stock moves: and from this we can get a large set of simulation episodes with appropriate sampling of the probability distributions.
Since our state space is large, as well as our action space, in our control algorithm, to find for the optimal policy, we will need to use a good function approximation of the Q-value function.
As for the actions possible at a given state, we would need to be able to efficiently determine a policy, given that there are lots of actions possible.