Mathematical Game theory
Assignment 7, due 2019-11-01

**Problem 1.** In a gameshow, Ruth has $200 and Chris has $300. Each must decide to pass or gamble, not knowing the choice of the other. A player who passes keeps the money they started with. If Ruth gambles, she wins $200 with probability 1/2 and loses $100 with probability 1/2. If Chris gambles, he wins or loses $300 with probability 1/2 each. These outcomes are independent. Finally, the contestant with the higher amount of money at the end wins a bonus of $400.

(a) Describe this in matrix form.
(b) Draw the Kuhn (game) tree, indicating information sets.
(c) Find the safety levels.
(d) Find a Nash equilibrium.

**Problem 2.** Find a Nash equilibrium in the game

\[
\begin{array}{ccc}
(1, 2) & (0, 1) & (0, 1) \\
(3, 0) & (1, 1) & (0, 3) \\
(2, 0) & (3, 0) & (5, 1)
\end{array}
\]

**Problem 3.** Find all Nash equilibria in the game

\[
\begin{array}{ccc}
(2,3) & (0,1) & (2,0) \\
(0,3) & (3,2) & (0,0) \\
(2,2) & (3,0) & (2,3)
\end{array}
\]

**Problem 4.** We consider a model for a duopoly in a new product. If the total amount produced is \( Q \), then the price of each unit is \( A - Q \), for some fixed and known \( A \).

(a) Company I pays \( C_1 \) to make each unit. How much should they produce (to maximize profit)?
(b) Company II enters the market, and can produce each unit for only \( C_2 < C_1 \). Suppose company I decides how much to produce and declares the decision. Then company II decides how much to produce. Find all Nash equilibria for this model, and compare the profits and price.

**Problem 5.** Consider a general-sum game with \( m \times n \) payoff matrices \( A, B \).

(a) For a fixed strategy \( x \in \Delta^m \), let \( S \subset \Delta^n \) be the set of all strategies \( y \) such that \((x, y)\) is a Nash equilibrium. Prove that \( S \) is convex.
(b) Give an example showing that the set of all \((x, y) \in \Delta^m \times \Delta^n \) that are Nash equilibria might not be convex. [hint: look at examples from class]

**Problem 6.** [bonus] Consider the following game. Each player can donate $1, in which case the other receives $2, or can keep their money (as in class).

\[
\begin{array}{cc}
(0,0) & (2,1) \\
(-1,2) & (1,1)
\end{array}
\]

Write a python program to play this repeatedly against other similar programs. The decision at each step can depend on what you did in the past, and the other program did in the previous rounds.
The program should be a function `play(us, them)`, where \( X, Y \) are lists of 0's and 1's of our and their previous actions. (In the first round, these are empty lists.) The function should return a number in \([0, 1]\), which is the probability that you decide to donate.

Each pair of programs will play each other 1000 times, accumulating their scores. Valid submissions get points. Winner gets a bonus.

Samples:

# always donate
def play(us, them):
    return 1

# donate with probability 1/3:
def play(us, them):
    return 1/3

# do what the opponent did last time. Donate in round 1.
def play(us, them):
    if len(them) == 0:
        return 1
    return them[-1]

# if the opponent donated k times out of m, donate with probability k/m.
# Coin toss first time
def play(us, them):
    if len(them) == 0:
        return 1/2
    return sum(them)/len(them)