Национальный исследовательский университет ИТМО
Факультет информационных технологий и программирования
Прикладная математика

Теория игр

Отчет по лабораторной работе №2

Работу выполнили:
Обиджанов Алишер
Казаков Андрей
Кузнецов Павел

Преподаватель:
Свинцов М.В.

Санкт-Петербург
2023

1 Постановка задачи

1. Реализуйте возможность ввода данных из файла в формате JSON, который содержит матрицу игры.
2. Упростите платежную матрицу путем анализа доминирующих стратегий.
3. Если это возможно найдите решение игры в чистых стратегиях. Определите оптимальные стратегии и соответствующую цену игры.
4. Если решение в чистых стратегиях найти невозможно, примените симплекс- метод для поиска седловой точки в смешанных стратегиях. Определите сме- шанные стратегии и соответствующую цену игры

2 Реализация

2.1 Реализуйте возможность ввода данных из файла в формате JSON, который содержит матрицу игры.

Пример json строки:

```
{"matrix":[[10,4,11,7],[7,6,8,20],[6,2,1,11]]}
```

Парсинг строки:

```
def parse_problem(json_str):
    try:
        data = json.loads(json_str)

        if "matrix" in data:
            matrix = data["matrix"]

            if isinstance(matrix, list) and all(isinstance(line, list) for line in matrix):
                return np.array(matrix)
            else:
                raise ValueError("Неверный формат матрицы в JSON.")
        else:
            raise ValueError("Отсутствует поле 'matrix' в JSON.")

    except json.JSONDecodeError as e:
        raise ValueError(f"Ошибка декодирования JSON: {e}")
```

2.2 Упростите платежную матрицу путем анализа доминирующих стратегий

Сперва разделим каждый элемент на максимальный общий делитель и избавимся от отрицательных значений

```
def simplefy_matrix(matrix):
    if matrix.dtype != np.float64:
        matrix //= np.gcd.reduce(matrix.flatten())

    matrix -= matrix.min() if matrix.min() < 0 else 0

    return matrix
```

Далее будем сокращать строки и столбцы путем анализа доминирующих стратегий до тех пор пока это возможно

```
def reduce_matrix(matrix):
abbreviated = True
deleteLines = []
deleteColumn = []

while abbreviated:
    ilines = list(set(range(matrix.shape[0]))-set(deleteLines))
    icolumns = list(set(range(matrix.shape[1]))-set(deleteColumn))

    abbreviated = False
    for iline1 in ilines:
        for iline2 in ilines:
            if iline1 != iline2 and np.all(matrix[iline1,icolumns] > matrix[iline2,icolumns]):
                deleteLines.append(iline2)
                # print("delete line:",iline2)
                abbreviated = True
                break


    for icolumn1 in icolumns:
        for icolumn2 in icolumns:
            if icolumn1 != icolumn2 and np.all(matrix[ilines,icolumn1] > matrix[ilines,icolumn2]):
                deleteColumn.append(icolumn1)
                # print("delete column:",icolumn1)
                abbreviated = True
                break

matrix = np.delete(matrix, deleteColumn, axis=1)
matrix = np.delete(matrix, deleteLines, axis=0)
return matrix, sorted(deleteLines), sorted(deleteColumn)
```

2.3 Если это возможно найдите решение игры в чистых стратегиях. Определите оптимальные стратегии и соответствующую цену игры.

В чистых стратегиях можно решить задачу только в том случае, если у матрицы есть седловая точка. Ее можно найти путем вычисления минимаксных значений. Так же её можно найти, если мы сократили нашу изначальную матрицу до размера 1:1. Использую второй способ.

```
def strategies(r_matrix, deleteLines, deleteColumn):
Pa = []
Qb = []

if r_matrix.shape[0] == 1 and r_matrix.shape[1] == 1:
    for i in range(len(deleteLines) + 1):
        Qb.append(0) if i in deleteLines else Qb.append(1)
    for i in range(len(deleteColumn) + 1):
        Pa.append(0) if i in deleteColumn else Pa.append(1)
else:
    f = np.ones(r_matrix.shape[1])
    b = np.ones(r_matrix.shape[0])
    result = linprog(c=f, A_eq=r_matrix, b_eq=b)
    if result.success:
        strategy_2 = result.x
        strategy_1 = r_matrix.dot(strategy_2)
        game_value = 1 / result.fun
        strategy_2 = [x / sum(strategy_2) for x in strategy_2]
        strategy_1 = [x / sum(strategy_1) for x in strategy_1]
        k=0
        for i in range(len(deleteLines) + len(strategy_1)):
            if i in deleteLines: Pa.append(0)
            else:
                Pa.append(strategy_1[k])
                k+=1
        k=0
        for i in range(len(deleteColumn) + len(strategy_2)):
            if i in deleteColumn: Qb.append(0)
            else:
                Qb.append(strategy_2[k])
                k+=1

return np.array(Pa), np.array(Qb)
```

Вычислить цену игры можно по формуле $Q_b^T * M * P_a$

```
    def game_price(matrix, Pa, Qb):
    return Qb.T @ matrix @ Pa
```

2.4 Запуск кода

```
json_string = '{"matrix":[[-2,1],[2,-1]]}'
try:
    matrix = parse_problem(json_string)
    matrix = simplefy_matrix(matrix)
    r_matrix, deleteLines, deleteColumn = reduce_matrix(matrix)
    print(f"reduced matrix={r_matrix}, delete lines:{deleteLines}, delete column:{deleteColumn}")

    Pa, Qb = strategies(r_matrix, deleteLines, deleteColumn)
    y = game_price(matrix, Pa, Qb)

    print(f"Pa={Pa}, Qb={Qb}, y={y}")

except ValueError as e:
    print(f"Ошибка: {e}")
```

3 Вывод

В ходе данной лабораторной работы был успешно реализован алгоритм для решения матричной игры, предназначенной для двух игроков. Проведен анализ сценариев как в чистых стратегиях, так и в смешанных. Для эффективного решения матрицы в смешанных стратегиях применен симплекс-метод. Полученные стратегии обеспечивают оптимальные решения, а использование симплекс-метода добавляет методу дополнительную гибкость и применимость к различным сценариям.