README для задания 1

Серегина Ирина, Хает Софья, Дербилов Александр, Чжи Инжуй

October 2020

1 Описание алгоритма

1.1 Теоретическое описание алгоритма

В математике под матричными играми понимается игра двух лиц с нулевой суммой, имеющих конечное число стратегий. Выигрыш определяется матрицей игры (матрицей платежей), она же является Нормальной формой игры. Основная задача - свести матричную игру к алгоритму симплекс-метода решения пары взаимодвойственных задач линейного программирования. Дадим определения линейного программирования и двойственной задачи линейного программирования:

> Линейное программирование — математическая дисциплина, посвящённая теории и методам решения экстремальных задач на множествах nxn-мерного векторного пространства, задаваемых системами линейных уравнений и неравенств.
>
> Задача линейного программирования – это задача, в которой требуется найти максимум или минимум (оптимум) функции, называемой функцией цели, при ограничениях, заданных системой линейных неравенств или уравнений.
>
> Двойственная задача для заданной задачи линейного программирования — это другая задача линейного программирования, которая получается из исходной (прямой) задачи следующим образом:
>
> - Каждая переменная в прямой задаче становится ограничением двойственной задачи;
>
> - Каждое ограничение в прямой задаче становится переменной в двойственной задаче;

◦ Направление цели обращается – максимум в прямой задаче становится минимумом в двойственной, и наоборот.

Пусть игра задана платёжной матрицей A, имеющей размерность m×n. Необходимо найти решение игры, т.е. найти значение игры и определить оптимальные смешанные стратегии первого и второго игроков:
$P^* = \begin{pmatrix} p_1^* & p_2^* & \dots & p_m^* \end{pmatrix}$ $Q^* = \begin{pmatrix} q_1^* & q_2^* & \dots & q_n^* \end{pmatrix}$
где P^* и Q^* - векторы, компоненты которых p_i^* и p_j^* характеризуют вероятности применения чистых стратегий i и j соответственно первым и вторым игроками и соответственно для них выполняются соотношения:
$p_1^* + p_2^* + \ldots + p_m^* = 1; q_1^* + q_2^* + \ldots + q_n^* = 1.$

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

В матрице A p_i^* - это вектор строка (т.е. $p_i^* = (a_{i1}a_{i2}\ldots a_{im})$), q_i^* - вектор столбец ($q_i^* = (a_{1i}a_{2i}\ldots a_{ni})$).

1.1.1 Нахождение оптимальной стратегии первого игрока

> Стратегия – это совокупность правил, определяющих выбор варианта действий при каждом личном ходе в зависимости от сложившейся ситуации. Оптимальная стратегия игрока – стратегия, обеспечивающая наилучшее положение в данной игре, т.е. максимальный выигрыш.

Найдём сначала оптимальную стратегию первого игрока P^*. Эта стратегия должна обеспечить выигрыш первому игроку не меньше V, т.е. V , при любом поведении второго игрока, и выигрыш, равный V , при его оптимальном поведении, т.е. при стратегии Q^*. Цена игры V нам пока неизвестна. Без ограничения общности, можно предположить её равной некоторому положительному числу V>0. Действительно, для того, чтобы выполнялось условие $V > 0$, достаточно, чтобы все элементы матрицы A были неотрицательными.
Предположим, что первый игрок A применяет свою оптимальную стратегию P^*, а второй игрок B свою чистую стратегию j -ю.

> Чистая стратегия даёт полную определённость, каким образом игрок продолжит игру. В частности, она определяет результат для каждого возможного выбора, который игроку может придётся сделать.

Тогда средний выигрыш (математическое ожидание) первого игрока А будет равен:
$a_j = \sum_{i=0}^{m} a_{ij} p_i^* = a_{1j} p_1^* + a_{2j} p_2^* + \ldots + a_{mj} p_m^*$;
Оптимальная стратегия первого игрока (А) обладает тем свойством, что при любом поведении второго игрока (В) обеспечивает выигрыш первому игроку, не меньший, чем цена игры V; значит, любое из чисел a_j не может быть меньше V (V). Следовательно, при оптимальной стратегии, должна выполняться следующая система неравенств:

$$\begin{cases} a_{11}p_1^* + a_{21}p_2^* + \ldots + a_{m1}p_m^* \geq V \\ a_{12}p_1^* + a_{22}p_2^* + \ldots + a_{m2}p_m^* \geq V \\ \ldots \\ a_{1n}p_1^* + a_{2n}p_2^* + \ldots + a_{mn}p_m^* \geq V \end{cases}$$

Разделим неравенства на положительную величину V и введём обозначения:
$y_1 = \frac{p_1^*}{V}, y_2 = \frac{p_2^*}{V}, \ldots, y_m = \frac{p_m^*}{V}$

$y_1 \geq 0, y_2 \geq 0, \ldots, y_m \geq 0.$

Тогда условия запишутся в виде:

$$\begin{cases} a_{11}y_1^* + a_{21}y_2^* + \ldots + a_{m1}y_m^* \geq 1 \\ a_{12}y_1^* + a_{22}y_2^* + \ldots + a_{m2}y_m^* \geq 1 \\ \ldots \\ a_{1n}y_1^* + a_{2n}y_2^* + \ldots + a_{mn}y_m^* \geq 1 \end{cases}$$

где $y_1, y_2, \ldots, y_m$ - неотрицательные переменные. В силу системы и того, что $p_1^* + p_2^* + \ldots + p_m^* = 1$ переменные $y_1, y_2, \ldots, y_m$ удовлетворяют условию, которое обозначим через F:
$F = y_1 + y_2 + \ldots + y_m = \frac{1}{V}$
Поскольку первый игрок свой гарантированный выигрыш (V) старается сделать максимально возможным (V$\rightarrow$ max) , очевидно, при этом правая часть F $1_{\overline{V}}$ - $\rightarrow$min - принимает минимальное значение. Таким образом, задача решения антагонистической игры для первого игрока свелась к следующей математической задаче:

определить неотрицательные значения переменных $y_1, y_2, \ldots, y_m$, чтобы они удовлетворяли системе функциональных линейных ограничений в виде неравенств, системе общих ограничений и минимизировали целевую функцию F:
$F = y_1 + y_2 + \ldots + y_m \rightarrow \min$

Это типичная задача линейного программирования (двойственная) и она может быть решена симплекс - методом. Таким образом, решая задачу линейного программирования, мы можем найти оптимальную стратегию $P^* = \begin{pmatrix} p_1^* & p_2^* & \ldots & p_m^* \end{pmatrix}$ игрока А.

1.1.2 Нахождение проигрышной стратегии второго игрока

Найдём теперь оптимальную стратегию $Q^* = \begin{pmatrix} q_1^* & q_2^* & \ldots & q_n^* \end{pmatrix}$ игрока В. Всё будет аналогично решению игры для игрока А, с той разницей, что игрок В стремиться не максимизировать, а минимизировать выигрыш (по сути дела его проигрыш), а значит, не минимизировать, а максимизировать величину , т.к. V →min. Вместо условий должны выполняться условия:

$$\begin{cases} a_{11}x_1^* + a_{21}x_2^* + \ldots + a_{n1}x_n^* \leq 1 \\ a_{12}x_1^* + a_{22}x_2^* + \ldots + a_{n2}x_n^* \leq 1 \\ \ldots \\ a_{1m}x_1^* + a_{2m}x_2^* + \ldots + a_{nm}x_n^* \leq 1 \end{cases}$$

где $x_1 = \frac{q_1^*}{V}, x_2 = \frac{q_2^*}{V}, \ldots, x_n = \frac{q_n^*}{V}$

Таким образом, задача решения антагонистической игры для второго игрока свелась к следующей математической задаче: определить неотрицательные значения переменных $x_1, x_2, \ldots, x_n$, чтобы они удовлетворяли системе функциональных линейных ограничений в виде неравенств, системе общих ограничений и максимизировать целевую функцию F':

$F' = x_1 + x_2 + \ldots + x_n \rightarrow \max$.

Это типичная задача линейного программирования (прямая) и она может быть решена симплекс - методом. Таким образом, решая прямую задачу линейного программирования, мы можем найти оптимальную стратегию $Q^* = \begin{pmatrix} q_1^* & q_2^* & \ldots & q_n^* \end{pmatrix}$ игрока В.

1.1.3 Симплекс - метод

Симплекс-метод — алгоритм решения оптимизационной задачи линейного программирования путём перебора вершин выпуклого многогранника в многомерном пространстве.

Сущность метода: построение базисных решений, на которых монотонно убывает линейный функционал, до ситуации, когда выполняются необходимые условия локальной оптимальности.

Симплекс-метод позволяет эффективно найти оптимальное решение. Основной принцип метода: вычисления начинаются с какого-то «стартового» базисного решения, а затем ведется поиск решений, «улучшающих» значение целевой функции. Это возможно только в том случае, если возрастание какой-то переменной приведет к увеличению значения функционала.

Необходимые условия для применения симплекс-метода:

- Задача должна иметь каноническую форму.
- У задачи должен быть явно выделенный базис.

Выбираем переменную, которую будем вводить в базис. Это делается в соответствии с указанным ранее принципом: мы должны выбрать переменную, возрастание которой приведет к росту функционала. Выбор происходит по следующему правилу:

- Если задача на минимум – выбираем максимальный положительный элемент в последней строке.
- Если задача на максимум – выбираем минимальный отрицательный.

Такой выбор, действительно, соответствует упомянутому выше принципу: если задача на минимум, то чем большее число вычитаем – тем быстрее убывает функционал; для максимума наоборот – чем большее число добавляем, тем быстрее функционал растет.
Выбираем переменную, которую будем вводить в базис. Для этого нужно определить, какая из базисных переменных быстрее всего обратится в нуль при росте новой базисной переменной. Алгебраически это делается так:

- Вектор левых частей почленно делится на ведущий столбец
- Среди полученных значений выбирают минимальное положительное (отрицательные и нулевые ответы не рассматривают)

(Такая строка называется ведущей строкой и отвечает переменной, которую нужно вывести из базиса.)
Ищем элемент, стоящий на пересечении ведущих строки и столбца. Далее начинается процесс вычисления нового базисного решения. Он происходит с помощью метода Жордана-Гаусса.

- Новая Ведущая строка = Старая ведущая строка / Ведущий элемент
- Новая строка = Новая строка – Коэффициент строки в ведущем столбце * Новая Ведущая строка

Теперь находим базисные переменные - единичные столбцы и соответствующие номера строк, в которых стоят единицы.
После этого проверяем условие оптимальности. Если полученное решение неоптимально – повторяем весь процесс снова.

Условие оптимальности полученного решения:

- Если задача на максимум – в строке функционала нет отрицательных коэффициентов (т.е. при любом изменении переменных значение итогового функционала расти не будет).
- Если задача на минимум – в строке функционала нет положительных коэффициентов (т.е. при любом изменении переменных значение итогового функционала уменьшаться не будет).

1.2 Описание функций программы

1.2.1 is_there_any_seddle_point

Функция is_there_any_seddle_point(a) получает на вход матрицу a, находит минимаксный элемент и максиминный элемент. Если это один и тот же элемент, то возвращаем его координаты (в виде вектора) в матрице a, иначе возвращаем пустой вектор (None, None).

1.2.2 transform_simplex_table

Функция transform_simplex_table(a, row, col) получает на вход матрицу a, номера ведущих строки (row) и столбца (col). Номируем ведущую строку - делим на ведущий элемент. Методом Гаусса обнуляем все остальные элементы ведущего столбца.

1.2.3 find_leading_col

Функция find_leading_col(a) получает на вход матрицу a, находит ведущий столбец в матрице - min отрицательный элемент в 0 строке (без a[0][0]). Если такого нет (все неотрицательные), то возвращаем -1, иначе возвращаем номер столбца.

1.2.4 find_leading_row

Функция find_leading_row(a, leading_col) получает на вход матрицу a и номер ведущего столбца leading_col. Находим ведущую строку - по минимальному из "приведенного"0 разрешающего столбца (под приведением понимаем деление на элемент соответствующей строки ведущего столбца). Возвращаем -1, если минимальный элемент отрицательный, иначе - номер строки (ведущей) минимального элемента.

1.2.5 check_b_positive

Функция check_b_positive(a) получает на вход матрицу a. Проверка правильного заполнения матрицы для симплекс метода, проверка на истинность выражения : если ложно, то завершает программу и выводит ошибку.

1.2.6 find_basis_variable

Функция find_basis_variable(a, col) получает на вход матрицу a и номер столбца col. Проверяем является ли столбец базисным, если да, то возвращаем номер строки, в которой стоит 1, иначе возвращаем -1.

1.2.7 optimal_strategy

Функция optimal_strategy(a, b) получает на вход матрицу a и число b = - минимальное значение исходной матрицы +1. Находим базисные переменные (столбцы) (-> find_basis_variable()) - и значение в соответствующей строке разрешающего столбца (или 0) записываем в вектор q. Если же переменная не базисная, то записываем в вектор p соответствующее значение из 0 строки. Нормируем вектора p и q и находим значение исходной игры, не забывая про b - возвращаем их.

1.2.8 nash_equilibrium

Функция nash_equilibrium(a) на вход получает матрицу, на выходе возвращает вектора оптимальных стратегий и значение игры. Сначала ищем седловую точку - функция is_there_any_seddle_point(a), если она есть, то формируем для нее вектора оптимальных стратегий, иначе переходим к исследованию смешанных стратегий: для уменьшения объемов вычислений удалим доминируемые стратегии.
Переходим к симплекс методу: сначала сделаем все эелементы матрицы положительными. Составляем новую матрицу для симплекс метода - функция simplex_table(), запускаем цикл для непосредственно самого симплекс метода, находим ведущий столбец find_leading_col(). Если его нет, то запускаем optimal_strategy() для нахождения оптимальной стратегии).
Далее ищем ведущую строку find_leading_row(), изменяем исходную матрицу transform_simplex_table() с учетом полученных ведущих строк, столбцов.

1.2.9 visualization

На вход функция visualization(p) получает вектор оптимальной стратегии. По данному вектору строит график: задается ось X с шагом 1 (необходимая длина - количесво элементов вектора), на оси Y задаются значения соответствующих элементов. Далее происходит оформление графика (название графика, выбор стиля).

1.2.10 main

Основная функция - на вход матрица, либо ничего и тогда осуществляется ввод с клавиатуры. Далее для матрицы вызывается функция nash_equilibrium() и для полученных векторов оптимальных стратегий выполняется визиулизация visualization(). Выводим значение игры и отпимальные стратегии.

2 Инструкция по запуску

Установка pip если его нет:

- Linux или macOS:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python get-pip.py
```

- Скачать get-pip.py в папку на вашем компьютере. Откройте окно командной строки и перейдите в папку, содержащую get-pip.py. Затем запустите python get-pip.py.
  Скачать пакет и установить его:

  ```
  cd nash-package
  pip install
  ```

  Использование пакета:

  ```
  python
  ≫ import nash
  ≫ nash.main()
  2
  2
  5 0
  6 0

  Game value is : 0.0
  optimal strategy for 1st player : [1.0, 0.0]
  optimal strategy for 2nd player : [0.0, 1.0]
  ≫
  ```

- Запуск тестов:
  nosetests nose.py

3 Вклад каждого участника

Серегина Ирина и Хает Софья разработали алгоритм решения (разбили большую задачу на отдельные этапы, которые реализовывались соответствующими функциями).

Серегина Ирина - реализовала симплекс метод и функцию nash_equilibrium. Также компоновала функции участников в общий код и вносила финальные правки. Выгружала проект в GIT.

Хает Софья - реализовывала функцию visualization и писала данный README;

Дербилов Александр - написал тесты;

Чжи Инжуй - реализовывал функцию is_there_any_seddle_point и формировал общую функцию main.

4 Литература

- Васин А.А., Морозов В.В. "Введение в теорию игр с приложениями к экономике"(учебное пособие). М.: 2003;
- Васин А.А., Краснощеков П.С., Морозов В.В. Исследование операций - М.: Издательский центр “Академия”.
- "Методы оптимизации лекции Д.А. Кропотов (МФТИ);
- https://math.semestr.ru/games/linear-programming.php ;
- http://fsweb.info/latex/ ;
- https://habr.com/ru/post/425609/ ;
- https://habr.com/ru/post/474286/ ;
- https://www.coursera.org/learn/matematicheskaya-teoria-igr ;
- https://python-graph-gallery.com ;
- http://www.astronet.ru/db/msg/1202050/frac.html ;