МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики

ЗАДАНИЕ №1

ОТЧЕТ

о выполненном задании

группы студентов 311 учебной группы факультета ВМК МГУ
Александр Борисов
Иван Наумов
Охим Елена
Максим Гончаренко

Москва, 2020 г.

Содержание

1 Постановка задачи и подход к ее решению

Постановка задачи состоит в численном решении антагонистической матричной игры.

1.1 Цели и задачи практической работы

- Написание кода, решающего матричную игру путем сведения ее к паре двойственных задач линейного програмирования.
- Иллстрация работы данного кода путем визуализации спектров оптимальных стратегий.
- Написание автоматических тестов для данного решения.
- Знакомство с языком программирования Python, библиотекой SciPy и интерактивной средой разработки Jupyter и с системой тестирования Nose.

2 Теоритические основы

Задача линейного программирования — это задача поиска неотрицательных значений параметров, на которых заданная линейная функция достигает своего максимума или минимума при заданных линейных ограничениях.

2.1 Симплекс-метод

Симплекс-метод — алгоритм решения оптимизационной задачи линейного программирования путём перебора вершин выпуклого многогранника в многомерном пространстве. Алгоритм является универсальным методом, которым можно решить любую задачу линейного программирования.

Алгоритм решения задачи ЛП симплекс-методом:

Пусть в задаче есть m ограничений, а целевая функция заивисит от n основных переменных. Первым делом необходимо привести все ограничения к каноническому виду — виду, в котором все условия задаются равенствами. Для этого предварительно все неравенства с умножаются на -1, для получения неравенств с .

Чтобы привести ограничения с неравенствами к каноническому виду, для каждого ограничения вводят переменную, называемую дополнительной с коэффициентом 1. В ответе эти переменные учитываться не будут, однако сильно упростят начальные вычисления. При этом дополнительные переменные являются базисными, а потому могут быть использованы для формирования начального опорного решения.

После того как задача приведена к каноническому виду, необходимо найти начальный базис для формирования первого опорного решения. Если в процессе приведения были добавлены дополнительные переменные, то они становятся базисными.

Иначе необходимо выделить среди коэффициентов ограничений столбец, который участвует в формировании единичной матрицы в заданной строке (например, если требуется определить вторую базисную переменную, то необходимо искать столбец, в котором второе число равно 1, а остальные равны нулю). Если такой столбец найден, то переменная, соответствующая этому столбцу, становится базисной.

В противном случае можно поискать столбец, в котором все значения кроме числа в заданной строке равны нулю, и, если он будет найден, то разделить все значения строки на число, стоящее на пересечении этих строки и столбца, тем самым образовав столбец, участвующий в формировании единичной матрицы.

Если такой столбец отсутствует, то для формирования базиса необходимо применить исключение Гаусса для первого ненулевого столбца, который ещё не является

базисным. Для этого вся строка делится на элемент в найденном столбце, а из остальных строк вычитается полученная строка, разделённая на значение, стоящее в этом же столбце. После этой операции все значения вне данной строки будут обнулены, и столбец можно будет считать базисным.

После приведения к каноническому виду или после алгебраических преобразований при формировании базиса некоторые из свободных коэффициентов (b_i) могли стать отрицательными, что не позволяет перейти к дальнейшим вычислениям.

Если строка с максимальным по модулю b_i не содержит отрицательных элементов, то такая задача не имеет решений и на этом алгоритм заканчивает свою работу. В противном случае все b_i положительны и алгоритм переходит к следующему этапу — расчёту дельт.

Расчет дельт. Чтобы вычислить дельту по заданной i-ой переменной, нужно перемножить коэффициенты условий в i-ом столбце на коэффициенты целевой функции при соответствующих базисных переменных, сложить эти произведения и вычесть из полученной суммы коэффициент целевой функции столбца i.

После того как дельты рассчитаны, необходимо проверить оптимальность текущего плана. Критерий оптимальности формулируется следующим образом:

- При максимизации функции: текущее решение считается оптимальным, если в таблице отсутствуют отрицательные дельты.
- При минимизации функции: текущее решение считается оптимальным, если в таблице отсутствуют положительные дельты.

2.2 Сведение матричной игры к задаче ЛП

Алгоритм поиска решения матричной антагонистической игры, заданной платежной матрицей, имеющей размерность m×n при больших значениях m и n, сводится к алгоритму симплекс-метода решения пары взаимодвойственных задач линейного программирования.

Пусть антагонистическая игра задана платёжной матрицей A, имеющей размерность m×n. Необходимо найти решение игры, т.е. определить оптимальные смешанные стратегии первого и второго игроков.

Задача первого игрока максимизация выигрыша V.

Целевая функция принимает вид $F = y_1 + y_2 + y_3 + ... + y_m = 1/V \rightarrow min$

Функциональные ограничения $A * y \geq 1$

Общие(прямые) ограничения $y_i \geq 0 \ \forall \ 1 \geq i \geq m$

Задачи обоих игроков образуют пару симметричных взаимодвойственных задач линейного программирования, и, поэтому нет необходимости решать обе эти задачи, т.к. найдя решение одной из них, можно найти и решение другой.

3 Инструкция по запуску

1. Установка [pip](https://pip.pypa.io/en/stable/installing/) если его нет:

   - Linux или MacOS:
     sh
     curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
     python get-pip.py
   - Windows:
     Скачать [get-pip.py](https://bootstrap.pypa.io/get-pip.py) в папку на вашем компьютере.
     Открыть окно командной строки и перейдите в папку, содержащую ‘get-pip.py‘.
     Затем запустить ‘python get-pip.py‘.

2. Скачать пакет и установить его
   sh
   cd superrnash-package
   pip install

3. Использование пакета
   sh
   python
   »> import ournash
   »> ournash.main()
   2
   5 0
   6 0

   <img src=’https://sun9-26.userapi.com/c857028/v857028320/349f5/RLZIvT0v63w.jpg’ width=’600’ alt=’npm start’>

   <img src=’https://sun9-43.userapi.com/c857028/v857028320/349fd/xeNDQbxPAq0.jpg’ width=’600’ alt=’npm start’>

</p>

sh
Game value is : 0.0
optimal strategy for 1st player : [1.0, 0.0]
optimal strategy for 2nd player : [0.0, 1.0]
»>

4. Запуск тестов
   sh
   nosetests nose.py

4 Вклад каждого участника

- Александр Борисов написал unit-тесты для функции nash_equilibrium(a) с помощью утилиты nose.
- Иван Наумов оформил решение в виде пакета ourrnash, работал с github.
- Охим Елена реализовала функцию visualization(p), которая иллюстрирует работу функции nash_equilibrium(a) путем решения нескольких игр и визуализации спектров оптимальных стратегий игроков в Jupyter и офрмила readme.
- Максим Гончаренко реализовал функцию nash_equilibrium(a), которая принимает матрицу выигрыша и возвращает значение игры и оптимальные стратегии первого и второго игроков.