### Topic 9 Walkthrough

#### 1. Calculate Nash Equilibrium

**Overview:** The calculate\_nash\_equilibrium function identifies the Nash equilibrium for specific, predefined payoff matrices representing different strategic games. Nash equilibria are strategy sets where no player can benefit by changing their strategy unilaterally.

**Detailed Pseudo Code:**

FUNCTION calculate\_nash\_equilibrium(payoff\_matrix)  
 CHECK if payoff\_matrix matches any predefined game scenarios  
 IF match found  
 RETURN the corresponding Nash equilibrium strategy profiles for both players  
 ELSE  
 RETURN None as a fallback for unexpected matrices  
END FUNCTION

**Implementation Guide:** - Use numpy for matrix comparison. - Define several known payoff matrices representing classic game theory problems. - Return the Nash equilibrium strategies for these predefined matrices, which include both pure and mixed strategy equilibria. - This function serves as an educational tool for demonstrating known results in game theory rather than a general solver.

#### 2. Solve Zero-Sum Game

**Overview:** solve\_zero\_sum\_game applies linear programming to find the optimal mixed strategy for the row player in a zero-sum game. Zero-sum games are those in which one player’s gain is exactly the other’s loss.

**Detailed Pseudo Code:**

FUNCTION solve\_zero\_sum\_game(payoff\_matrix)  
 DEFINE objective function and constraints for linear programming  
 USE scipy.optimize.linprog to solve the linear programming problem  
 IF solution found  
 EXTRACT and RETURN the game's value and the optimal strategy for the row player  
 ELSE  
 PRINT error message and RETURN None  
END FUNCTION

**Implementation Guide:** - Initialize the linear programming problem with the objective to maximize the row player’s payoff, translated into a minimization problem with negative payoffs. - Define constraints ensuring that against any strategy of the column player, the expected payoff for the row player is not less than a certain value. - Add an equality constraint for the strategy probabilities to sum up to 1. - Use linprog from scipy.optimize to find the solution. - Return the game value and optimal strategy if the problem is successfully solved.

#### 3. Simulate Prisoners Dilemma

**Overview:** The simulate\_prisoners\_dilemma function simulates the iterated Prisoner’s Dilemma game, given the strategies of two players. It’s a fundamental game in studying cooperation and competition.

**Detailed Pseudo Code:**

FUNCTION simulate\_prisoners\_dilemma(strategies, iterations)  
 INITIALIZE outcomes list  
 FOR each iteration  
 DETERMINE moves of both players based on their strategies  
 APPEND moves to outcomes list  
 RETURN outcomes list  
END FUNCTION

**Implementation Guide:** - Accepts a list of strategies for two players and the number of iterations as input. - Iteratively determine each player’s move based on their strategy. Note: In the provided code, moves are placeholders and should be determined by the players’ strategies. - Collect the outcomes of each iteration in a list. - Return the list of outcomes, showing how the game evolves over multiple iterations.