Национальный исследовательский университет ИТМО
Факультет информационных технологий и программирования Прикладная математика и информатика

Теория игр

Отчёт по лабораторной работе №2

Работу выполнили:
Гуревич Михаил
Трохан Александр
Соловьёв Роман

Преподаватель:
Москаленко Мария Александровна

Санкт-Петербург 2023

Лабораторная работа №2

Выполнили: Гуревич Михаил, Трохан Александр и Соловьёв Роман, M33001

✅ Полный код лабораторной работы с комментариями можно найти на Github, а отчёт в **Notion**.

Цель работы

Реализация метода решения в чистых или смешанных стратегиях игры по платёжной матрице.

Задачи

- Реализовать возможность ввода данных из файла в формате JSON.
- Упрощение платёжной матрицы путём анализа доминирующих стратегий.
- Реализация нахождения решения игры в чистых стратегиях или определения, что такого решения нет.
- Реализация применения симплекс-метода для поиска седловой точки в смешанных стратегиях для случаев, когда решения в чистых стратегиях не существует.

Ход работы

Реализация методов

Методы инициализации кастомного класса для игры и ввода данных из json’а

Описание метода: класс будет содержать платежную матрицу, её копию, над которой мы будем производить операции и оставшиеся для рассмотрения стратегии. При чтении из json’а мы инициализируем экземпляр класса от матрицы, полученной из json’а.

Реализация методов:

```
def __init__(self, matrix: np.ndarray):
      self.matrix = matrix.copy()
      self.original_matrix = matrix.copy()

      self.remaining_strategies = [list(range(matrix.shape[0])), list(range(matrix.shape[1]))]

@staticmethod
def from_json(path: str):
    with open(path) as f:
        matrix = np.array(json.load(f)["payoff_matrix"])
    return Game(matrix)
```

Метод упрощения платёжной матрицы путём анализа доминирующих стратегий

Описание метода: для каждого игрока перестаём рассматривать те стратегии, для которых существует такая стратегия, что для любого выбора стратегии соперником выигрыш данного игрока нестрого больше, чем по рассматриваемой стратегии. Так как удаление некоторых стратегий для одного игрока может повлиять на возможность удаления стратегий другого игрока, производим анализ итеративно, пока не перестанут происходить изменения.

Реализация метода:

```
def _eliminate_dominated_by_player(self, matrix: np.ndarray, player: Literal[0, 1]):
    """Eliminate dominated strategies for a given player."""

    dominated_strategies = []

    for i in range(matrix.shape[player]):
        for j in range(matrix.shape[player]):
            if i != j and (
                (player == 0 and np.all(matrix[i, :] <= matrix[j, :])) or
                    (player == 1 and np.all(matrix[:, i] >= matrix[:, j]))
            ):
                dominated_strategies.append(i)
                break

    for i, strategy in enumerate(dominated_strategies):
        del self.remaining_strategies[player][strategy - i]

    return np.delete(matrix, dominated_strategies, axis=player)

def _eliminate_dominated(self):
    """Eliminate dominated strategies for both players."""

    while True:
        matrix = self.matrix.copy()
```

```
        matrix = self._eliminate_dominated_by_player(matrix, 0)
        matrix = self._eliminate_dominated_by_player(matrix, 1)
        if matrix.shape == self.matrix.shape:
            break
        self.matrix = matrix
```

Метод поиска седловой точки

Описание метода: ищем максимальное среди минимальных значений по строкам и минимальное среди максимальных значение по столбцам. Если это один и тот же элемент матрицы, то игра имеет решение в чистых стратегиях.

При поиске седловой точки мы ищем такую комбинацию ходов обоих игроков, при которой ни один из них не может улучшить свой результат, если он изменит свою стратегию. Такая комбинация называется равновесной точкой.

Поиск максимина и минимакса при определении седловой точки позволяет игрокам выбрать оптимальную стратегию в условиях неопределенности и достичь равновесия. Если оба игрока будут играть оптимально, то они не смогут изменить свою стратегию и получить больший выигрыш или меньшие потери.

Реализация метода:

```
def _get_lower_value(self) -> tuple[float, np.ndarray]:
        """Get the lower value of the game and the corresponding strategy."""

        min_ = np.min(self.matrix, axis=1)
        maxmin = np.max(min_)
        return maxmin, np.nonzero(min_ == maxmin)[0]

def _get_upper_value(self) -> tuple[float, np.ndarray]:
    """Get the upper value of the game and the corresponding strategy."""

    max_ = np.max(self.matrix, axis=0)
    minmax = np.min(max_)
    return minmax, np.nonzero(max_ == minmax)[0]

def _get_saddle_point(self) -> tuple[float, tuple[int, int]]:
    """Get the saddle point of the game and the corresponding strategy."""

    maxmin, maxmin_strategies = self._get_lower_value()
    minmax, minmax_strategies = self._get_upper_value()
    if maxmin == minmax:
        for strategy in product(maxmin_strategies, minmax_strategies):
            if self.matrix[strategy] == maxmin:
                return maxmin, strategy
```

Метод решения в чистых стратегиях

Описание метода: упрощаем платёжную матрицу анализом доминирующих решений, ищем седловую точку. Если нашли — решение в чистых стратегиях есть, иначе его нет.

Седловая точка — это точка в матричной игре, где одна стратегия игрока является оптимальной для всех возможных стратегий другого игрока.

Чистые стратегии — это стратегии игры, где игрок выбирает только одно действие, без использования вероятностных распределений. Например, в игре крестики-нолики чистой стратегией может быть выбор определенной клетки для хода.

Необходимо найти седловую точку, потому что она позволяет определить оптимальные стратегии для игроков и достичь наилучшего результата для обоих игроков. Если седловой точки нет, то это означает, что игроки не могут достичь наилучшего результата при использовании чистых стратегий, и им нужно использовать смешанные стратегии (вероятностные распределения выбора действий).

Реализация метода:

```
def solve_in_pure_strategies(self):
    """Solve the game in pure strategies."""

    self._eliminate_dominated()
    saddle_point = self._get_saddle_point()
    if saddle_point is not None:
        return saddle_point
    else:
        raise ValueError("The game has no saddle point and cannot be solved in pure strategies.")
```

Метод применения симплекс-метода для поиска седловой точки в смешанных стратегиях

Описание метода: представляем платежную матрицу как коэффициенты в неравенствах задачи линейного программирования и решаем для каждого игрока. Для первого коэффициенты берутся построчно и ограничение ≤ 1. Для второго игрока коэффициенты берутся по столбцам и ограничение ≥ 1. При этом если в матрице были неположительные элементы, мы прибавляем наименьший элемент ко всем, чтобы матрица была положительно определена. После работы симплекс-метода получаем смещённую цену игры (из-за прибавления наименьшего элемента), смешанные стратегии игроков.

Реализация метода:

```
def _solve_linear_programs(self):
    """Solve primal and dual linear programs associated to the game."""

    positive_matrix = self.matrix.copy()
    if any(positive_matrix.flatten() <= 0):
        positive_matrix -= np.min(positive_matrix) - 1

    primal_constraints = []
    for row in positive_matrix:
        primal_constraints.append(Constraint(row, "leq", 1))
    primal = LinearProgram([1] * positive_matrix.shape[1], "max", primal_constraints)

    dual_constraints = []
    for column in positive_matrix.T:
        dual_constraints.append(Constraint(column, "geq", 1))
    dual = LinearProgram([1] * positive_matrix.shape[0], "min", dual_constraints)

    primal_solution = SimplexSolver(primal).solve()
    dual_solution = SimplexSolver(dual).solve()

    if primal_solution[1] != dual_solution[1]:
        raise ValueError("An error occured: the primal and dual solutions have different values.")

    return 1 / primal_solution[1], primal_solution[0], dual_solution[0]
```

Метод определения смешанных стратегий

Описание метода: упрощаем платёжную матрицу анализом доминирующих решений, запускаем метод нахождения смешанных стратегий. После этого считаем несмещённую цену игры.

Реализация метода:

```
def solve_in_mixed_strategies(self) -> tuple[float, tuple[tuple[Fraction], tuple[Fraction]]]:
    """Solve the game in mixed strategies."""

    self._eliminate_dominated()
    value, primal_solution, dual_solution = self._solve_linear_programs()

    optimal_strategy_player_1 = tuple(np.multiply(dual_solution, value).tolist())
    optimal_strategy_player_2 = tuple(np.multiply(primal_solution, value).tolist())

    real_value = np.dot(np.dot(optimal_strategy_player_1, self.matrix), optimal_strategy_player_2)

    return real_value, (optimal_strategy_player_1, optimal_strategy_player_2)
```

Вывод о работе методов решения в чистых и смешанных стратегиях:

Для решения игры в чистых стратегиях используется простой подсчёт, которому достаточно по одному разу проитерироваться по каждому элементу матрицы. Но в реальном мире “игры” создаются/возникают исходя из того, что лучшего ответа быть не должно, из-за чего игры с чистыми стратегиями встречаются гораздо реже. Из-за этого приходится использовать более сложный математический аппарат (симплекс-метод), чтобы получить информацию о смешанных стратегиях и оценить цену игры.

Вывод

Был реализован парсинг платёжной матрицы из JSON-файла и решение этой игры в чистых или смешанных стратегиях.