МУЛЬТИПЛИКАТИВНОЕ ПРЕДСТАВЛЕНИЕ ОБРАТНОЙ МАТРИЦЫ В МОДИФИЦИРОВАННОМ СИМПЛЕКС-МЕТОДЕ*

И. В. Романовский

josephromanovsky@gmail.com

5 февраля 2011 г.

Рассматривается задача линейного программирования в канонической форме:

$$f(x) = c[N] \times x[N] \to \min_{x \in \Omega}, \tag{1}$$

где

$$\Omega = \{x[N] \mid A[M, N] \times x[N] = b[M],\ x[N] \geqslant \mathbb{O}[N]\},$$
$$M = \{1, 2, \ldots, m\}, \quad N = \{1, 2, \ldots, n\}.$$

В докладе показано, как мультипликативное представление обратной матрицы влияет на алгоритмическую сторону модифицированного симплекс-метода. Обсуждаются также вопросы компьютерной реализации метода.

Введение

Как известно, в модифицированном симплекс-методе большое внимание уделяется способу представления обратной базисной матрицы $B[N', M]$ и среди таких представлений важное место занимает мультипликативное представление, в котором эта обратная матрица записывается в факторизованном виде

*Семинар по дискретному гармоническому анализу и геометрическому моделированию «DHA & CAGD»: `http://www.dha.spb.ru/`

— как произведение матриц-мультипликаторов[1]

$$B[N'_k, M] = D_k[N'_k, N'_{k-1}] \times D_{k-1}[N'_{k-1}, N'_{k-2}] \times \cdots \times D_1[N'_1, M],$$

а каждый мультипликатор отличается от единичной матрицы всего одним столбцом. Это представление было использовано ещё Г. Зойтендейком [1], а затем подробно описано Л. Лэсдоном [2], который сам ссылается на статью Л. Ларсена [3]. Уже в 1962 появился краткий отчет Д. Смита и У. Орчард-Хейса об экспериментах с программной реализацией метода [4]. В моей книге 1977 г. [5] это представление, конечно, тоже излагается, хотя и вкратце[2].

Меня самого перейти на мультипликативное представление матрицы вынудили обстоятельства: я передал Н. Я. Краснеру в Воронежский университет свою программу модифицированного симплекс-метода, написанную на Алголе-60, и Краснер пожаловался, что программа медленно работает на задачах с двумя-тремя десятками ограничений. Тогда я переделал представление обратной матрицы на мультипликативное, и проблема снялась. С тех пор в основе наших программ лежит именно это представление.

Модуль обратной матрицы

Сейчас уже принято разрабатывать программы в объектно-ориентированном стиле и представлять все действия, связанные с обратной матрицей, как функции некоторого класса (по-старому говоря, модуля).

Таких действий (экспортируемых, т. е. вызываемых из программного окружения) совсем немного

Создание начальной матрицы. Имеется в виду обнуление списка мультипликаторов.

Изменение столбца. Замена одного из столбцов базисной матрицы требует добавления мультипликатора, изменяющего обратную матрицу. Этот мультипликатор добавляется в конец списка.

Умножение матрицы на вектор. Действие требуется для решения прямой системы $A[M, N'] \times x[N'] = b[M]$, выполняемого умножением вектора $b[M]$ слева на обратную матрицу $B[N', M]$.

[1] Мы выписываем здесь индексные множества мультипликаторов, так как с ними формула понятнее. Вычисления ведутся так, как будто все эти множества заменены множеством M.

[2] Популярность самого названия была так велика, что в стандартном учебнике для нашего экономического факультета им назван обычный метод обратной матрицы.

Умножение вектора на матрицу. Действие требуется для решения двойственной системы $u[N'] \times A[M, N'] = c[N']$, выполняемого умножением вектора $c[N']$ справа на обратную матрицу $B[N', M]$.

Иногда набор операций над данными называют **кластером операций**.

В случае мультипликативного представления обратной матрицы обычно предусматривается ещё недоступное снаружи действие

Повторное обращение базисной матрицы. Это действие предназначено для сокращения информации о текущей базисной матрице.

Умножение матрицы на вектор

Умножение вектора на обратную матрицу, составленную из r мультипликаторов, составляется из последовательных умножений на отдельные мультипликаторы $D_1, D_2, \ldots, D_r$. При этом удается выполнять «умножение на месте»: результат записывается на то место, где находился исходный вектор.

Мы можем сейчас считать, что все эти матрицы имеют одни и те же множества индексов и строк и столбцов $M = 1 : m$.

Напомним, что каждый мультипликатор D_k — это матрица, в которой один столбец, скажем, j_k — это произвольный столбец, обозначим его через $d_k[M]$, а остальные — столбцы единичной матрицы. Поэтому в произведении $D_k[M, M] \times x[M] = x'[M]$ получаем

$$\begin{aligned} x'[j_k] &:= d_k[j_k] \cdot x[j_k], \\ x'[j] &:= x'[j] + d_k[j] \cdot x[j_k], \quad j \neq j_k. \end{aligned}$$

Видно, что элементы вектора $x[M]$, которым соответствуют нулевые элементы вектора $d_k[M]$, не изменяются, а изменяемые элементы будут вычисляться правильно, если значение $x[j_k]$ появится при просмотре элементов столбца единообразно — в начале или в конце просмотра. Если оно появится в начале, то его нужно выписать отдельно для дальнейшего использования.

Возможна, таким образом, такая схема умножения на мультипликатор с сохранением результата на том же месте:

1. Прочесть пару $(d_k[j_k], j_k)$. Положить[3] $x_{\mathrm{pivot}} := x[j_k]$. Положить $x[j_k] :=$ $:= d_k[j_k] \cdot x[j_k]$.

2. Для каждой следующей пары $(d_k[j], j)$ положить $x[j] := x[j] + d_k[j] \cdot x_{\mathrm{pivot}}$. Здесь очень красиво это действие можно записать с помощью операции $+ :=$

[3] Отмечу, что это не определение, а знак присваивания. Нужное нам значение помещается в безопасное место.

(читается «плюс-присвоить»; такая операция есть в ряде языков программирования и в наборе машинных команд).

Легко видеть, что если нужно выполнять умножение на последовательность матриц, то можно организовать «поточный» алгоритм»: будем записывать главный элемент в начале каждого мультипликатора, причём значение j_k будем записывать с минусом. Предположим, что у нас есть операция `GetNextPair(a,k)`, вырабатывающая логическое значение «следующая пара существует» и при благоприятном исходе записывающая в a и k вещественную и целочисленную компоненты пары. Алгоритм умножения будет выглядеть примерно так:

```
<записать в x[1:m] правые части системы>
<приготовиться к прямому просмотру мультипликаторов>
while GetNextPair(a,k) do
  if k < 0 then begin
    k := -k; xPivot := x[k]; x[k] := a
  end else
    x[k] +:= a*xPivot;
<записать из x[1:m] решение системы>
```

Умножение вектора на матрицу

Вычисление вектора $u[M] = c[N'] \times B[N', M]$, являющегося решением двойственной системы, также состоит из последовательных умножений на мультипликаторы, но эти умножения идут в обратном порядке — от D_k к D_1. В каждом таком умножении вычисление идёт по формулам

$$\begin{aligned} u'[k] &= \sum_i u[i] \cdot d[i], \\ u'[i] &= u[i] \quad \text{при } i \neq k. \end{aligned} \tag{$*$}$$

Стало быть, удобно, чтобы при просмотре элементов столбца $d_k[M]$ элемент d_k просматривался последним. Тогда можно просто вычислять сумму из $(*)$ и при появлении k-го элемента мультипликатора записать полученную сумму в нужную позицию вектора.

При описанном выше обратном просмотре последовательности пар обеспечивает нам правильный порядок и в перечислении мультипликаторов и при просмотре пар конкретного мультипликатора. Нужно только потребовать, чтобы такой обратный порядок просмотра был обеспечен. Будем считать, что у нас есть операция обратного просмотра `GetPrevPair(a,k)`, вырабатывающая логическое значение «предыдущая пара существует» и при благоприятном исходе записывающая в a и k вещественную и целочисленную компоненты пары. Алгоритм умножения будет выглядеть примерно так:

```
<записать в x[1:m] правые части системы>
<приготовиться к обратному просмотру мультипликаторов>
sum := 0;
while GetPrevPair(a,k) do
  if k < 0 then begin
    k := -k; x[k] := a*x[k] + sum;
    sum := 0
  end else
    sum +:= a*x[k];
<записать из x[1:m] решение системы>
```

Запись нового мультипликатора

Для записи нового мультипликатора нужно использовать операцию приписывания пары в конец набора. Назовём эту операцию `SavePair(a,r)`. После формирования массива d при известном индексе столбца r сначала записывается пара $(d[r], -r)$, а затем, в любом порядке, все ненулевые элементы столбца. Точнее, все элементы, абсолютная величина которых превосходит некоторую заранее выбранную величину `epsMultiplEntry`.

Повторное обращение базисной матрицы

При каждой итерации (модифицированного) симплекс-метода к последовательности мультипликаторов добавляется еще один, и умножения, трудоёмкость которых линейно зависит от длины последовательности, растёт. Считается целесообразным выполнять время от времени перевычисление (reinversion — переобращение) обратной матрицы, получая её из единичной по кратчайшему пути (длина его, если считать число мультипликаторов, равна числу неортов в текущем базисном множестве).

При обращении можно существенно сэкономить место за счет рационального выбора последовательности включения в формируемое заново базисное множество тех элементов, которые в нём должны быть. Я не буду рассказывать про это (очень интересное направление), отмечу, что у нас в лаборатории исследования операций этими вопросами издавна занимался С. С. Сурин (см., например, [6]).

Хранение информации о мультипликаторах

Набор мультипликаторов, как мы видели, представляет собой последовательность структур, каждая из которых составлена из двух элементов, и можно ожидать, что размер этой последовательности неудобно велик, но точно неизвестен. Он заслуживает серьёзного обсуждения.

Хранение информации о мультипликаторах должно удовлетворять следующим условиям.

- Информация состоит из последовательности пар (d, k), где поле d предназначено для хранения одного числа в формате с плавающей точкой (оно выравнивается на 8), а поле k предназначено для целочисленного индекса, и здесь достаточно короткого целого, выровненного на 2.

- Хранение должно обеспечивать эффективное выполнение следующих операций:

  – создание пустой последовательности или опустошение имеющейся,

  – добавление очередного элемента в конец последовательности,

  – просмотр последовательности от начала к концу,

  – просмотр последовательности от конца к началу.

Отметим некоторое неудобство хранения структуры, состоящей из пары полей неодинаковой длины, имеющих разную «степень выравнивания» — нам придется мириться с потерями либо памяти, либо эффективности.

Мы предлагаем следующее.

В современных компьютерах с огромными ресурсами оперативной памяти появилась конструкция `MemoryStream` — поток данных, размещаемый в оперативной памяти (среди языков, поддерживающих этот тип, назову `Delphi` и `C#`). Этот тип данных не требует особых действий по захвату памяти, она добавляется по мере надобности. Поток может иметь блочную структуру — состоять из элементов предписанного размера. Чтение потока может осуществляться от начала к концу и от конца к началу. Важно, что в отличие от массива изменение размера в потоке не требует от программы никаких специальных действий.

В соответствии со сказанным можно создать объект типа `MemoryStream`, фактически состоящий из блоков фиксированного размера. Например, блок может состоять из 1024 чисел формата с плавающей точкой, 1024 коротких целых чисел и возможно учетной информации — номера блока, его текущего заполнения и диапазона номеров пар. Чтение очередной пары будет состоять из возможной загрузки требуемого блока в пользовательский буфер, независимого чтения компонент пары и сдвига текущего индекса.

ЛИТЕРАТУРА

1. Zoutendijk G. *Methods of feasible directions.* Elsevier Publishing Co., 1960 (русский пер.: Г. Зойтендейк, Методы возможных направлений. М.: ИЛ, 1963).

2. Lasdon L. S. *Optimization theory for large systems.* MacMillan Co., 1970 (русский пер.: Л. Лэсдон, Оптимизация больших систем. М.: Наука, 1975).

3. Larsen L. J. *A modified inversion procedure for product form of the inverse linear programming codes* // Comm. of ACM. V. 5. 1962. P. 382–383.

4. Smith D. E., Orchard-Hays W. *Computational efficiency in product form LP codes* // In: R. L. Graves, Ph. Wolfe, eds. Recent Advances in Mathematical Programming. McGraw-Hill Book Co., 1963. P. 211–218.

5. Романовский И. В. *Алгоритмы решения экстремальных задач.* М.: Наука, 1977.

6. Брэгман Л. М., Прыгичев А. Н., Сурин С. С. *Повышение эффективности мультипликативного алгоритма метода последовательного улучшения плана* // В сб.: И. В. Романовский, ред. «Исследование операций и статистическое моделирование». Вып. 4. Изд-во ЛГУ, 1977. С. 3–49.