Project 4: Online Revenue Maximization

Introduction

In this project, we analyze the truthful second-price auction with reserve for selling a single-item in each round using online learning algorithms to optimize the reserve price and revenue. We then compare our findings to the theoretical optimal reserve price and revenue given parameters of the repeated auction. For instance, when using distribution $F \sim U[0,1]$ for a truthful second price auction with 2 bidders and 1 item, our online learning algorithm predicted the optimal reserve price to be 1/2 and the optimal revenue to be 5/12, which agrees with the respective theoretical values.

In addition, we also explore variations in this scenario including different distributions of bidders' values, different numbers of bidders in each auction, and different number of items for sale each round. We expected that optimal reserve price would not change as we varied these parameters and that revenue would increase as we increased the number of bidders or items. Our results confirmed these predictions.

In the second part of our project, we study the mechanism design of selling introductions. The motivation of studying such scenarios is that observations learned can be applied to situations such as employee matching as well as online dating matching. In this study, we assign different distributions to two players of which their value of meeting the other person is drawn from. Then we use Exponential Weights online learning algorithm to try to determine the optimal truthful mechanism for each scenario. Our results were very satisfactory and we found that the online learning algorithm was able to quickly find the optimal reserve price for the sum of the two player's values.

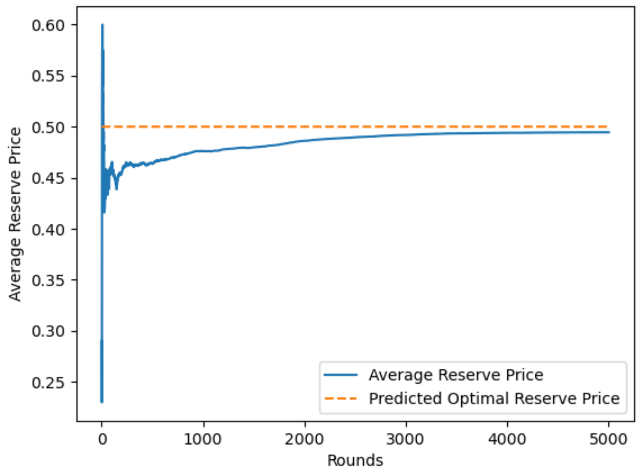
Preliminaries

In our experiment, we have m bidders and k items, where $m > k$, and generate the bidder values for each round using a given distribution (e.g. $F \sim U[0,1]$). We then create a data set to learn from, using the action-space consists of reserve prices R_i; R$\in [0,h]$, where h is the highest payoff. For each round, we calculate the revenue based on the bidder's values and the reserve price and enter it into the data set. We then use the exponential weights online learning algorithm on this data set to find the converged action (average reserve price) and the corresponding converged revenue (average payoff for each round). We can compare these values to the theoretical optimal reserve price given by $\phi'(0)$ and the theoretical optimal revenue given by $E[\phi]$, where $\phi(v) = v = \frac{1-F(v)}{f(v)}$. We also looked at what would happen to these converged values as we varied the number of bidders or items by using the same set up as before but running the experiment with different revenue or value generations and then plotting the values we get from our online learning algorithm.

In the second part of our experiment, the main focus was seeing if our online algorithm could find the optimal mechanism for various distributions for the two players. We used exponential weights for our online learning algorithm and used $\epsilon = 0.1$ which was found to be optimal by testing different values of ϵ. For our action space, we divided our possible reserve price from 0 to the maximum sum of the two players values divided into intervals of 0.01. We only added to our payoff history if the sum of the two players value was greater than the reserve price and if the player had a greater value than the reserve price, we would add 0 for that particular player. We found the theoretical expected revenue from the optimal mechanism by running a simulation 100,000 times using the theoretical reserve price and calculating the average revenue.

Results

For a truthful second price auction with price reserve (2 bidders, 1 item) with bidder values from $F \sim U[0, 1]$ we get the optimal reserve price and revenue from our learning algorithm.



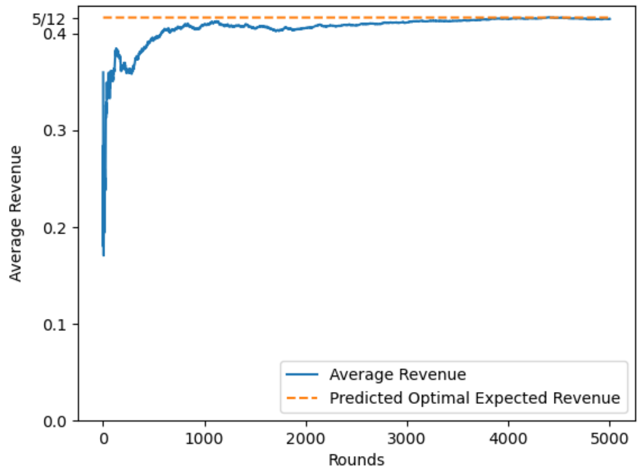
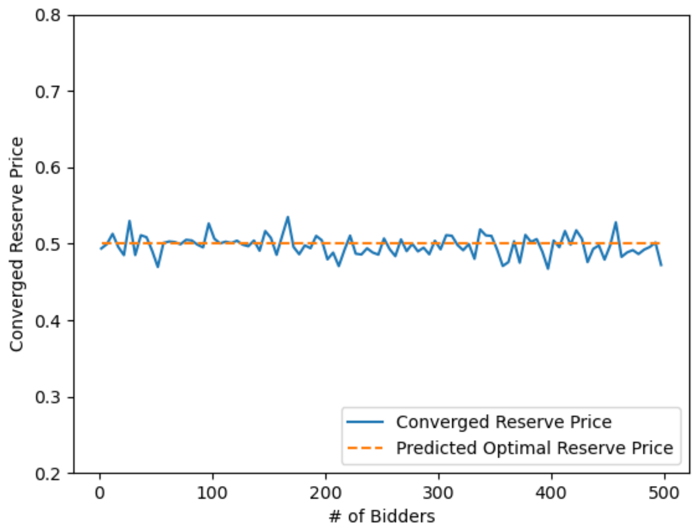


Figure 1: Average Optimal Reserve Price and Revenue Predicted by Learning Algorithm (Truthful Second Price Auction With Reserve)

As seen in Figure 1, the optimal reserve price converges to 1/2 and the optimal revenue converges to 5/12, which is what we expect when bidder's values are drawn from $F \sim U[0,1]$ since $\phi'(0) = 1/2$ and $E[\phi] = 5/12$ when $\phi(v) = v = \frac{1-F(v)}{f(v)}$. In each case, the value converged in about 2000-3000 rounds.

We now look at how the converged values change as we vary the number of bidders in the auction. We have bidders' values drawn from $U[0,1]$ and have $k = 1$ item each round.



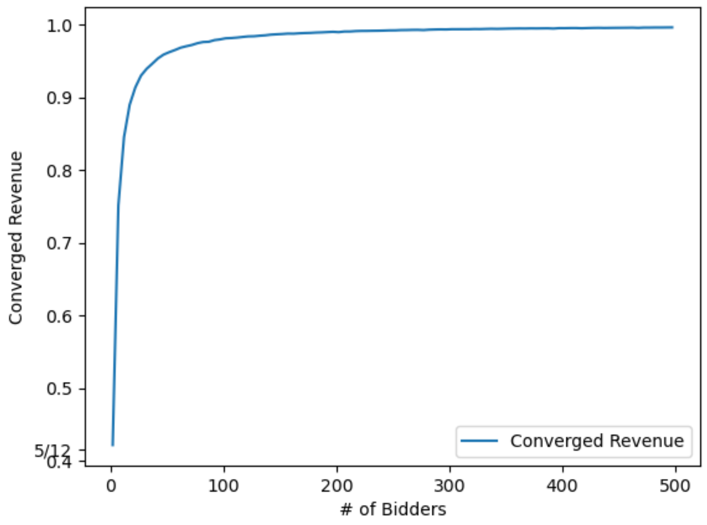
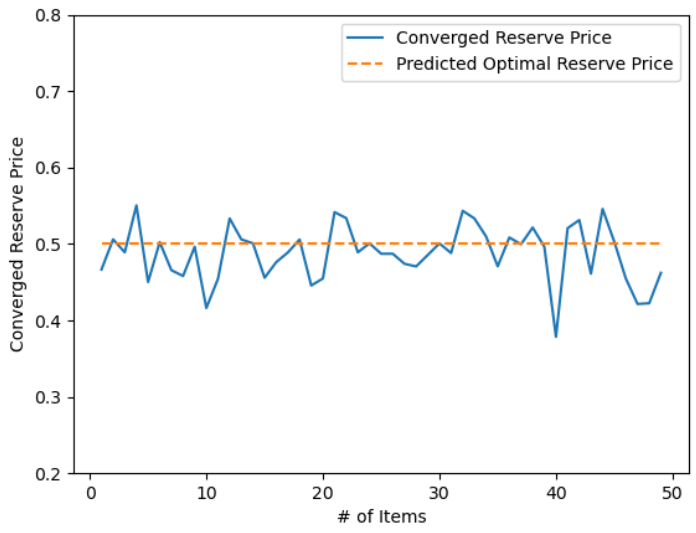


Figure 2: Converged Reserve Price and Revenue Predicted by Learning Algorithm With Varying Number of Bidders

We can see in Figure 2 that optimal reserve price appears to be constant as we vary the number of bidders. Looking at optimal revenue, as we increase the number of bidders, the revenue converges to 1.

We also wish to see how converged values change as we vary the number of items in the auction. We have bidders' values drawn from $U[0,1]$ and have $m = 100$ bidders each round.



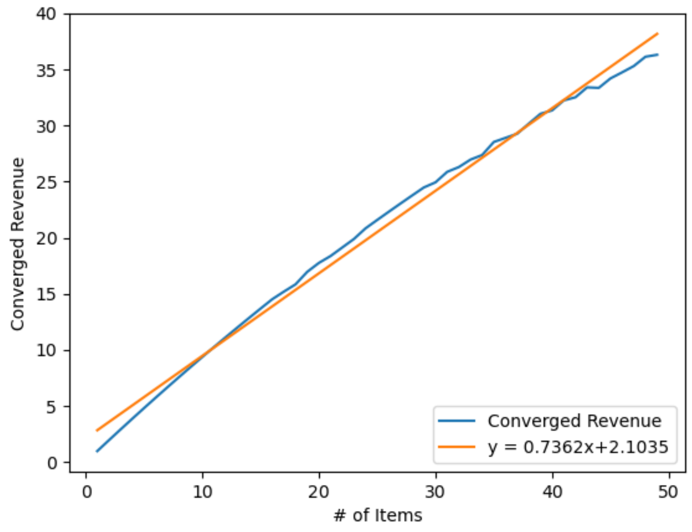
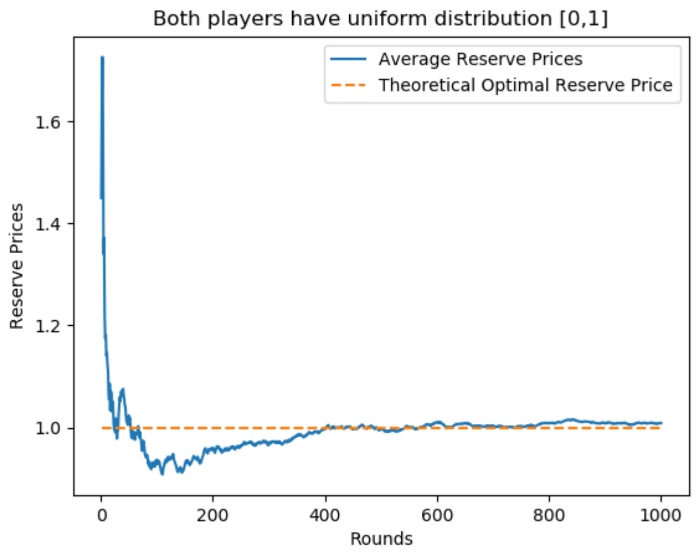


Figure 3: Converged Reserve Price and Revenue Predicted by Learning Algorithm With Varying Number of Items

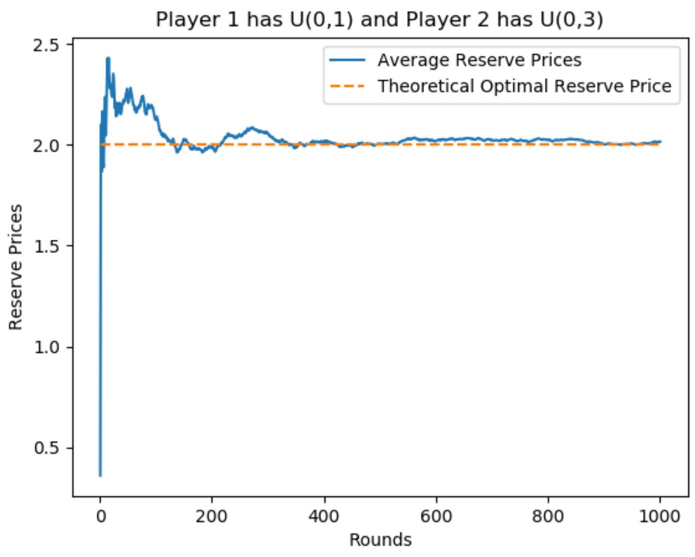
We can see in Figure 3 that optimal reserve price appears to be constant as we increase the number of items. Moreover, we see that optimal revenue appears to increase linearly as we increase the number of items in each round of the auction.

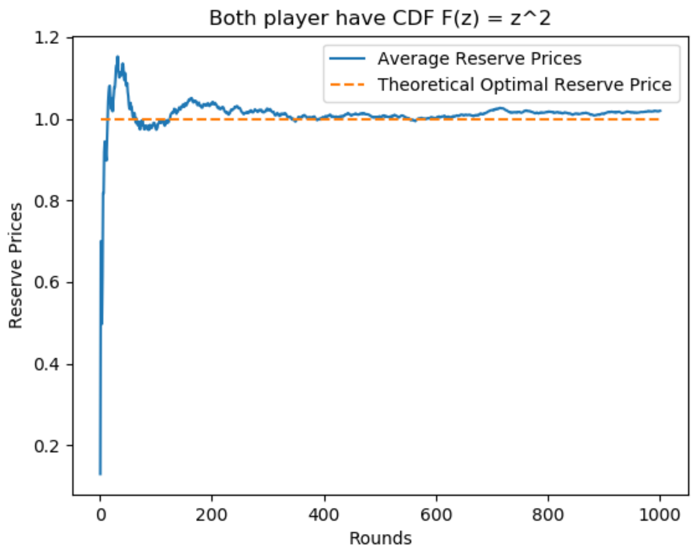
For the second part of our study, we first look at the results of when both players have a uniform distribution from 0 to 1. The graph of our online learning algorithm is show below.

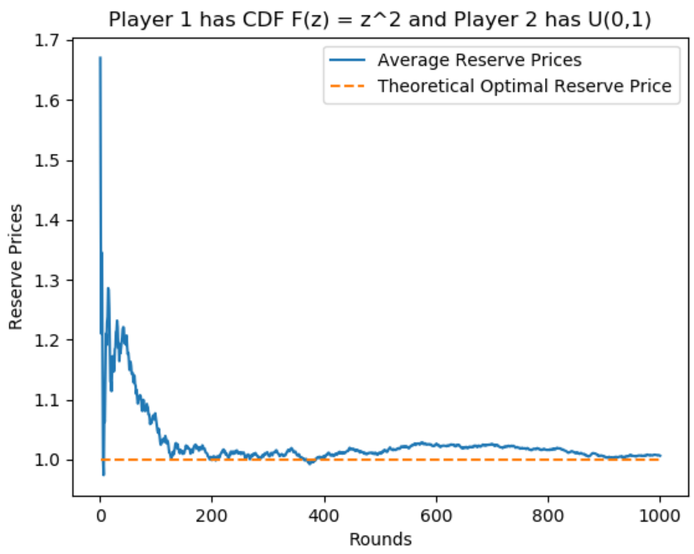


Project 4: Online Revenue Maximization

The theoretical optimal reserve price was calculated using the known distributions and using the player's virtual values. As we can see in the graph above, our online learning algorithm converges very quickly to the theoretical optimal reserve price. Using simulations, we found the theoretical expected revenue is around 0.33. The results of comparing different distributions are shown below.







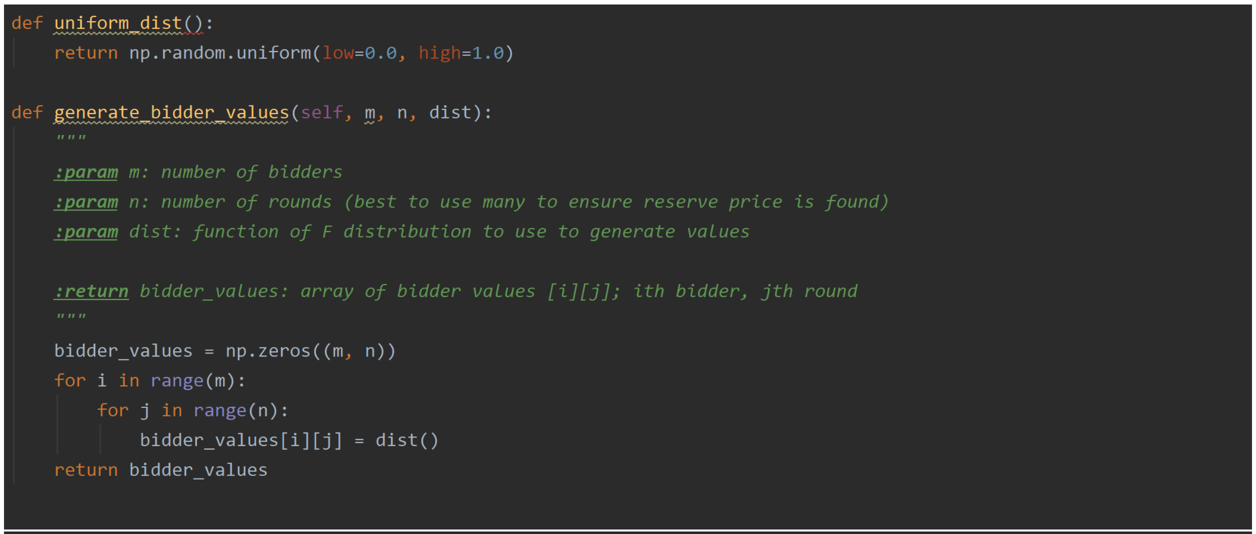
We can see that even when the distributions are both different and asymmetrical, our online learning algorithm performs very well.

Conclusions

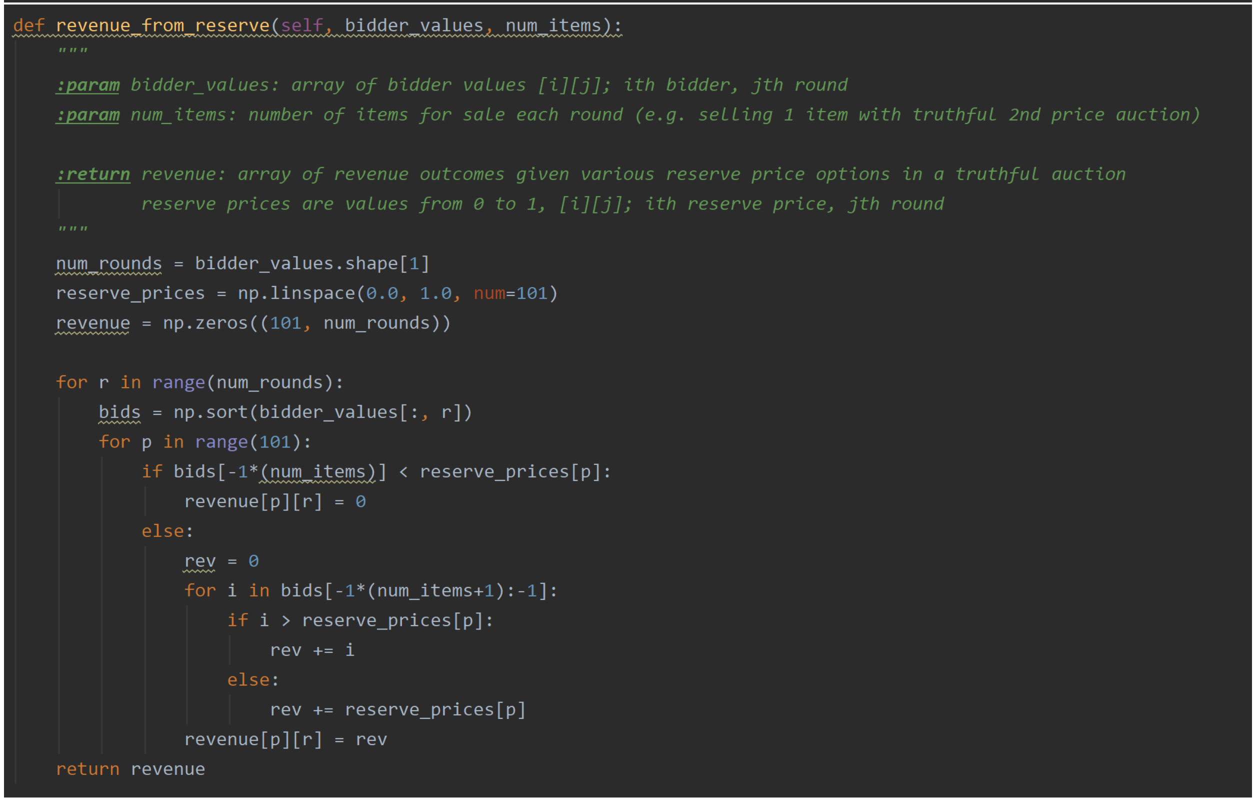
Using an exponential weights online learning algorithm we were able to learn the optimal reserve price and revenue for truthful auctions. We analyzed the truthful second price auction (2 bidders, 1 item) where bidders' values were generated from $F \sim U[0,1]$ and got the theoretical optimal values from our algorithm. We also looked at how these values would be affected as we vary the number of bidders or items in each round of the auction. As we increase the number of bidders, the reserve price remains constant, while the revenue increases, eventually converging to 1. This makes sense since having more bidders will not affect the reserve price, while increasing the chance of having high bids and thus increasing the potential revenue. As we increase the number of items, the reserve price also remains constant, while the revenue increase at a constant rate.

Even in different scenarios such as selling introductions, we found out that we were able to use exponential weights algorithm to quickly determine the optimal mechanism, even when the players had different distributions. However, in the real world, there's not only one employer and one employee. Something that could be explored further is whether our algorithm will still perform optimally when we add more players to each side.

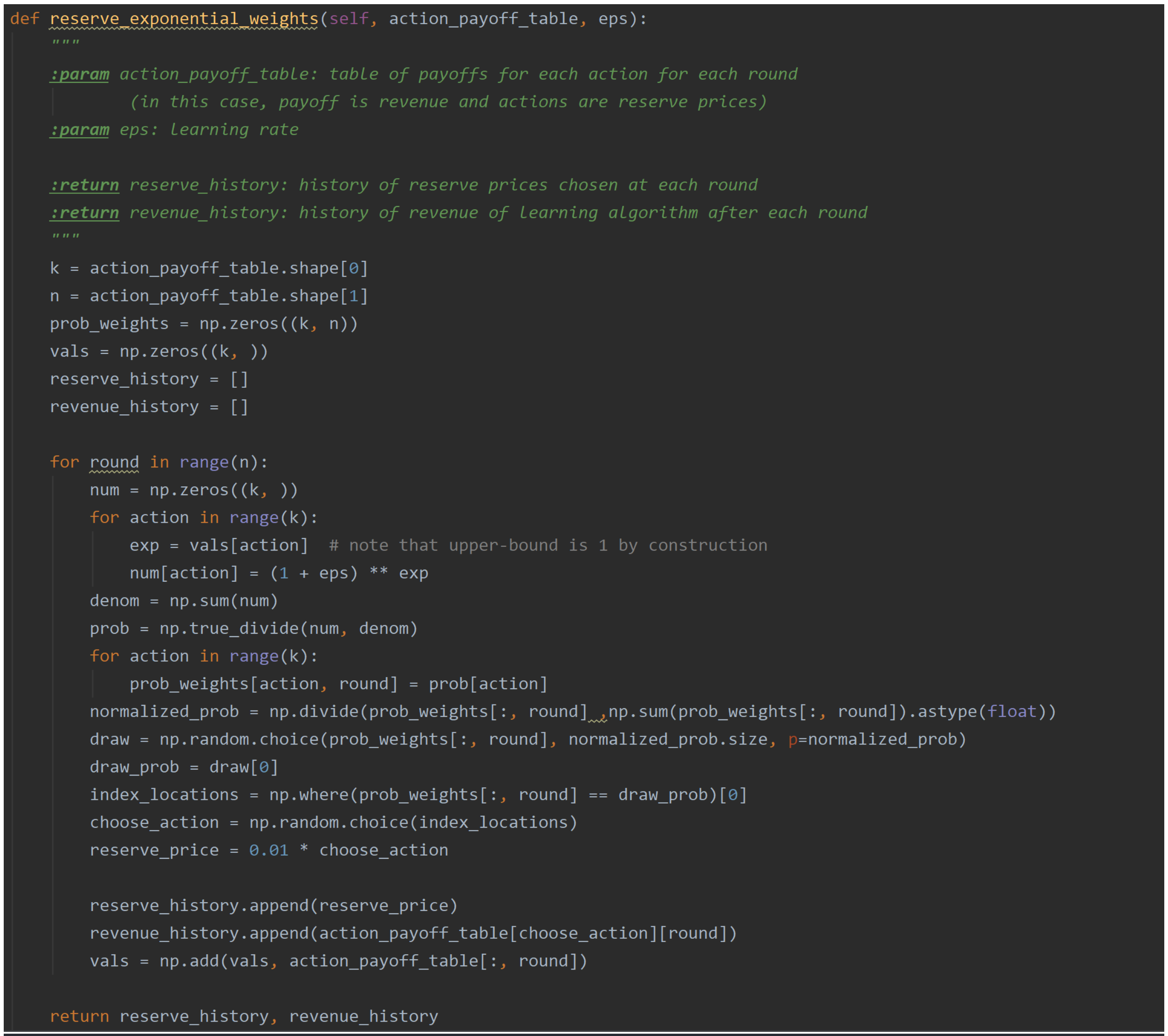
Appendix



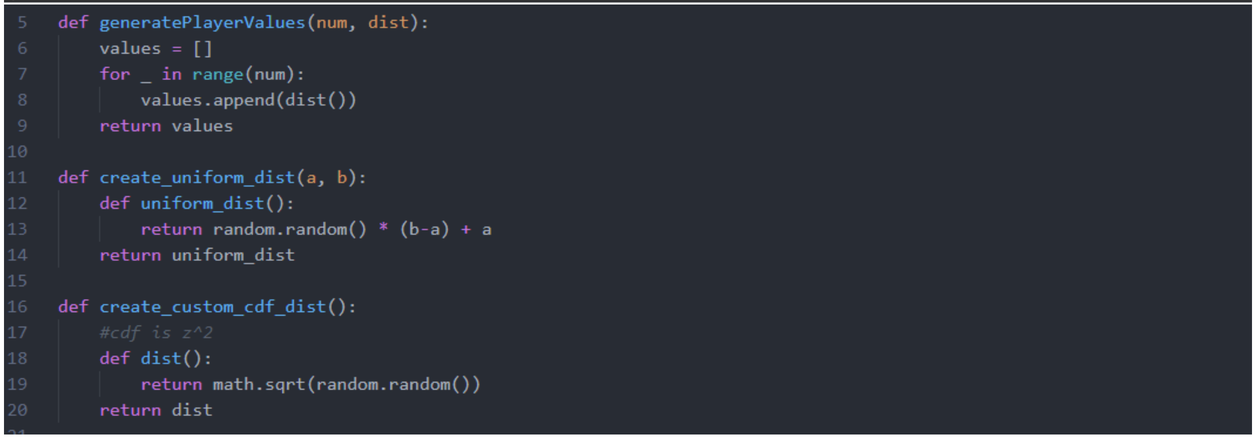
```python
def uniform_dist():
    return np.random.uniform(low=0.0, high=1.0)

def generate_bidder_values(self, m, n, dist):
    """
    :param m: number of bidders
    :param n: number of rounds (best to use many to ensure reserve price is found)
    :param dist: function of F distribution to use to generate values

    :return bidder_values: array of bidder values [i][j]; ith bidder, jth round
    """
    bidder_values = np.zeros((m, n))
    for i in range(m):
        for j in range(n):
            bidder_values[i][j] = dist()
    return bidder_values
```



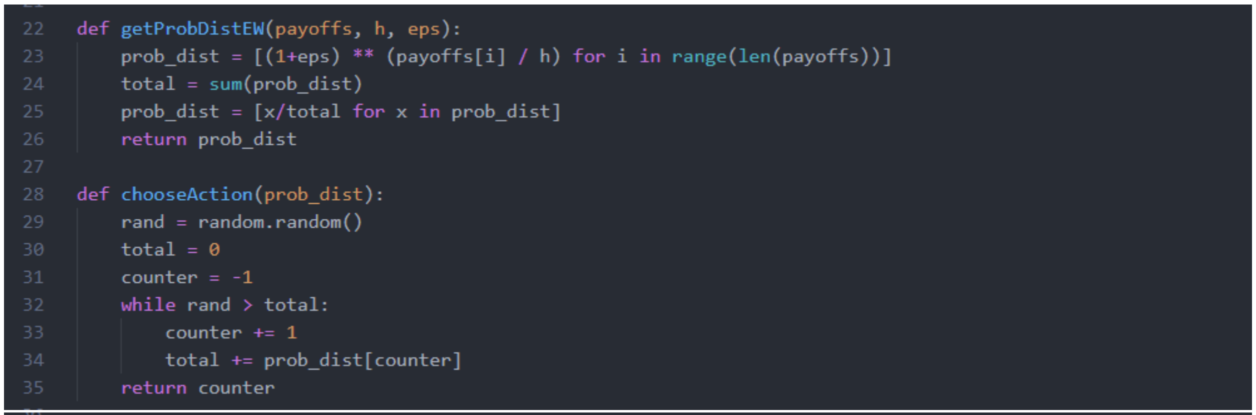
```python
def revenue_from_reserve(self, bidder_values, num_items):
    """
    :param bidder_values: array of bidder values [i][j]; ith bidder, jth round
    :param num_items: number of items for sale each round (e.g. selling 1 item with truthful 2nd price auction)

    :return revenue: array of revenue outcomes given various reserve price options in a truthful auction
            reserve prices are values from 0 to 1, [i][j]; ith reserve price, jth round
    """
    num_rounds = bidder_values.shape[1]
    reserve_prices = np.linspace(0.0, 1.0, num=101)
    revenue = np.zeros((101, num_rounds))

    for r in range(num_rounds):
        bids = np.sort(bidder_values[:, r])
        for p in range(101):
            if bids[-1*(num_items)] < reserve_prices[p]:
                revenue[p][r] = 0
            else:
                rev = 0
                for i in bids[-1*(num_items+1):-1]:
                    if i > reserve_prices[p]:
                        rev += i
                    else:
                        rev += reserve_prices[p]
                revenue[p][r] = rev
    return revenue
```



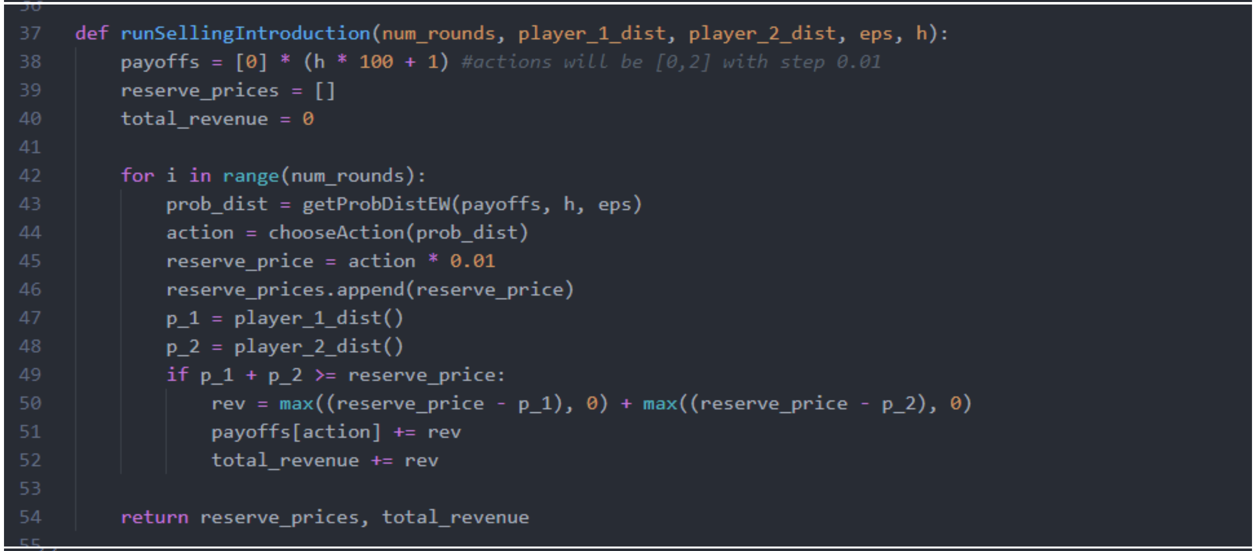
```python
def reserve_exponential_weights(self, action_payoff_table, eps):
    """
    :param action_payoff_table: table of payoffs for each action for each round
            (in this case, payoff is revenue and actions are reserve prices)
    :param eps: learning rate

    :return reserve_history: history of reserve prices chosen at each round
    :return revenue_history: history of revenue of learning algorithm after each round
    """
    k = action_payoff_table.shape[0]
    n = action_payoff_table.shape[1]
    prob_weights = np.zeros((k, n))
    vals = np.zeros((k, ))
    reserve_history = []
    revenue_history = []

    for round in range(n):
        num = np.zeros((k, ))
        for action in range(k):
            exp = vals[action]  # note that upper-bound is 1 by construction
            num[action] = (1 + eps) ** exp
        denom = np.sum(num)
        prob = np.true_divide(num, denom)
        for action in range(k):
            prob_weights[action, round] = prob[action]
        normalized_prob = np.divide(prob_weights[:, round],np.sum(prob_weights[:, round]).astype(float))
        draw = np.random.choice(prob_weights[:, round], normalized_prob.size, p=normalized_prob)
        draw_prob = draw[0]
        index_locations = np.where(prob_weights[:, round] == draw_prob)[0]
        choose_action = np.random.choice(index_locations)
        reserve_price = 0.01 * choose_action

        reserve_history.append(reserve_price)
        revenue_history.append(action_payoff_table[choose_action][round])
        vals = np.add(vals, action_payoff_table[:, round])

    return reserve_history, revenue_history
```



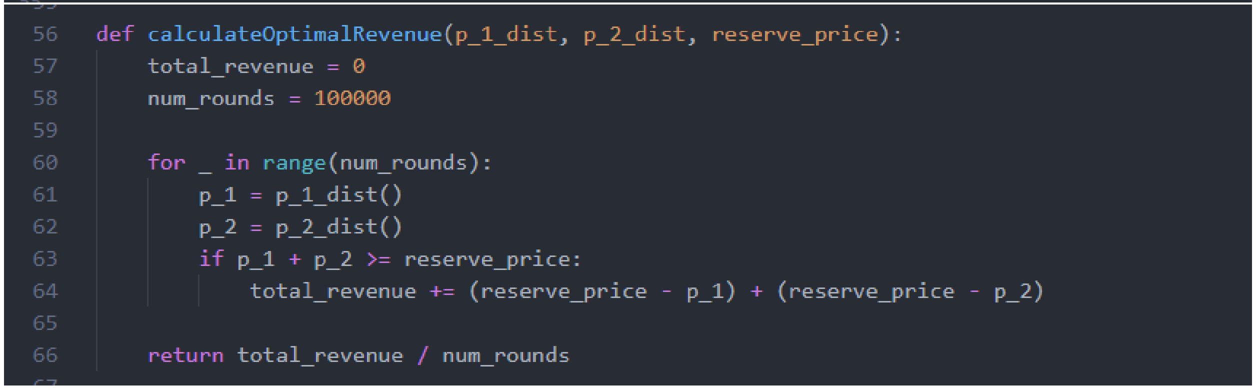
```python
def generatePlayerValues(num, dist):
    values = []
    for _ in range(num):
        values.append(dist())
    return values

def create_uniform_dist(a, b):
    def uniform_dist():
        return random.random() * (b-a) + a
    return uniform_dist

def create_custom_cdf_dist():
    #cdf is z^2
    def dist():
        return math.sqrt(random.random())
    return dist
```



```python
def getProbDistEW(payoffs, h, eps):
    prob_dist = [(1+eps) ** (payoffs[i] / h) for i in range(len(payoffs))]
    total = sum(prob_dist)
    prob_dist = [x/total for x in prob_dist]
    return prob_dist

def chooseAction(prob_dist):
    rand = random.random()
    total = 0
    counter = -1
    while rand > total:
        counter += 1
        total += prob_dist[counter]
    return counter
```



```python
def runSellingIntroduction(num_rounds, player_1_dist, player_2_dist, eps, h):
    payoffs = [0] * (h * 100 + 1) #actions will be [0,2] with step 0.01
    reserve_prices = []
    total_revenue = 0

    for i in range(num_rounds):
        prob_dist = getProbDistEW(payoffs, h, eps)
        action = chooseAction(prob_dist)
        reserve_price = action * 0.01
        reserve_prices.append(reserve_price)
        p_1 = player_1_dist()
        p_2 = player_2_dist()
        if p_1 + p_2 >= reserve_price:
            rev = max((reserve_price - p_1), 0) + max((reserve_price - p_2), 0)
            payoffs[action] += rev
            total_revenue += rev

    return reserve_prices, total_revenue
```



```python
def calculateOptimalRevenue(p_1_dist, p_2_dist, reserve_price):
    total_revenue = 0
    num_rounds = 100000

    for _ in range(num_rounds):
        p_1 = p_1_dist()
        p_2 = p_2_dist()
        if p_1 + p_2 >= reserve_price:
            total_revenue += (reserve_price - p_1) + (reserve_price - p_2)

    return total_revenue / num_rounds
```