Zero-Sum Games and Linear Programming

Derryl Sayo

```
import numpy as np
import cvxpy as cp
```

Game Theory

In general, game theory is the study of mathematical models representing the interactions between agents (whether that be a person, animal, computer program, government, etc.) who aims to perform the most optimal action given the circumstances (see Wikipedia contributors 2024).

The study of game theory can be applied to analyzing simple games such as Rock-Paper-Scissors to extending into the study of cooperation among living organisms (as seen in (Axelrod and Hamilton 1981)).

Zero-Sum Matrix Games

One specific form of game that is analyzed is the "zero-sum" game.

Consider the situation in which player 1 (P1) and player 2 (P2) each choose from a finite set of stretegies and play a game such that P1 wins. This results in in P1 gaining some amount c and P2 losing that same amount $-c$, whether that be money, points, or some other notion of "utility". Then, for each intersection of each of the choices, the net gain/loss is equal to zero such that the entire game's sum of gains/losses is zero as well (hence "zero-sum").

In other words, zero-sum games are scenarios in which one player gains some amount and the other loses that same amount.

Zero-sum games are played in the following manner:

- P1 will choice some action available to them secretly and P2 will choice some action available to them independently secretly. Let's say that these actions are chosen at the exact same time and neither player can react to the other's choice until the game is played and resolved.
- Then, each choice is revealed and the points are awarded based on the pre-defined outcome of each player choosing their strategy.

We say that P1 will choose some row $i \in 1, \dots, m$ and P2 will choose some column $j \in 1, \dots, n$ and the payoff to P1 is the intersection a_{ij} in an $m \times n$ payoff matrix A.

So, a zero-sum game can be summarized as a single matrix:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}$$

(Note that there are representations in which the payouts for both players are written as (x, y) where x is the payoff to P1 and y is the payout to P2, but in zero-sum games it is implied that $y = -x$ so we could omit the P2 payoff as seen above.)

Another important aspect of zero-sum games is that the choice of strategy does not have to be picking a single row/col deterministically (ie. play with a "pure" strategy) and there exists the notion of playing with a "mixed" strategy where the choice is based on a probability distribution among the possibilities for each player.

Now, the "optimal" strategy for each player is to maximize their points but for every singular choice there may be a choice that the other player can make that could reduce your payout. So how should we decide which strategy, pure or mixed, to play to maximize the point gains for both players?

Nash Equilibrium, Minimax, and Solving Zero-Sum Games

If both players play optimally, intuitively there should be some point in which both players are satisfied with the gain/loss outcome.

Consider the game represented as a matrix from P1's perspective. Consider the scenario where P1 picks some strategy and then P2 counters. For any strategy, P1 wants to maximize their payoff but P2 wants to respond by minmizing their own loss. To mitigate the worst case of gaining the least payoff, P1 wants to maximize their minimum gain over all strategies. Using such a strategy implies that they will always make at least this amount.

In the inverse scenario, P2 picks some strategy and P1 responds. P2 wants to minimize their loss but P1 wants to respond with the move that gains them the most. The worst case that P2 wants to mitigate is that P1 will pick the response to maximize P2's loss, so P2 wants to try to minimize the maximum loss. Finding the strategy that mins the max loss implies that using this strategy, P2 will always lose at most this amount.

Together, these two "safety strategies" to mitigate the worst case for each player eventually intersect into what is known as a "Nash Equilibrium". Consequently, we get the result that both strategies are optimal responses to each other and there is no incentive to switch since no more value can be gained/lost by switching.

In particular, this is point and value is defined and proven in John van Neumann's minimax theorem which states over the set of all (potentially mixed) strategies for P1 Δ^m (ie. a vector of length m that represents probability of each row which sums to 1) and all (potentially mixed) strategies for P2 Δ^n and a game A,

$$\max_{x \in \Delta^m} \min_{x \in \Delta^n} x^T A y = \min_{y \in \Delta^n} \max_{x \in \Delta^n} x^T A y$$

for a game value of $x^T A y$.

So for two player zero-sum games, if we find the pair of strategies that result in a Nash Equilibrium then we will know exactly how much is lost/gained worst case for each player and the game is considered "solved" since there will be no incentive for each player to switch strategies against the opponent who is playing optimally.

Matrix Games Formulated as a Linear Programming Problem

So, to solve game, we want to find the Nash Equilibrium between strategies x for P1 and y for P2 for a game represented by payoff matrix A. We set up the components of a linear programming problem:

Decision Variables

We consider P1's perspective. P1's decision is what strategy x to use where x is a vector of length m and x_i is the probability of choosing strategy i:

Conversely, P2's decision is what strategy y to use where y is a vector of length n and y_j is the probability of choosing strategy j:

Objective Function

The objective is to find the value of the zero-sum game v. For P1, that means to maximize this value so the objective for P1 is:

$$\max v$$

And for P2, this means to minimize this value:

$$\min v$$

Constraints

We have the following common constraints:

- Both x and y are probability vectors and must sum to 1:

$$\sum_i x_i = 1,\ \sum_j y_j = 1$$

- You cannot play a strategy with negative probability:

$$x_i \geq 0,\ y_j \geq 0$$

Then, we add one more constraint specific to each player:

- From above, the objective for P1 is to find some strategy that maximizes their minimum gain:

$$x^T A \geq v$$

- The objective for P2 is to find some strategy that minimizes their maximum loss:

$$Ay \leq v$$

LPP for P1

$$\begin{aligned} &\max && v \\ &\text{subject to} && x^T A \geq v \\ &&& \sum_i x_i = 1 \\ &&& x \geq 0 \end{aligned}$$

LPP for P2

$$\begin{aligned} \min \quad & v \\ \text{subject to} \quad & Ay \leq v \\ & \sum_j y_j = 1 \\ & y \geq 0 \end{aligned}$$

LP and Minimax

One last note is that these two LPPs are the dual for each other and thus by strong duality, their values v are equal and this is precisely the result of the minimax theorem above!

Solving Matrix Games using CVXPY

Using the above LPP formulation, we can create the following python function to solve an arbitrary zero-sum game using CVXPY:

```
def solve_P1(A):
  """
  Solve P1's optimal strategy for a game given by matrix A

  Parameters:
   - A: m x n payout matrix for the game

  Returns:
    - The value of the game and the optimal strategy x for P1
  """
  m, _ = np.shape(A)

  # X is the decision variables for the strategy
  X = cp.Variable(m)
  # V is the value of the game
  V = cp.Variable()
  objective = cp.Maximize(V)

  # P1 gets at least v for each choice of strategy
  constraint1 = [(X.T @ A) >= V]
  # Probabilities are >= 0
  constraint2 = [X.T >= 0]
  # X[i] must sum to 1
```

```
  constraint3 = [cp.sum(X.T) == 1]
  constraints = constraint1 + constraint2 + constraint3

  problem = cp.Problem(objective, constraints)
  problem.solve()

  return problem.value, X.value

def solve_P2(A):
  """
  Solve P2's optimal strategy for a game given by matrix A

  Parameters:
   - A: m x n payout matrix for the game

  Returns:
    - The value of the game and the optimal strategy y for P2
  """
  _, n = np.shape(A)

  # Y is the decision variables for the strategy
  Y = cp.Variable(n)
  # V is the value of the game
  V = cp.Variable()
  objective = cp.Minimize(V)

  # P2 gets at most v for each choice of strategy
  constraint1 = [(A @ Y) <= V]
  # Probabilities are >= 0
  constraint2 = [Y >= 0]
  # Y[j] must sum to 1
  constraint3 = [cp.sum(Y) == 1]
  constraints = constraint1 + constraint2 + constraint3

  problem = cp.Problem(objective, constraints)
  problem.solve()

  return problem.value, Y.value
```

Solving Rock-Paper-Scissors

The game of canonical Rock-Paper-Scissors is given by the matrix below where the player gains \$1 if they win or \$0 if they tie. Let $i, j = 0$ be Rock, $i, j = 1$ be Paper, and $i, j = 2$ be Scissors:

$$A = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

```
A = np.array([[0, -1, 1], [1, 0, -1], [-1, 1, 0]])

value_x, X = solve_P1(A)
print("Value: " + str(value_x))
print("P1 Strategy: " + str(X) + "\n")

value_y, Y = solve_P2(A)
print("Value: " + str(value_y))
print("P2 Strategy: " + str(Y))
```

```
Value: 2.4450394122394545e-10
P1 Strategy: [0.33333333 0.33333333 0.33333333]

Value: -2.4450394122394545e-10
P2 Strategy: [0.33333333 0.33333333 0.33333333]
```

So, each player should pick uniformly $[1/3, 1/3, 1/3]$ as their strategy such that P1 would gain at most \$0 and P2 would lose at most \$0.

Tech Reads in Super Smash Bros.

In the fighting game Super Smash Bros (and other fighting games in general), there is a game state called "advantage" in which a player (let's say P1) is playing to take a stock/life of player 2 and the other is in "disadvantage" in which they are playing to live and reset to a "neutral" game state or gain an advantage. There is a mechanic in Smash in which a player can "tech" when hitting the ground while in the "hitstun" state after getting attacked to remain invulnerable while moving left, right, or stay in-place. A player can also mistime the input and "miss" their tech which leaves them vulnerable and have to perform a slow standing animation. A "tech-chase" situation is when the advantage state player attempts to "read" (ie. correctly guess) the defensive option chosen by P2 to maintain advantage state and potentially take a stock.

In a tech situation, P1 will attempt to read one of 4 options that P2 can make:

1. Tech in-place
2. Tech left
3. Tech right
4. No tech

Let the payout be $a_{i,j} \in [0, 1]$ which represents the probability that P1 takes the stock for each situation. Then, consider the following matrix where row/col index 0 is in-place, 1 is left, 2 is right, and 3 is no option.

$$A = \begin{bmatrix} 1 & 0.3 & 0.3 & 0.8 \\ 0.25 & 1 & 0 & 0.6 \\ 0.25 & 0 & 1 & 0.6 \\ 0.5 & 0.3 & 0.3 & 1 \end{bmatrix}$$

```
A = np.array([
  [1, 0.25, 0.25, 0.8],
  [0.25, 1, 0, 0.6],
  [0.25, 0, 1, 0.6],
  [1, 0.3, 0.3, 1]
])

value_x, X = solve_P1(A)
print("Value: " + str(value_x))
print("P1 Strategy: " + str(np.round(X, 4)) + "\n")

value_y, Y = solve_P2(A)
print("Value: " + str(value_y))
print("P2 Strategy: " + str(np.round(Y, 4)))
```

```
Value: 0.44736842104900976
P1 Strategy: [0.     0.3684 0.3684 0.2632]

Value: 0.44736842104415886
P2 Strategy: [0.2105 0.3947 0.3947 0.    ]
```

Under these conditions, it is best for P1 to predict in-place 26%, left or right 37%, and never no-tech for the payout of taking a stock at least 45% of the time. Conversely, P2's best options are to pick in-place 21%, left or right 39.5%, and never no-tech for the payout of losing a stock at most 45% of the time.

Conclusion

The structure and properties of zero-sum games allows it to be solved using linear programming to leverage strong duality in finding the optimal strategies for either player and finding the value of the game which represents the minimum gain/maximum loss for each player in the worst case.

References

Axelrod, Robert, and William D. Hamilton. 1981. "The Evolution of Cooperation." *Science* 211 (4489): 1390–96. http://www.jstor.org/stable/1685895.

Wikipedia contributors. 2024. "Game Theory — Wikipedia, the Free Encyclopedia." https://en.wikipedia.org/w/index.php?title=Game_theory&oldid=1209544242.