Problem 1. A constant-sum game is a game where there is some \( L \) so that \( a_{ij} + b_{ij} = L \) for every \( i, j \). Prove that in a constant-sum game every Nash equilibrium gives Player 1 the same payoff (and similarly for Player 2).

Problem 2. Find a Nash equilibrium in the game

\[
\begin{array}{ccc}
(2, 2) & (1, 1) & (1, -1) \\
(4, 0) & (0, 1) & (1, 3) \\
(3, 0) & (-2, 0) & (6, -1)
\end{array}
\]

Problem 3. Find all Nash equilibria in the game

\[
\begin{array}{ccc}
(2,4) & (0,2) & (2,1) \\
(0,4) & (3,3) & (0,1) \\
(2,3) & (3,1) & (2,4)
\end{array}
\]

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 \$\{0,1,2\}, in which case the other receives double the donation.

\[
\begin{array}{ccc}
(0,0) & (2,-1) \\
(4,-2) & (1,1) \\
(-1,2) & (0,3) \\
(3,0) & \end{array}
\]

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 \texttt{play(us,them)}, where \texttt{X}, \texttt{Y} are lists of 0’s, 1’s and 2’s of our and their previous actions. (In the first round, these are empty lists.) The function should return a vector with the probabilities of donating \(0,1,2\).

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

Samples:

```python
#always donate 2
def play(us,them):
    return (0,0,1)
```
#donate with equal probabilities 1/3:
def play(us,them):
    return (1/3,1/3,1/3)

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

# do what the opponent did on a random previous turn. 
# donate in round 1
def play(us,them):
    if len(them) == 0: return (0,0,1)
    L = len(them)
    return [them.count(i)/L for i in (0,1,2)]