In this homework, we will see a few of the numerous applications of the weighted majority algorithm. It is remarkable how powerful a simple algorithm, such as the multiplicative update rule, can be. The problem statements here may appear long, but that is mainly because this homework is combining some aspects of lecture notes and practice problems. It is suggested that you attempt these problems in preparation for the final exam. The solutions to these problems will be posted in the middle of next week.

**Problem 1.** (von Neumann’s Minmax Theorem). The first problem is one in game theory. We consider a game between two players, where player 1 can choose from among $n$ options and player 2 can choose from among $m$ options. The payoff to player 1 is given by a payoff matrix, $A_{n \times m}$, where $A_{ij}$ is the payoff received by player 1 if she plays $i$ and player 2 plays $j$. We will consider zero-sum games, thus, the payoff matrix from the second player will simply be $-A$. We will assume that each entry of the matrix $A$ is bounded in the range $[-M, M]$.

The players may play according to mixed strategies, i.e. instead of choosing a single option, they choose a distribution over the available options. Suppose, player 1 chooses a distribution, $x \in \Delta_n$, and player 2 chooses a distribution, $y \in \Delta_m$, then the expected payoff received by player 1 is $x^T A y$.

In order to consider the value of the game, it is useful to think of one player going first and announcing her strategy, and the second player using best response. Suppose player 1 plays first, then she guarantee that her expected payoff is $v_1 = \max_{x \in \Delta_n} \min_{y \in \Delta_m} x^T A y$. This is the max-min value, denoted by $v_1$. Here, the second player tries to minimize the expected payoff of player 1 (since it is a zero-sum game). One way to think of this as follows, suppose player 1 is going to use $x$ as her strategy and she announces it. In that case, player 2 will choose $y$ that minimizes $(x^T A y)$. Player 1 should choose the vector $x$ that maximizes this minimum value.

Similarly, if player 2 goes first, and announces $y$. Then the maximum expected value that player 1 can obtain is $\min_{y \in \Delta_m} \max_{x \in \Delta_n} x^T A y$. This is the min-max value, denoted by $\bar{v}_1$. The min-max theorem, states that $v_1 = \bar{v}_1$, or that the max-min and min-max values are the same. Thus, who plays first is irrelevant.

We will prove the min-max theorem using the repeated $n$-decision problem. Suppose, the two players play the same game for time steps $t = 1, \ldots, T$. Player 1 goes first and plays using the weighted majority algorithm. Player 2 plays best response, i.e. if player 1 plays $x^t \in \Delta_n$ at time step $t$, player 2 chooses $y^t$, that minimizes $(x^t)^T A y^t$.

1. First show that the max-min value is smaller than the min-max value, i.e. $v_1 \leq \bar{v}_1$. (This should obviously be the case, since going second is an advantage.)

2. Show that the average payoff of player 1, $\frac{1}{T} \sum_{t=1}^{T} (x^t)^T A y^t$ is at most $v_1$.

3. Show that the average payoff that player 1 could have obtained in hindsight is at least $\bar{v}_1$.

4. Use the weighted majority theorem in the limit as $T \to \infty$ to conclude the fact that $v_1 = \bar{v}_1$. 

1
**Problem 2.** *(Linear Programming)* A typical linear program has the following form:

\[
\min \ c^T \cdot x \\
\text{subject to:} \\
Ax \leq b \\
x \geq 0
\]

where \( x \in \mathbb{R}^m, x_i \geq 0 \) for \( i = 1, \ldots, n \). \( A \) is an \( n \times m \) matrix, \( b \in \mathbb{R}^n \). Thus, there are \( n \) constraints; and \( c \in \mathbb{R}^m \). Let \( a_i \) denote the \( i^{th} \) row of \( A \) and \( b_i \) denote the \( i^{th} \) entry of the vector \( b \). Thus, \( a_i^T x \leq b_i \) is simply the \( i^{th} \) constraint of a linear program. One way to solve a linear program is to guess the value of the objective function, say \( z^* \) (by binary search) and try to find a vector in the set \( \mathcal{P} = \{ x \mid x \geq 0, c^T x = z^* \} \). The superscript, \( T \), here means the transpose.

1. Let \( a \in \mathbb{R}^m \) and \( b \in \mathbb{R} \). Show that if there is only one constraint, \( a^T x \leq b \), then it is easy to determine if there exists \( x \in \mathcal{P} \) for which the constraint is satisfied.

2. Now, let \( \rho = \max \{ \max_{i, x \in \mathcal{P}} |a_i^T x - b_i|, 1 \} \). We will set up a repeated \( n \)-decisions problem. Each constraint is thought of as one of \( n \) choices. Suppose \( \mathbf{w}^t \) is the distribution over the \( n \) constraints obtained by weighted majority at time step \( t \). Let \( a^t = (\mathbf{w}^t)^T A \) and let \( b^t = \mathbf{w}^t b \). We consider the constraint \( (a^i)^T x \leq b^t \). Show that if there is no \( x \in \mathcal{P} \) that satisfies \((a^i)^T x \leq b_i \), then the original program is infeasible, i.e. there is no \( x \in \mathcal{P} \), such that \( Ax \leq b \).

3. Otherwise, let \( x^t \) be any vector in the set \( \mathcal{P} \) that satisfies the constraint, \((a^i)^T x^t \leq b_i \). The payoff for action (constraint) \( i \) is \((a^i)^T x^t - b_i \). Show, that the payoff of the algorithm is positive at each round, unless the program was declared infeasible.

4. Let \( x^* = \frac{1}{T} \sum_{t=1}^{T} x^t \) and suppose \( T = 16\rho^2 \ln(n)/\epsilon^2 \). Then show that, \( a_i^T x^* \leq b_i + \epsilon \) for each \( i \).

**Remark:** What we have shown is that we have almost solved the linear program. Note that each constraint may be violated slightly (by \( \epsilon \)), and a tighter guarantee may be obtained using a larger \( T \). This is not the most optimal method to solve linear programs. More involved algorithms, such as the ellipsoid algorithm, do give truly polynomial time algorithms for linear programming. None the less, the above approach can be used to obtain interesting polynomial time approximation algorithms, when it is sufficient to find an approximately feasible solution to a linear program.

**Problem 3.** *(Sleeping Experts)* A variation of the standard \( n \)-decision problem, is the sleeping experts problem. Here, each of the \( n \) decisions is just some expert advice. However, at any given time step \( t \), the decision-maker (your algorithm) may only have access to a subset, \( S^t \subseteq [n] \) of experts. The remaining experts may be sleeping.

The notion of regret in hindsight is more involved in this case. Note that the payoff of the best expert makes little sense, because some experts may not be available (awake) on every round. We consider the following to be the best strategy in hind-sight: we consider a ranking over the experts. A ranking, \( \sigma \) is simply a permutation of \( n \) elements. At time-step \( t \), the ranking strategy according to \( \sigma \), is implemented as follows: let \( \sigma(S^t) \) denote the highest-ranked expert that is in \( S^t \) according to ranking \( \sigma \). For example, if \( n = 5 \), \( \sigma = (3, 2, 4, 1, 5) \) and \( S^t = \{1, 4\} \), then the strategy would be to follow the advice of expert 4, since the 4th expert is awake and ranked higher than other awake experts (in this case just expert 1). Thus the regret is measured with respect to the best ranking in
hindsight (when the payoffs for all experts are known). We assume that the payoffs for each expert lie in the range \([-M, M]\). Then,

\[
\text{regret} = \max_{\sigma \in \text{perm}(n)} \frac{1}{T} \sum_{t=1}^{T} p_{\sigma(S^t)}^t - \frac{1}{T} \sum_{t=1}^{T} p_{d^t}^t,
\]

where \(d^t\) is the expert (from the set \(S^t\)), chosen by the decision-maker, and \(\sigma(S^t)\) is the expert that would have been chosen by the ranking \(\sigma\).

Show that it is possible to obtain an algorithm that after \(T\) time-steps has regret \(O(M \sqrt{\frac{n \log(n)}{T}})\). Show that in fact you can assume that each of the \(n!\) possible rankings is a new meta-expert. Now implement weighted-majority algorithm as if you were playing a repeated \(n!\)-decision game with \(n!\) experts. You may assume that at the end of the round, the entire payoff vector is revealed, including the payoffs that experts who were sleeping (and were not available to the decision-maker) would have received. What can you say about the running time of your algorithm?