Problem 1: Linear Programming (15 points, no collaboration)

Alice is trying to get enough oranges and bananas to host a party. To successfully host a party she needs at least 6 oranges and at least 12 bananas. Unfortunately, her local grocery store only sells fruit in bundles. Bundle A costs 2 dollars and contains one orange and one banana. Bundle B costs 4 dollars and contains one orange and three bananas. Fortunately, the grocery store will allow Alice to buy fractions of bundles (i.e. she can buy 2.5 bundle As to get 2.5 oranges and 2.5 bananas). They will not allow Alice to buy negative bundles (i.e. she cannot buy -1 bundle As and 3 bundle Bs to get 2 oranges and 8 bananas).

Alice would like to buy $x_A$ bundle As and $x_B$ bundle Bs to guarantee she has at least 6 oranges and at least 12 bananas. Moreover, she would like to find the solution that minimizes her dollars spent.

Part a (5 points)

Write a linear program whose solution is the optimal choice of $x_A, x_B$ for Alice’s problem.

Part b (5 points)

Take the dual of the linear program from part a.

Part c (5 points)

State the optimal choice of $x_A, x_B$, and prove that it’s optimal.
Problem 2: An Algorithm for Nash (15 points)

A feasibility LP is an LP but without the objective function. That is, a feasibility LP is just a list of inequalities. A feasibility LP either outputs some \( \vec{x} \) that satisfies all the inequalities (if a solution exists), or outputs “no” if no such solution exists.\(^1\)

Part a (10 points)

Consider a two-player (non-zero-sum) game with payoff functions \( p_1(\cdot, \cdot) \) for player 1, and \( p_2(\cdot, \cdot) \) for player 2. Player 1 has strategies \( A_1 = \{s_1^1, ..., s_n^1\} \) available, and player 2 has strategies \( A_2 = \{s_1^2, ..., s_n^2\} \).

Suppose you are given specific subsets of strategies \( S_1 \subseteq A_1, S_2 \subseteq A_2 \). Write a feasibility LP to determine whether there exists a Nash for this game where every strategy in \( S_1 \) is a best response for player 1, player 1 only uses strategies in \( S_1 \) with non-zero probability, every strategy in \( S_2 \) is a best response for player 2, and player 2 only uses strategies in \( S_2 \) with non-zero probability.

Part b (5 points)

Recall \( n \) is the number of actions each (of two) player has. Design an exponential time algorithm (exponential in \( n \)) to find a Nash equilibrium. You may use without proof the fact that linear programs can be solved in polynomial time. You may also use without proof the fact that Nash equilibria always exist.

Problem 3: Biased Information Cascades (20 points)

Say I have an urn with one red ball, one blue ball, and one ball that is red with probability \( p \) and blue with probability \( 1 - p \) (but it stays the same color - just like in class). One at a time, participants draw a ball from the urn and guess whether the urn has more red balls or blue balls, and then return the ball to the urn. Participants are fully rational, trying to guess correctly, and know that every other participant is fully rational.

Recall that we say an information cascade occurs at step \( t \) if the \( t^{th} \) player to draw a ball ignores their draw. We say a cascade is red if in all future steps, every player guesses red (and blue otherwise).

Part a (5 points)

Let \( p > 2/3 \). What is the probability that an information cascade occurs by time \( t \) (for all \( t \))? Given that a cascade occurs, what is the probability that the cascade is red?

Part b (15 points)

Let \( p \in (1/2, 2/3) \). What is the probability that an information cascade occurs by time \( t \) (for all \( t \))? Given that a cascade occurs, what is the probability that the cascade is red?

---

\(^1\)For example, a feasibility LP might have the constraints \( x + y \leq 3 \) and \( 2x - y \leq 4 \). One solution to this feasibility LP would be \( x = y = 0 \) (but there are many solutions). Another example might have constraints \( x + y \leq 3 \) and \( -x - y \leq -4 \). There are no solutions to this LP (because if we add the two constraints we get a contradiction \( 0 \leq -1 \). Another way to see this is that we have the constraints \( x + y \leq 3 \), and \( x + y \geq 4 \).
Problem 4: Ultimatums with Reputations (15 points)

In this challenge, you’ll play the following game:

Setup:

- You will play a sequence of rounds. Every round consists of a simple two-player game (defined below).
- You will be matched with a random partner in each round who you did not play before.
- You will see some information about your partner’s play in previous rounds before deciding what to do.

Round:

- Each round you will play the Ultimatum game with your partner.
- One of you will be labeled as Alice, the other will be labeled Bob.
- First, Bob announces any number $B \in \{0, \ldots, 99\}$. This has no impact on the game. But Bob may use it to communicate, if he wishes (this is sometimes referred to as “cheap talk”).
- Second, Alice proposes to Bob any number in $A \in \{0, \ldots, 99\}$.
- Bob may then accept or reject. If Bob accepts, Alice gets payoff $A$ and Bob gets payoff $100 - A$. If Bob rejects, both Alice and Bob get payoff 0.

History:

- You will learn some information about your partner’s past play before deciding what to offer (and whether to accept/reject).
- You will see, for their most recent round as Alice, what Bob said, what Alice proposed and whether it was accepted or not.
- You will also see, for their most recent round as Bob, what Bob said, what Alice proposed and whether it was accepted or not.
- If any such previous rounds don’t exist, the default will be $(-1, -1, \text{accept})$.
- You will also see the total number of rounds as Bob where they accpeted, and the total number of rounds as Bob where they rejected.

Total Payoffs:

- Your total payoff will sum your payoff in every round. Remember that your goal is to maximize your payoff so that you perform well in comparison to the course staff, not to perform better than your classmates.

Your strategy should have four functions. The first function should take as input a history in the format described above. The second function, as Alice, should take as input a number in $B \in \{0, \ldots, 99\}$ corresponding to Bob’s small talk, and output an integer $A \in \{0, \ldots, 99\}$ corresponding to your proposal to (try to) keep for yourself. As Bob, you first should define a function to announce a number $B \in \{0, \ldots, 99\}$ after having received the history in the first function. You should also define a function to output accept or reject given an offer $A$. Note that $A$ will always refer to the payoff that Alice keeps, and $B$ is cheap talk.

Code it up according to the specifications below, and write a brief justification. Please visit see GradesForProgramming.pdf for brief details on grading policies with respect to programming challenges. Recall that all of your main justification should fit in one page. If you wish to include calculations or simulation results, you may do so in-line, but your writeup should contain at most one page of English justification. This will not be strictly enforced, but graders may not read justifications that go significantly beyond the one-page guideline in full.
Part a (5 points)
What is a subgame-perfect Nash for the Ultimatum game? Describe another Nash equilibrium that is not subgame-perfect.

Part b (5 points)
If the game were modified so that none of your behavior as Bob was ever passed on. What would you do as Bob? Knowing this, what would you do as Alice?

Part c (5 points) Programming Component Specifications and Writeup
You will implement the Ultimatum interface provided in Ultimatum.java, which requires the following methods, as documented in Ultimatum.java.

```java
// Feel free to rename these variables in your implementation!
public void setup(int opponentAsBobSaid,
                   int opponentSawAliceSaid,
                   boolean opponentAsBobAccepted,
                   int opponentSawBobSaid,
                   int opponentAsAliceSaid,
                   boolean opponentAsAliceWasAccepted,
                   int opponentAsBobAcceptedCumulative,
                   int opponentAsBobRejectedCumulative);

public int cheapTalk();

public int propose(int bobCheapTalk);

public boolean accept(int aliceProposal);
```

We’ve also provided a sample strategy, Ultimatum_smattw.java, and the usual construction of testing code: SimulatorBase.java and a makefile to run it against varied strategies.

Your file must follow the naming convention Ultimatum_netID1_netID2.java, where netID1 and netID2 are the Net IDs of the submitters. If you work alone, use only one name; if we approved you working in a triple, use all three. All advice from previous homeworks applies again to this homework, especially that you should consider the performance of your strategy against the strategies and distribution thereof which you expect your peers to play.

Your writeup should provide an overview of the main ideas in your code (remember that we also have your code — so you don’t need to provide pseudocode or a step-by-step description of your algorithm), and justify why you think it will perform well, in addition to concrete answers to parts a and b.

Penalties may be given for code which does not compile, throws exceptions, or violates assertions. Remember to test your code with settings besides the default settings of our testing environment. Extra credit may be awarded for reporting substantive bugs in our testing code.
Extra Credit: Another Algorithm for Nash

Recall that extra credit is not directly added to your PSet scores, but will contribute to your participation grade. Some extra credits are quite challenging and will contribute significantly.

For this problem, you may collaborate with any students. You may not consult course resources or external resources. In this problem we will guide you through the proof of a well-known result, so you should not copy the proof from one of the course texts (nor should you try to find a proof from external sources). You must follow the guide below (and not provide an alternative proof).

Consider any symmetric two-player game. That is, \( p_1(x, y) = p_2(y, x) \) for all \( x, y \). Consider also the following system of inequalities. We’ll refer to this as the Lemke-Howson Polytope.

- Variables \( x_1, \ldots, x_n \).
- (Non-negativity) \( x_i \geq 0 \) for all \( i \).
- (Responsiveness) \( \sum_j x_j p_1(i, j) \leq 1 \).

**Part a**

Prove that any point \( \vec{x} \neq \vec{0} \) inside the Lemke-Howson polytope such that for all \( i \) either the non-negativity constraint is tight (i.e. \( x_i = 0 \)), or the Responsiveness constraint is tight satisfies \( \vec{x}/|\vec{x}|_1 \) is a symmetric Nash equilibrium (\( |\vec{x}|_1 = \sum_j x_j \)).

**Part b**

The Lemke-Howson algorithm starts from the point \( \vec{0} \) and repeatedly pivots. That is, the current point will always have exactly \( n \) tight constraints. The pivot will pick one of these constraints and “relax” it (keeping the other \( n - 1 \) tight). A new constraint will become tight, and this will be the new point (you do not need to prove that this procedure is well-defined).

From \( \vec{0} \), the pivot rule simply picks an arbitrary tight constraint to relax (let’s say \( x_1 = 0 \)). This causes a new constraint to become tight. If it’s the 1st Responsiveness constraint, then by Part a we’ve found a Nash and are done! If not, then we have exactly one double-covered action. That is, there is some action \( i \) such that the non-negativity and Responsiveness constraints are both tight. We pick the non-negativity constraint for \( i \) to relax next.

In general, for our current point \( \vec{x} \neq \vec{0} \), if there is no double-covered action we terminate (and hope that it’s a Nash and not back at \( \vec{0} \)). If there’s a double-covered action, it’s because we just made one of the constraints for \( i \) tight. So relax the other one and continue.

Prove that the Lemke-Howson algorithm will never visit a vertex \( \vec{y} \) without first revisiting the origin. You may use without proof the fact that if \( \vec{z} \) pivots to \( \vec{w} \) when constraint \( C \) is relaxed, causing constraint \( D \) to become tight, then \( \vec{w} \) pivots to \( \vec{z} \) when constraint \( D \) is relaxed, causing constraint \( C \) to become tight.

**Part c**

Prove that the Lemke-Howson algorithm cannot ever return to \( \vec{0} \). Conclude that the Lemke-Howson algorithm finds a Nash after at most \( (2^n) \) pivots.

Hint: You may want to prove that the algorithm terminates as soon as 1 becomes covered.