Абрамов С. А., Бордаченкова Е. А.

Функции Ляпунова и сложность алгоритмов

Аннотация

Рассматриваются функции, убывающие в ходе выполнения алгоритма. Специфика множества значений и характер убывания рассматриваемой функции позволяет в ряде случаев делать выводы о завершимости алгоритма и о границах числа его шагов. Такие функции выступают как аналоги функций Ляпунова, используемых в исследовании решений ОДУ. Заметка носит характер эссе.

1. Убывающие функции на множестве состояний

Типичный итерационный алгоритм содержит цикл, в котором набор начальных величин v преобразуется в набор v', затем v' преобразуется в v'' и т.д. Пусть функция L, определенная на наборах величин или на значениях размеров таких наборов, принимает неотрицательные целые значения и при этом убывает по ходу выполнения алгоритма: $L(v) > L(v') > L(v'') > \dots$ или $L(||v||) > L(||v'||) > L(||v''||) > \dots$ (в последней цепочке неравенств обозначение $||v||$ используется для *размера* рассматриваемого набора величин; что именно считать размером зависит от соглашения, которое принимается с учетом характера решаемой алгоритмом задачи).

Из того, что функция L принимает целые неотрицательные значения, заключаем, что выполнение алгоритма закончится не позднее, чем после $L(v)$, или соответственно после $L(||v||)$, шагов (итераций), где v обозначает набор исходных данных, или, как говорят, *вход* алгоритма. Функцию L можно назвать *функцией Ляпунова* цикла, имея в виду аналогию с классической функцией Ляпунова, убывающей вдоль решения дифференциального уравнения; аналогия отмечена в [7, разд. 3.1.3].

С другой стороны, L может выступать в качестве характеристики класса алгоритмов решения задачи или самой задачи. Если функция L подобрана так, что, например, один шаг любого алгоритма из некоторого класса уменьшает значение $L(||v||)$ не болеее, чем на известное положительное число h, при том, что неотрицательность значения L служит условием продолжения работы любого из таких алгоритмов, то число шагов для каждого алгоритма из рассматриваемого класса будет не меньше, чем $\lceil L(||v||)/h \rceil$, т.е. мы получаем зависящую от $||v||$ *нижнюю границу* числа шагов для любого алгоритма рассматриваемого класса и произвольного входа v фиксированного размера (здесь и далее $\lceil x \rceil$ — наименьшее целое, большее или равное вещественному числу x, соответственно $\lfloor x \rfloor$ — наибольшее целое, не превосходящее x).

Проиллюстрируем сказанное простыми примерами.

Пример 1. Бинарный поиск места элемента y в упорядоченном массиве $x_1 < \cdots < x_n$. Априори может осуществляться любая из $n+1$ возможностей:

$$y \le x_1, \;\; x_1 < y \le x_2, \;\; \dots\;, \;\; x_{n-1} < y \le x_n, \;\; x_n < y,$$

которым присваиваются номера $1, 2, \dots, n+1$. Требуется найти номер осуществившейся для данного y возможности (см. [1, пример 9.2]). При бинарном поиске от поиска места элемента в упорядоченном сегменте длины k исходного массива (первоначально $k = n$) мы, выполнив одно сравнение, переходим к поиску места элемента в сегменте длины $\lfloor \frac{k}{2} \rfloor$ или $\lfloor \frac{k}{2} \rfloor - 1$. Возьмем $L(k) = \lambda_2(k)$, где значение $\lambda_2(k)$ равно числу цифр в двоичной записи k. Мы видим, что с каждым шагом алгоритма функция $L(k)$ убывает по крайней мере на единицу.

При $k = 1$ потребуется одно-единственное сравнение. Таким образом, эта функция $L(k) = \lambda_2(k)$ может выступить в роли функции Ляпунова. Получаем, что общее число шагов (сравнений) при бинарном поиске не превосходит $\lambda_2(n) = \lfloor \log_2 n \rfloor + 1$.

Пример 2. Каждый шаг алгоритма Евклида вычисления НОД(a_0, a_1) для данных целых $a_0 \ge a_1 \ge 0$ имеет дело с парой (a_{i-1}, a_i) неотрицательных целых чисел и при условии $a_i \ne 0$ перерабатывает ее в (a_i, a_{i+1}), где a_{i+1} равно остатку от деления a_{i-1} на a_i: $a_{i-1} = q_i a_i + a_{i+1}$. В данном случае мы можем определить функцию Ляпунова как $L(a_{i-1}, a_i) = a_i$.

Так как остаток меньше делителя, эта функция убывает, при том, что ее значениями являются неотрицательные целые числа. Можно заключить, что потребуется не более a_1 шагов (делений с остатком) для получения нулевого остатка.

В разд. 2 будут рассмотрены границы числа шагов алгоритма Евклида как такие функции от a_1, которые имеют логарифмический рост.

Значения функции Ляпунова L не обязаны быть целочисленными, они могут быть вещественными. Важно, чтобы с каждым шагом алгоритма происходило убывание функции на величину, не меньшую (для получения верхней границы числа шагов) или не большую (для нижней границы) некоторого фиксированного $h > 0$.

Пример 3. Вычисление a^n с помощью умножений. Здесь n — заданное неотрицательное целое, a можно считать вещественным числом, но можно и, например, квадратной матрицей. Для простоты, считаем a вещественным или целым числом.

На каждом шаге алгоритма вычисления a^n с помощью умножений имеется уже вычисленный набор степеней $a^{m_1}, \dots, a^{m_k}$, при этом изначально набор содержит один элемент a^1. Выполнив одно умножение, мы не можем увеличить максимальный показатель $m = \max\{m_1, \dots, m_k\}$ более, чем вдвое. Поэтому функция $L(m) = \log_2 n - \log_2 m$ в результате одного шага (одного умножения) не может уменьшиться более, чем на 1. Значение этой функции на исходном наборе равно $\log_2 n$, а на итоговом наборе она равна 0. Отсюда следует, что любой алгоритм рассматриваемого класса затрачивает не менее $\lceil \log_2 n \rceil$ умножений. Невозможен, например, алгоритм, вычисляющий a^n за число умножений $O(\log \log n)$, $n \to \infty$. Известный бинарный алгоритм ([1, пример 4.2]) затрачивает число умножений, имющее порядок $\log_2 n$, и поэтому является асимптотически оптимальным.

2. Укрупнение шагов

Для получения оценки числа шагов иногда оказывается удобным укрупнить шаги, считая k последовательных изначальных шагов

($k \geq 2$), одним шагом, и, уже исходя из этого разбиения на шаги, подбирать функцию Ляпунова; в полученную оценку вносится множитель k. Это может привести к более точной оценке в сравнении с полученной при изначальном разбиении на шаги.

Пример 4. Возвращаясь к алгоритму Евклида (пример 2), замечаем, что два последовательных шага уменьшают меньшее число пары по крайней мере вдвое: $a_i = q_{i+1}a_{i+1} + a_{i+2} > a_{i+2} + a_{i+2} = 2a_{i+2}$. Итак, целое число a_{i+2} по крайней мере вдвое меньше целого числа a_i. Это означает, что цифр в двоичной записи a_{i+2} меньше, чем в записи a_i: $\lambda_2(a_{i+2}) < \lambda_2(a_i)$. В качестве функции Ляпунова может выступить функция L на парах $(a_0, a_1), (a_2, a_3), \ldots$, определенная как $L(a_{2m}, a_{2m+1}) = \lambda_2(a_{2m+1})$, ее убывание установлено. Число сдвоенных шагов не превосходит $L(a_1) = \lambda_2(a_1)$. Таким образом, число шагов (делений с остатком) алгоритма Евклида не превосходит $2\lambda_2(a_1) = 2\lfloor \log_2 a_1 \rfloor + 2$, это значительно более точная оценка, чем полученная в примере 2. Из нее получаем также асимптотическую оценку $O(\log a_1)$.

Нелишне отметить, что для алгоритма Евклида оценки, подобные $2\lambda_2(a_1)$, можно находить не только для двоичной, но для любой системы счисления с произвольным целым основанием $q \geq 2$. Каждая такая верхняя оценка (граница сверху) для числа шагов алгоритма Евклида имеет вид $c_q\lambda_q(a_1)$, где $\lambda_q(a_1)$ — число q-ичных цифр в записи a_1 и c_q — соответствующий целый коэффициент. Как мы покажем, верхней оценкой будет и $5\lambda_{10}(a_1)$, т.е. число шагов алгоритма Евклида, применяемого к целым a_0, a_1, где $a_0 \geq a_1$, не превосходит упятеренного числа десятичных цифр в записи a_1.

Универсальный метод получения такого рода оценок числа шагов алгоритма Евклида опирается на известную теорему Ламе ([5, гл. V, §4], [6, разд. 4.5.3]): если алгоритм Евклида требует m делений, включая последнее деление, дающее нулевой остаток $a_{m+1} = 0$, то $a_1 \geq F_{m+1}$ (здесь и далее обозначение F_n привлекается для числа Фибоначчи с номером n: $F_0 = 0, F_1 = 1, \ldots$). Из этого выводится, что $m \leq (\lfloor \log_{(1+\sqrt{5})/2} q \rfloor + 1)\lambda_q(a_1)$. По этой последней формуле получаем, в частности, $c_2 = \lfloor \log_{(1+\sqrt{5})/2} 2 \rfloor + 1 = 2$, $c_{10} = \lfloor \log_{(1+\sqrt{5})/2} 10 \rfloor + 1 = 5$.

Двоичная оценка *допускает локализацию* в следующем смысле: для $i = 1, ..., m-2$ выполнено $a_i \geq 2a_{i+2}$. Утверждение же $a_i \geq 10a_{i+5}$, вообще говоря, неверно — пример: $a_0 = 128, a_1 = 79, i = 1$. Это говорит о том, что десятичная оценка не может быть получена укрупнением шагов, при котором первоначальные шаги собираются в пятерки.

С другой стороны, двоичная оценка *допускает улучшение*: на самом деле для общего числа m шагов выполнено строгое неравенство $m < 2\lambda_2(a_1)$, но пример $a_0 = 13, a_1 = 8$ показывает, что десятичная оценка такого улучшения не допускает. Обнаруживается (см. [2, теоремы 2, 3]), что

1) оценка обсуждаемого вида допускает локазизацию, если и только если она допускает улучшение в указанном смысле;

2) верхняя оценка (граница сверху) для числа шагов алгоритма вида $m \leq c_q\lambda_q(a_1)$ допускает локализацию если и только если $((1+\sqrt{5})/2)^{t-1} < q \leq F_{t+1}$ для некоторого натурального t.

Список $2, 3, 5, 7, 8$ состоит из всех $q \leq 10$, для которых оценка $m \leq c_q\lambda_q(a_1)$ допускает локализацию и улучшение.

3. Учет предыстории

В предыдущих примерах значение функции Ляпунова L определялось текущим состоянием величин, с которыми оперирует алгоритм. Но в некоторых случаях бывает полезен учет предыстории этого состояния.

Пример 5. В [9] исследуется задача нахождения с помощью сравнений в данном массиве $x_1, \ldots, x_n$ двух элементов — наибольшего и наименьшего. Элементы массива считаются попарно различными. Основной вопрос связан с нахождением точной нижней границы необходимого числа сравнений и видом оптимального алгоритма.

Каждый этап какого-либо алгоритма решения указанной задачи поиска может быть охарактеризован четверкой (A, B, C, D) подмножеств множества исходных элементов $\{x_1, \ldots, x_n\}$: A состоит из всех тех элементов, которые вообще не сравнивались; B состоит из всех тех элементов, которые участвовали в некоторых сравнениях и всегда оказывались бо́льшими; C состоит из всех тех элементов, которые участвовали в некоторых сравнениях и всегда оказывались меньшими; D состоит из всех тех элементов, которые участвовали в некоторых сравнениях, иногда оказываясь бóльшими, а иногда — меньшими.

Пусть a, b, c, d — количества элементов множеств A, B, C, D соответственно. Исходная ситуация характеризуется равенствами $a = n$, $b = c = d = 0$. После завершения алгоритма должны выполняться равенства $a = 0, \quad b = c = 1, \quad d = n - 2$. После первого сравнения на протяжении всего выполнения алгоритма постоянно будут иметь место неравенства $b \geq 1$, $c \geq 1$.

Все сравнения, совершаемые при выполнении алгоритма, можно разбить на типы, обозначаемые AA, AB, AC, AD, BB, BC, BD, CC, CD, DD, например: сравнение принадлежит типу AB, если один из сравниваемых элементов берется из A, другой — из B, и т.д. Исходя из этого, можно описать все возможные изменения четверки (a, b, c, d) под действием сравнений разных типов. Здесь в качестве функции Ляпунова рассмотрим $L(a, b, c, d) = \frac{3}{2}a + b + c - 2$. Для начальной и конечной стадий имеем $L(n, 0, 0, 0) = \frac{3}{2}n - 2$ и $L(0, 1, 1, n-2) = 0$ соответственно.

Каждое сравнение типов AA, BB, CC понижает значение L на 1, т.е. дает $\Delta L = -1$. Сравнения, относящиеся к типам AB, AC, BC, могут приводить к $\Delta L < -1$, но это не может быть гарантировано никаким специальным выбором элементов из соответствующих множеств четверки (A, B, C, D), даже если принимать во внимание результаты всех сравнений, в которые были вовлечены конкретные элементы этих множеств. Например, сравнение типа AB в том случае, когда выбранный элемент из A оказывается больше выбранного элемента из B, преобразует (a, b, c, d) в $(a - 1, b, c, d + 1)$, что дает $\Delta L < -1$. Но результаты предшествующих сравнений не дают оснований для отметания возможности того, что выбранный элемент из B равен $\max\{x_1, \ldots, x_n\}$ (ибо этот элемент оказался бóльшим во всех сравнениях, в которые он был вовлечен). Но тогда будет выполнено $\Delta L = -\frac{1}{2}$. Аналогичным образом дело обстоит для сравнений, принадлежащих типам AC, BC. Поэтому, рассматривая поведение алгоритма в худшем случае, надо считать, что на всех этапах $\Delta L \geq -1$. Получаем, что для достижения равенства

$L(a,b,c,d) = 0$ потребуется не менее $\lceil L(n,0,0,0)\rceil$ сравнений. Это значит, что общее число сравнений в худшем случае не меньше, чем $\lceil 3n/2\rceil - 2$.

Имеющий сложность $\lceil 3n/2\rceil - 2$ алгоритм известен — см. [9] (а также [3, прил. A2, п.3]). Из сказанного выше следует, что этот алгоритм оптимален.

Обращает на себя внимание то, что, во-первых, в этом примере определение функции Ляпунова L основывается на предыстории выполнения алгоритма, т.е. на информации о ходе его выполнения (множества A, B, C, D), во-вторых, имеется также отсылка к предыстории в доказательстве того, что $\Delta L = -1$ в худшем случае выполнено на каждом шаге. При этом сам оптимальный алгоритм экономит память и не запоминает подробности предыстории текущего состояния. Он основывается на последовательном рассмотрении пар $(x_1, x_2), (x_3, x_4), \ldots$ Последний элемент массива может остаться без пары, с ним в конце надо разбираться отдельно. В каждой паре определяется наибольший и наименьший, они сравниваются с наибольшим и наименьшим из элементов, попавших в предыдущие пары и т.д.

Упомянутый алгоритм и идея использования четверок (A, B, C, D) в доказательстве его оптимальности были предложены И. Полом в [9], но при этом убывающие функции в доказательстве Пола не привлекались, из-за чего потребовались дополнительные словесные рассуждения.

4. Сложность в среднем

В предыдущих примерах речь шла о сложности в худшем случае. Функции Ляпунова могут оказаться полезными и при рассмотрении сложности в среднем. Важную роль играет оценивание $\mathsf{E}\Delta L$— математического ожидания изменения значения функции L в результате выполнения одного шага алгоритма.

Пример 6. Продолжение примера 5. Будем считать равновероятными все возможные взаимные порядки расположения элементов в исходном массиве и будем пытаться выяснить вид нижней границы сложности в среднем для алгоритмов решения рассматриваемой задачи.

Вновь обратимся к множествам A, B, C, D, типам сравнений $AA, AB, \ldots, DD$, количествам a, b, c, d элементов множеств A, B, C, D и функции $L(a, b, c, d) = \frac{3}{2}a + b + c - 2$ (равенство $L = 0$ является необходимым и достаточным условием того, что искомые элементы найдены).

Если при четном n ограничиться сравнениями типов AA, BB, CC, то сравнений потребуется в точности $\frac{3}{2}n - 2$. Если же n нечетно, то потребуется выполнить хотя бы одно сравнения типа AB, AC или AD. Можно показать (см. [9], [1, §17]), что математическое ожидание величины, на которую изменится значение L после такого сравнения, удовлетворяет неравенству $\mathsf{E}\Delta L \geq -1 + \frac{1}{2n}$. При нечетном n, таким образом, среднее число сравнений, обеспечивающее заключительное нулевое значение функции L, будет не меньше, чем сумма $\frac{3}{2}n - 2 + (-1 + \frac{1}{2n})$ и 1 (первое выписанное слагаемое - это нижняя граница среднего числа сравнений типов AA, BB, CC, второе - равная 1 нижняя граница числа сравнений, принадлежащих типам типов AB, AC, AD). Как следствие, функция

$$f(n) = \begin{cases} \frac{3}{2}n - 2, & \text{если } n \text{ четно,} \\ \\ \frac{3}{2}n - 2 + \frac{1}{2n}, & \text{если } n \text{ нечетно,} \end{cases} \tag{1}$$

является нижней границей сложности в среднем алгоритмов одновременного выбора наибольшего и наименьшего элементов массива длины n, $n \geq 2$, с помощью сравнений (полное обоснование см. в [1, теорема 17.1, предложение 17.2]).

В [3] и [1, §17] указан алгоритм одновременного нахождения наибольшего и наименьшего элементов массива длины $n \geq 2$ с помощью сравнений, сложность в среднем которого совпадает с (1). Очевидно, этот алгоритм является оптимальным.

5. Функции Ляпунова со значениями в фундированных множествах

По существу, содержание этого раздела основано на [4, разд. 2.3].

В некоторых случаях для доказательства завершимости некоторого итерационного процесса (цикла) имеет смысл выбирать функцию Ляпунова со значениями в множествах, отличных от множества неотрицательных целых чисел. Линейно упорядоченное множество M является *фундированным*, если не существует бесконечной убывающей последовательности $c_0 > c_1 > \dots$ элементов этого множества. В роли такого множества может выступать само множество неотрицательных целых чисел и, например, множество S_k конечных последовательностей (кортежей) фиксированной длины k неотрицательных целых чисел, оснащенное лексикографическим отношением порядка: меньшим считаем тот из двух кортежей, который лексикографически предшествует другому, – скажем, $(1,1) > (0,2)$. Это множество — фундированное (см. [4, разд. 2.3]). Содержание следующего примера составляет задача для самостоятельного решения №108 из [4].

Пример 7. Имеется конечная последовательность нулей и единиц. За один шаг разрешается найти в ней группу 01 и заменить на $10\dots0$ (при этом можно написать сколько угодно нулей). Доказать, что такие шаги нельзя выполнять бсконечно много раз.

Пусть исходная последовательность содержит k единиц. Допустимые шаги не изменяет этого числа. Рассмотрим кортежи вида $(n_1, \dots, n_k)$ длины k, состоящие из порядковых номеров единиц каждой из последовательностей, возникающих на выполняемых шагах. Определим значение функции Ляпунова L на конечной последовательности из нулей и единиц как соответствующий кортеж. Очевидно, что значения этой функции, рассматриваемыее шаг за шагом, образуют убывающую (в смысле лексикографического порядка) последовательность кортежей длины k. Фундированность S_k влечет требуемое.

6. Завершимость циклического процесса

Как видно из предыдущего, для доказательства завершимости алгоритма можно использовать функции Ляпунова со значениями в различных фундированных множествах. При этом в ряде случаев доказательство завершимости остается сложной задачей.

Пример 8. Алгоритм, заданный оператором цикла

while $n > 3$ do

if $2|n$ then $n := \frac{n}{2} + 1$ else $n := n + 1$ fi,

od

(запись $2|n$ означает, что 2 является делителем n) завершает свою работу для любого натурального n. Для доказательства достаточно рассмотреть функцию Ляпунова $L(n) = n - (-1)^n$ и убедиться в ее убывании при $n > 3$ в ходе выполнения алгоритма.

Завершимость же для любого натурального n алгоритма (вариант гипотезы, выдвинутой Л. Коллатцом в 1932 г)

while $n > 1$ do

if $2|n$ then $n := \frac{n}{2}$ else $n := 3n + 1$ fi

od

не доказана и не опровергнута по сей день, хотя на это направлялись серьезные усилия (см. [8]) и многократно предлагались решения, оказывавшиеся в дальнейшем ошибочными или неполными.

Отмеченная аналогия убывающих функций на изменяемых алгоритмом данных (состояниях) и функций Ляпунова, используемых в исследованиях решений обыкновенных дифференциальных уравнений, не является единственной в этом плане. Скажем, легко просматривается параллель между инвариантами циклов и первыми интегралами дифференциальных уравнений.

Литература

1. Абрамов С.А. Лекции о сложности алгоритмов. М.: МЦНМО, 2021.
2. Абрамов С.А. Некоторые оценки, связанные с алгоритмом Евклида. *Ж. вычисл. матем. и матем. физ.* 1979. Т. 19. № 3. С. 756–760.

3. Абрамов С.А. Исследование алгоритмов одновременного нахождения наибольшего и наименьшего элементов массива. *Ж. вычисл. матем. и матем. физ.* 1982, Т. 22. № 2, С. 424–428.

4. Верещагин Н. К., Шень А. Начала теории множеств. М.: МЦНМО, 2002.

5. Гельфонд А. О. Исчисление конечных разностей. М.: Наука, 1967.

6. Кнут Д. Искусство программирования для ЭВМ, т. 1, Основные алгоритмы. ИД «Вильямс», 1999.

7. Лавров С. С. Программирование. Математические основы, средства, теория. СПб: БХВ-Петербург, 2001.

8. The Ultimate Challenge: The $3x + 1$ Problem. J. C. Lagaris, editor. Washington: AMS, 2010.

9. Pohl I. A sorting problem and its complexity. *Comm. ACM*, 1972, V. 15, № 6, P. 462–466.

Сведения об авторах

Абрамов Сергей Александрович – д.ф-м.н., Федеральный исследовательский центр "Информатика и управление", Вычислительный центр им. А.А.Дородницына РАН, гл.н.с.,
МГУ имени М.В.Ломоносова, проф. (sergeyabramov@mail.ru)

Бордаченкова Елена Анатольевна – к.ф-м.н., МГУ имени М.В.Ломоносова, доцент (lenabord@mail.ru)