Метод решения

1) Сначала проверяем есть ли седловая точка. Если есть, то вычисляем V, p, q. Если нет, то переходим к след.пункту решения.
2) Если матрица неположительная, то вычислим её минимальный элемент, прибавим модуль этого элемента ко всем элементам матрицы. После этого прибавляем ко всем элементам ещё 1. Матрица становится положительной. Строим пару двойственных задач линейного программирования.
   Задача ЛП для первого игрока:

   $F = \sum y_j = 1/V \rightarrow min,\ j = 1,\ ...,\ m$

   $\sum_{i=1}^{n} a_{ij} y_j \geq 1,\ \forall j = 1,\ ...,\ m$

   $y_j \geq 0\ \forall j = 1,\ ...,\ m$, где $y_j = p_j/V\ \forall j = 1,\ ...,\ m$

   Задача ЛП для второго игрока:

   $F = \sum x_i = 1/V \rightarrow max,\ i = 1,\ ...,\ n$

   $\sum_{j=1}^{m} a_{ji} x_i \leq 1,\ i = 1,\ ...,\ n$

   $x_i \geq 0\ \forall i = 1,\ ...,\ n$, где $x_i = q_i/V\ \forall i = 1,\ ...,\ n$

   Эти задачи могут быть решены симплекс-методом.
3) Этапы алгоритма симплекс-метода:
   a) Введение базисных переменных, запись задачи в симплекс-таблицу. Количество строк равно количеству уравнений ограничений, количество столбцов соответствует количеству свободных переменных. В первом столбце таблицы - базисные переменные, в в первой строке - небазисные. В нижнюю строку записываются коэффициенты целевой функции. В правом столбце - свободные члены.
   b) Анализ строки целевой функции на оптимальность. Если хотя бы 1 коэффициент отрицательный, то решение не оптимально. Если все неотрицательные, то решение найдено.
   c) Определяем колонки с отрицательными элементами в строке целевой функции. Выбираем элемент, наибольший по модулю.Этот столбец называется разрешающим(столбец j). Для определения разрешающей строки находим отношения свободных чисел к элементам разрешающей колонки. Строка, которой соответствует наименьшее положительное отношение, называется разрешающей(строка i). Элемент на пересечении разрешающей колонки и разрешающей строки - разрешающий.
   d) Разрешающий элемент указывает на базисную и свободную переменные, которые необходимо поменять местами в симплекс-таблице. Переменные x_{n+i} и x_j меняем местами. Преобразуем разрешающий элемент: $a_{ij} - 1$. Результат записываем в новую таблицу.Элементы разрешающей строки делим на разрешающий

элемент данной симплекс-таблицы. Элементы разрешающей колонки делим на разрешающий элемент данной симплекс-таблицы и умножаем на -1. Преобразование остальных элементов таблицы происходит по правилу «прямоугольника»: мысленно вычеркиваем прямоугольник. Одна вершина располагается в клетке, значение которой преобразуем, а вторая, диагональная первой – в клетке с разрешающим элементом. Преобразованное значение клетки будет равно прежнему значению данной клетки минус дробь, в знаменателе которой разрешающий элемент, а в числителе произведение двух других вершин. В результате данных преобразований получили новую симплекс-таблицу. Переход к пункту b) .

Прим. После последней итерации симплекс-метода вычисляем оптимальный план и для первого, и для второго игроков.

Описание программы

В main.ipynb описаны тестовые матрицы, для каждой из которых выведен результат работы функций nash_equilibrium(a) и visualize(p) из nash.py
В nash.py описаны функции, необходимые для решения антагонистической матричной игры и его визуализации:

- check_natural(a) - проверка на положительность элементов исходной матрицы
- check_saddle_point(a) - проверка на седловую точку
- nash_equilibrium(a) - вычисляет значение игры и оптимальные стратегии
- output_decision(x) - вывод решения
- output_matrix(a) - вывод матрицы
- simple_win_matrix(a) - редуцирование матрицы выигрыша
- simplex_metod(a) - симплекс-метод
- visualize(p) - визуализация решений

Пакет Python

Пакет Python - это организованные модули Python. Модуль Python - один файл. Создание пакета с помощью __init__.py позволяет упростить разработку проектов на Python. Он предоставляет способ сгруппировать большие папки в один импортируемый модуль.

Unit-тесты

В файле test.py описаны функции test_1 - test_8, которые используются для проверки результатов работы программы. Они сверяют векторы стратегии и цену игры, полученными в результате выполнения функции nash_equilibrium, с ожидаемым(верным) решением. Для этого используется функция библиотеки unittest assertAlmostEqual(a,b).

В файле test_runner.py описана функция, вызывающая тесты и направляющая отчет об ошибках в стандартный поток вывода. Запуск тестов: nosetests test_runner.

В результате работы

1) написана функция nash_equilibrium(a), которая принимает матрицу выигрыша и возвращает значение игры и оптимальные стратегии первого и второго игроков
2) проиллюстрирована работа кода путём решения нескольких игр и визуализации спектров оптимальных стратегий игроков в Jupyter
    - спектр оптимальной стратегии состоит из одной точки (т.е. существует равновесие Нэша в чистых стратегиях)
    - спектр оптимальной стратегии неполон (т.е. некоторые чистые стратегии не используются)
    - спектр оптимальной стратегии полон
3) оформлено решение в виде пакета [fn:packaging]
4) написаны unit-тесты для функции nash_equilibrium [fn:testing]

Для решения использовались следующие библиотеки

- numpy - поддержка многомерных массивов и функций, предназначенных для работы с многомерными массивами
- fractions - поддержка рациональных чисел
- math - функционал для работы с числами
- matplotlib.pyplot - визуализация решений
- scipy.optimize - выполнение научных и инженерных расчётов
- unittest, nose - тестирование функций

Список литературы

- См. Васин А.А., Краснощеков П.С., Морозов В.В. Исследование операций - М.: Издательский центр “Академия”.
- [fn:packaging] https://python-packaging.readthedocs.io/en/latest/
- [fn:testing] https://pythontesting.net/framework/nose/nose-introduction/
- https://pythonworld.ru/moduli/modul-unittest.html
- https://devpractice.ru/unit-testing-in-python-part-1/
- https://habr.com/ru/company/otus/blog/433358/