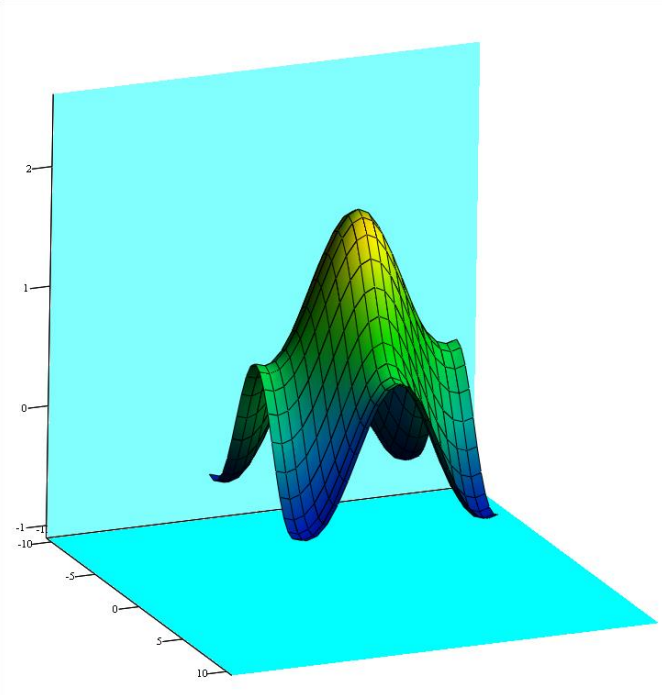
Владимирский государственный университет

В. Н. ЛОБКО

МАТЕМАТИЧЕСКИЕ МЕТОДЫ В ХИМИИ И ХИМИЧЕСКОЙ ТЕХНОЛОГИИ

ЗАДАЧИ ОПТИМИЗАЦИИ

Учебное пособие



Владимир 2025

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

В. Н. ЛОБКО

МАТЕМАТИЧЕСКИЕ МЕТОДЫ В ХИМИИ И ХИМИЧЕСКОЙ ТЕХНОЛОГИИ

ЗАДАЧИ ОПТИМИЗАЦИИ

Учебное пособие

Электронное издание

Владимир 2025

ISBN 978-5-9984-2285-0

УДК 519.61:54
ББК 22.193

Рецензенты:
Кандидат физико-математических наук, доцент
доцент кафедры физики и прикладной математики
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
А. Ю. Лексин

Кандидат технических наук
генеральный директор ООО «ФС Сервис»
Д.С. Квасов

Председатель правления
ООО «НПП ”Макромер” имени В. С. Лебедева»
А. О. Стюнина

Лобко, В. Н.
МАТЕМАТИЧЕСКИЕ МЕТОДЫ В ХИМИИ И ХИМИЧЕСКОЙ ТЕХНОЛОГИИ. Задачи оптимизации [Электронный ресурс] : учеб. пособие / В. Н. Лобко ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2025. – 207 с. – ISBN 978-5-9984-2285-0. – Электрон. дан. (3,38 Мб). – 1 электрон. опт. диск (CD-ROM). – Систем. требования: Intel от 1,3 ГГц ; Windows XP/7/8/10 ; Adobe Reader ; дисковод CD-ROM. – Загл. с титул. экрана.

Рассмотрены теоретические аспекты и практическая реализация такой важнейшей для химии области, как оптимизация химических и технологических процессов. Представлены теоретические основы решения задач оптимизации и численные методы одномерной и многомерной оптимизации с их реализацией в виде компьютерных программ на языке Lazarus, дано понятие об условной оптимизации.

Предназначено для студентов, обучающихся по направлениям подготовки 04.03.01, 04.04.01 – Химия, 18.03.01 – Химическая технология.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Ил. 74. Библиогр.: 26 назв.

ISBN 978-5-9984-2285-0

ОГЛАВЛЕНИЕ

v0 ВВЕДЕНИЕ

Химическая технология немыслима без определения оптимальных параметров проведения химических реакций и других физико-химических процессов. Математически такие задачи сводятся к представлению параметров в виде целевой функции (чаще всего – нескольких переменных) и определению её экстремумов (максимумов или минимумов). В высшей математике кардинально проработаны теоретические аспекты оптимизации и разработан так называемый классический метод решения этой задачи. К сожалению, на практике он имеет крайне ограниченное применение и в настоящее время используется лишь для анализа конкретных задач.

В этом плане исключительная роль принадлежит численным методам решения задач оптимизации, которые охватывают оптимизацию одномерную и многомерную, а также оптимизацию с ограничениями.

В пособии представлены теоретические аспекты данного вопроса, рассмотрены и описаны многие методы одномерной и многомерной оптимизации, а также дано понятие условной оптимизации, то есть с ограничениями.

По большинству методов представлены компьютерные программы на языке программирования высокого уровня Lazarus. Программы могут быть напрямую перенесены в среду программирования этого пакета (для консольного приложения) и использованы для расчётов. Обучающиеся могут не только изучить действие алгоритма, но и самостоятельно экспериментировать с этими программами, например в плане их модернизации или подведения под свою конкретную задачу.

Программы апробированы на большом количестве функций, причём одних и тех же для одномерной и (отдельно) – для многомерной оптимизации. Это позволяет сравнить результаты, полученные разными методами, между собой и сделать выводы о преимуществах и недостатках тех или иных методов. Программы легко позволяют пользователям вставлять свои собственные целевые функции и проводить их оптимизацию.

Результаты апробации представлены без форматирования чисел, то есть в самой общей форме. Это сделано потому, что заданный

априорно формат во многих случаях может привести к потере информации, если заранее не известен порядок ожидаемого результата. Общий формат представляет число в виде мантиссы с максимальным количеством значащих цифр и показателя степени также в максимальном представлении. Например, результат 5.0000262927776185E-001 представляет собой число $-5.0000262927776185 \cdot 10^{-1}$. Этот результат может быть легко преобразован в число без степени и округлён соответствующим образом.

Кроме этого, пользователи могут переносить с очень небольшими изменениями эти программы в среду программирования другого профессионального языка (DELPHI) и работать в ней.

Основы составления программ в этих пакетах рассмотрены в первой части курса «Математические методы в химии и химической технологии» [11].

Профессионализм будущих химиков во многом будет формироваться с помощью умения правильно выбрать численный метод для решения конкретной задачи и решить её на языке программирования высокого уровня или в математических пакетах, таких как MathCAD или MATLAB (в последнем случае – избежав дилетантских ошибок).

Внутренние гиперссылки позволяют перейти из оглавления к конкретному параграфу или главе («z№»). Для возврата в оглавление используются ссылки «v№».

v1 Глава 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ОПТИМИЗАЦИИ

v2 § 1. Введение. Задача оптимизации

Задачи оптимизации ((от лат. optimus – наилучшее, то есть наилучший вариант, решение, выбор) возникают не только на производстве и в других областях хозяйственной деятельности (транспорт, сельское хозяйство, маркетинг и др.), не только в фундаментальной и прикладной науке, но и в повседневной жизни и быту. Такие задачи могут возникать, когда требуется подобрать какой-то оптимальный параметр при действии на него многих, независимых друг от друга, факторов (математически это описывается как функция (собственно оптимальный параметр) многих переменных (в нашем случае – факторов)). Например, при поиске работы выпускником университета следует учитывать следующие факторы:

1. Зарплата.
2. Должность.
9. Возможность карьерного роста.
10. Сложность работы (разъезды, командировки) и условий труда.
3. Удалённость от места жительства.
4. Транспортная проблема.
6. Лояльность и адекватность начальства.
7. Атмосфера в коллективе.
8. Длительность и время отпуска.
5. Наличие соцпакетов.

и т. д.

Со своей стороны, работодатель при приёме на работу также предъявляет кандидату множество своих требований. Как правило, найти работу, полностью удовлетворяющую по всем параметрам. не удаётся. Всегда приходится чем-то в какой-то степени, жертвовать. Так или иначе, делается выбор, более-менее оптимальный в данных условиях.

Аналогичная картина может возникать во многих областях хозяйственной деятельности. На транспорте это может быть, например, выбор оптимального маршрута, оптимального графика

перевозок, наилучший географический выбор источников снабжения материалами или комплектующими. В сельском хозяйстве – например, оптимальные графики посевных или уборочных работ. На производстве – оптимальное взаимодействие всех технологических и обслуживающих цепочек, в технологии – подбор оптимальных параметров по времени изготовления продукции надлежащего качества. В научной деятельности задачи оптимизации встречаются часто и повсеместно.

Современная химическая промышленность представляет собой высокотехнологичную область деятельности, совмещающую в себе широкий спектр технологических, экономических и логистических задач. Чаще всего, в химической технологии, речь идёт о подборе оптимальных параметров проведения тех или иных процессов с целью получения наибольшего выхода продукта требуемого качества. Например, при проведении химической реакции в реакторе идеального смешения (автоклаве) действующими на процесс **факторами** являются температура, давление, время проведения реакции, начальные концентрации исходных веществ и их качество. предварительная подготовка сырья и т. д. Оптимизируемым параметром (то есть **целевой функцией**) чаще всего будет **выход продукта**.

В основе математического описания оптимизируемых объектов и процессов лежит представление о **функции многих переменных** и **нахождение её экстремумов** (минимума или максимума). Чисто математически оказывается, что методики нахождения максимумов и минимумов практически совпадают между собой. Чтобы перейти от одной задачи к другой, очень часто требуется всего лишь заменить «+» на «–» или знак «>» на знак «<», или что-то подобное. Поэтому термин **оптимизация** заменяют конкретным термином **минимизация** (функции), подразумевая при этом, что всё сказанное будет относиться и к поиску максимума.

Таким образом, **задача оптимизации** математически сводится к **задаче минимизации функции**. Чаще всего такой функцией является функция многих переменных (**многомерная оптимизация**), но встречаются задачи и с функцией одной переменной (**одномерная оптимизация**). Последние сами по себе не являются тривиальными задачами и требуют использования специальных математических

методов. Очень часто методы одномерной оптимизации являются вспомогательными или составными частями методов многомерной оптимизации.

v3 § 2. Общая постановка задачи оптимизации

Если **целевую функцию**, определённую в n-мерном пространстве, обозначить $f(x)$, где x – **допустимая точка** (вектор n-мерного пространства), принадлежащая **допустимому множеству** X решений, то общая постановка задачи оптимизации записывается следующим образом:

$$f(x) \to \underset{x \in X}{extr} \tag{1}$$

Требуется найти вектор $x^* \in X$, доставляющий экстремальное значение целевой функции на множестве X:

$$f(x^*) = \underset{x \in X}{extr}\ f(x) \tag{2}$$

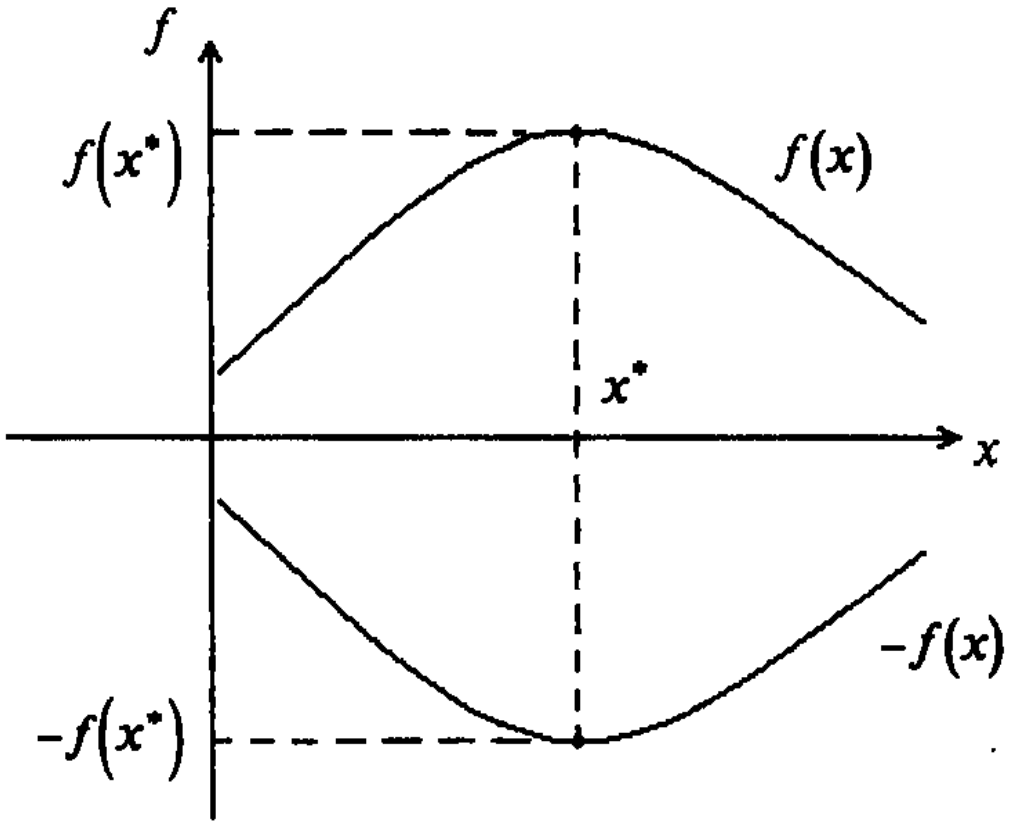


Рис. 1. Задачи минимизации и максимизации целевой функции [2, с. 6]

При конкретизации задачи экстремум заменяется на минимум или максимум, причём [2, с. 6]

$$f(x^*) = \min_{x \in X} f(x) = -\max_{x \in X} \left(-f(x)\right) \qquad (3)$$

Это проиллюстрировано на рис. 1.

Таким образом, необходимо найти значение $x^* \in X$, для которого выполняется одно из условий:

либо $f(x^*) \le f(x) \quad \forall\, x \in X$, (4)

либо $f(x^*) \ge f(x) \quad \forall\, x \in X$. (5)

Как уже отмечалось, при общем рассмотрении для конкретики формулируется именно **задача минимизации**, то есть первая часть равенства (3).

Решением задачи оптимизации является сама точка x^* и значение целевой функции в ней $f(x^*)$.

В самом общем случае функции, для которых ищутся экстремумы, могут быть функциями нескольких переменных, иметь много максимумов и минимумов на разных участках, испытывать разрывы, быть недифференцируемыми, быть неоднозначными в отдельных точках или на каких-то отрезках, иметь какие-то особые точки. Всё это усложняет задачу оптимизации. Для оптимизации функций одной переменной и многих переменных используются, вообще говоря, разные подходы и методы.

Проиллюстрируем это на примере функции одной переменной рис. 2). Лишь в простых случаях «поведение» функции известно для всей числовой оси, поэтому, прежде чем приступать к решению задачи оптимизации, функцию нужно просканировать (полностью это сделать нельзя) и исследовать, насколько это возможно. Чаще всего ограничиваются каким-то достаточно широким интервалом, на котором можно ожидать постановку и решение задачи минимизации. При этом исходят из условий общей стоящей задачи. Среди многих минимумов или максимумов можно выделить **глобальные** (по крайней мере на этом отрезке) **максимум** (точка u_3) и **минимумы** (точка u_2 и точки отрезка и $u_8 \le u \le u_9$), и **локальные максимумы** (u_1, u_7 и точки отрезка $u_5 \le u \le u_6$) и **минимумы** (точка u_4 и точки отрезка $u_5 \le u \le u_6$). Отрезок $u_5 \le u \le u_6$ можно рассматривать одновременно

как локальный минимум и как локальный максимум. Отрезки $u_5 \le u \le u_6$ и $u_8 \le u \le u_9$ являются **нестрогими максимумами и минимумами**, тогда как остальные перечисленные точки – **строгими экстремумами**. Особый статус имеют крайние точки u_0 и u_{10}. Если отрезок $[a, b]$ выбран неправильно (слева от точки a может находиться локальный минимум, а справа от точки b – локальный максимум), то оптимизация может привести к неправильному решению общей стоящей задачи, если же выбор отрезка сделан осознанно, то формально точку u_0 можно считать локальным минимумом, а точку u_{10} – локальным максимумом. Некоторые функции, например, синус или косинус, имеют много равнозначных максимумов или минимумов.

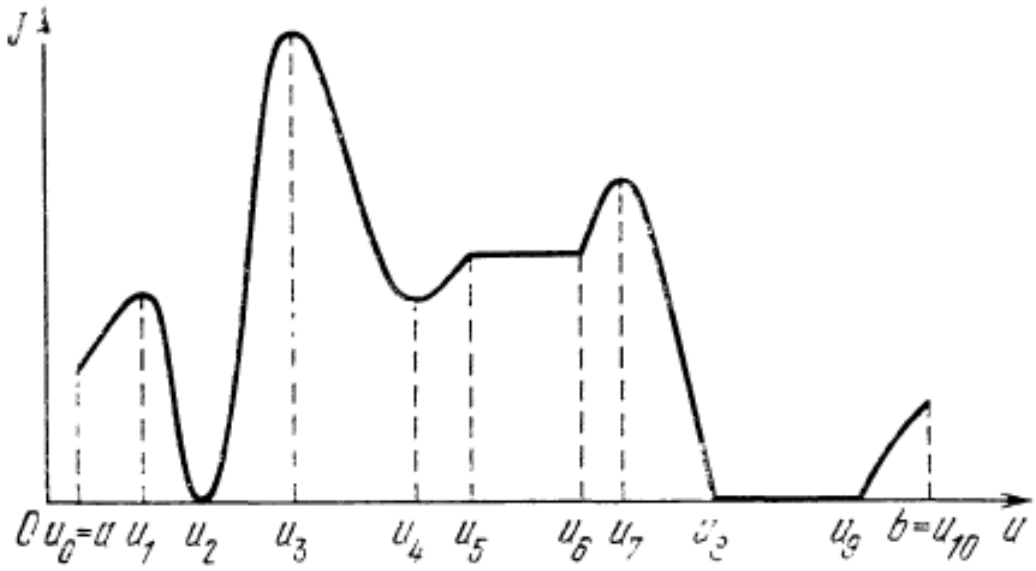


Рис. 2. Глобальные и локальные экстремумы целевой функции [1, с. 13], обозначения – см. текст

Возможны случаи, когда в крайних точках отрезка функция принимает наименьшее или наибольшее значение даже при наличии внутренних локальных минимумов или максимумов (например, рис 3). При этом может ставиться и решаться задача нахождения именно экстремального значения на отрезке. То же относится и к монотонно возрастающим или убывающим функциям (в том числе – линейным).

Причём здесь могут использоваться многие методы решения собственно экстремальных задач.

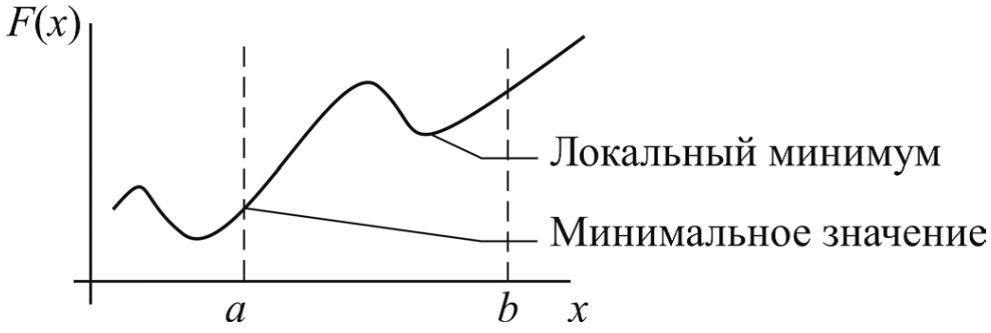


Рис. 3. Минимальное значение функции на отрезке [a, b] [4, с.9]

Всякая точка глобального экстремума функции является одновременно локальным экстремумом, поэтому для нахождения глобального экстремума нужно определить все локальные точки и сделать из них выбор.

Строгим **определением глобального минимума** может служить выражение (4), а **глобального максимума** – (5).

Локальные экстремумы определяются для **окрестностей** соответствующих точек. Решение является точкой локального минимума, если существует такая окрестность этой точки, полностью лежащая во множестве оптимизации, что значение функции в любой точке из этой окрестности меньше, чем в точке локального минимума [5, с.18].

Точка $x^* \in X$ является точкой **локального минимума** целевой функции f на X (или **локальным решением** задачи минимизации), если существует число $\varepsilon > 0$ такое, что

$$f(x^*) \le f(x) \quad \forall\, x \in X \cap U_\varepsilon(x^*), \tag{6}$$

где $U_\varepsilon(x^*) = \{x \in \mathrm{R}^n \mid \| x - x^* \| \le \varepsilon\}$ – шар радиуса $\varepsilon > 0$ с центром в x^* [3 - Сухарев с7].

(В математике

$\forall$ – означает «для всех, для каждого»;

$\in$ – знак принадлежности, в данном случае – множеству;

$\cap$ – означает множество одинаковых элементов, принадлежащих и X и $U_\varepsilon(x^*)$;

R^n – обозначение n-мерного евклидова пространства; проще говоря, это действительная (или вещественная) функция n переменных;

$\| x - x^* \|$ – норма матрицы, точнее – вектора;

$\mathrm{R}^n \mid \| x - x^* \| \le \varepsilon$ означает множество всех R^n, таких, что верно (то есть выполняется) $\| x - x^* \| \le \varepsilon$.)

Аналогично для **локального максимума**:

$$f(x^*) \ge f(x) \quad \forall\, x \in X \cap U_\varepsilon(x^*), \tag{7}$$

Если неравенство в формулах (4) – (6) – строгое при $x \ne x^*$, то x^* – точка **строгого минимума** или **максимума** (строгое решение) в глобальном или локальном смысле.

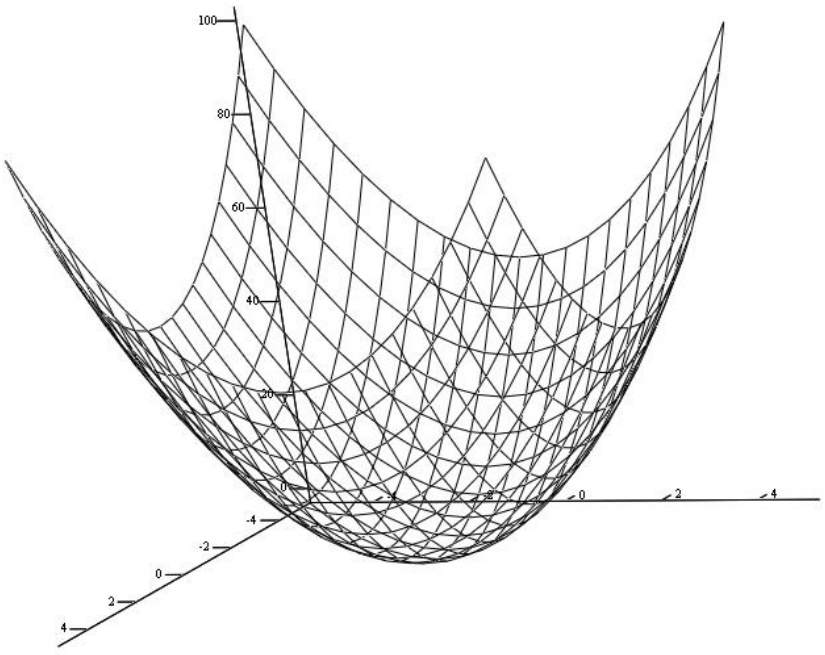


Рис. 4. График функции двух переменных $y(x1, x2) = 2 \cdot x1^2 + 2 \cdot x2^2$

Если стоит задача нахождения не экстремума, а наибольшего или наименьшего значения функции на отрезке (см., например, рис. 3), то строится последовательность $\{x_k\}$, $k = 1, 2, \ldots, x_k \in X$, такая, чтобы выполнилось одно из соотношений [6, с.24]:

$$\lim_{k\to\infty} f(x_k) = \inf_{x\in X} f(x) \quad (8)$$

$$\lim_{k\to\infty} f(x_k) = \sup_{x\in X} f(x) \quad (9)$$

Определение. **Поверхностью уровня** функции $f(x)$ называется множество точек, в которых функция принимает постоянное значение, т.е. $f(x) = const$. Если $n = 2$, поверхность уровня изображается **линией уровня** на плоскости R^2 [2, с. 7].

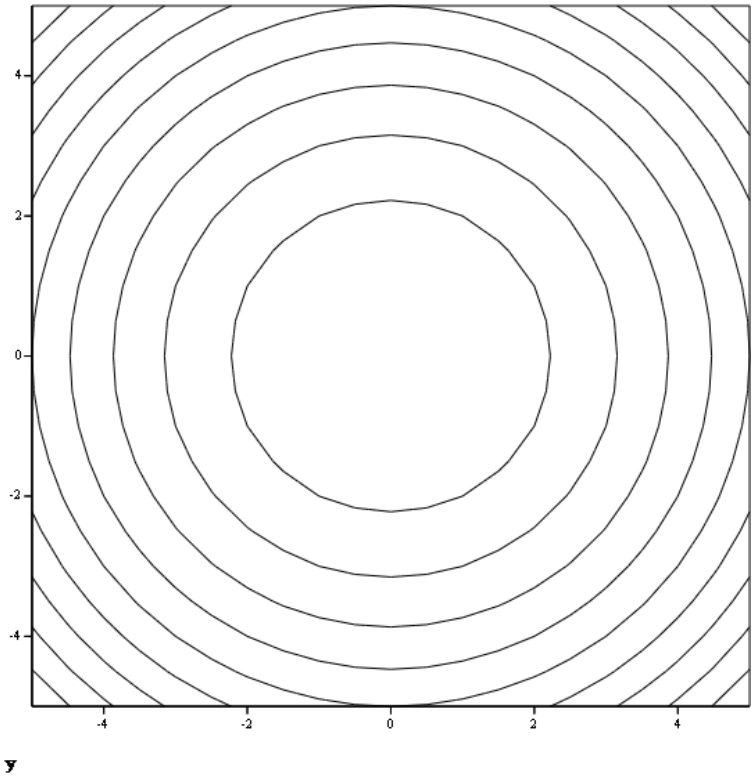


Рис. 5. Линии уровня функции двух переменных

$$y(x1, x2) = 2 \cdot x1^2 + 2 \cdot x2^2$$

Особое значение имеют поверхности уровня для функций нескольких переменных. На рис. 4 изображён график симметричной функции двух переменных $y(x1, x2) = 2 \cdot x1^2 + 2 \cdot x2^2$, а на рис. 5 – линии уровня этой функции, которые представляют собой концентрические окружности, получающиеся сечением графика рис. 4 плоскостями, параллельными плоскости осей $x1$ и $x2$.

При решении задач оптимизации необходимо, насколько это возможно, исследовать целевую функцию, в частности – на её **ограниченность**.

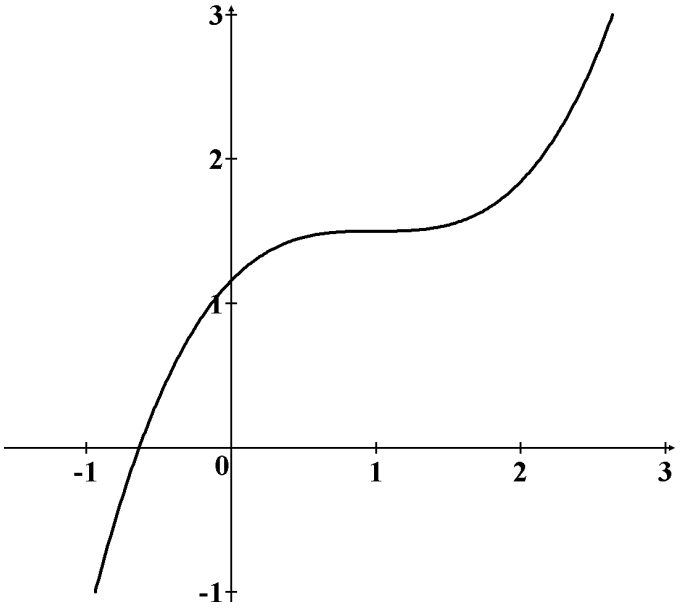


Рис. 6. Неограниченная функция $y = (0.7 \cdot (x-1))^3 + 1.5$

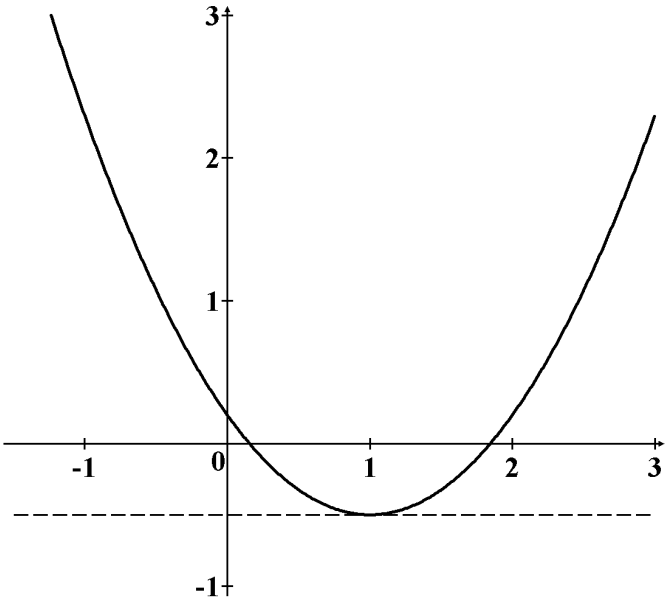


Рис. 7. Ограниченная снизу функция $y = 0.7 \cdot (x-1)^2 - 0.5$. Точка $y = -0.5$ – нижняя грань функции

На рис. 6 показан пример **неограниченной функции**, то есть функции, которая может принимать любые значения по оси ординат, если она непрерывна.

Определение. Функция $J(u)$ называется **ограниченной снизу** на множестве U, если существует такое число M, что $J(u) \geq M$ для всех $u \in U$. Функция $J(u)$ **не ограничена снизу** на U, если существует последовательность $\{u_k\} \in U$, для которой $\lim\limits_{k\to\infty} J(u_k) = -\infty$ [1, с. 10].

Пример такой функции дан на рис. 7.

Определение. Пусть функция $J(u)$ ограничена снизу на множестве U. Тогда число J_* называют **нижней гранью** $J(u)$ на U, если: 1) $J_* \leq J(u)$ при всех $u \in U$; 2) для любого сколь угодно малого числа $\varepsilon > 0$ найдется точка $u_* \in U$, для которой $J(u_\varepsilon) < J_* + \varepsilon$. Если функция $J(u)$ не ограничена снизу на U, то в качестве нижней грани $J(u)$ на U принимается $J_* = -\infty$. Нижнюю грань $J(u)$ на U обозначают через $\inf\limits_{u\in U} J(u) = J_*$ [1, с. 10].

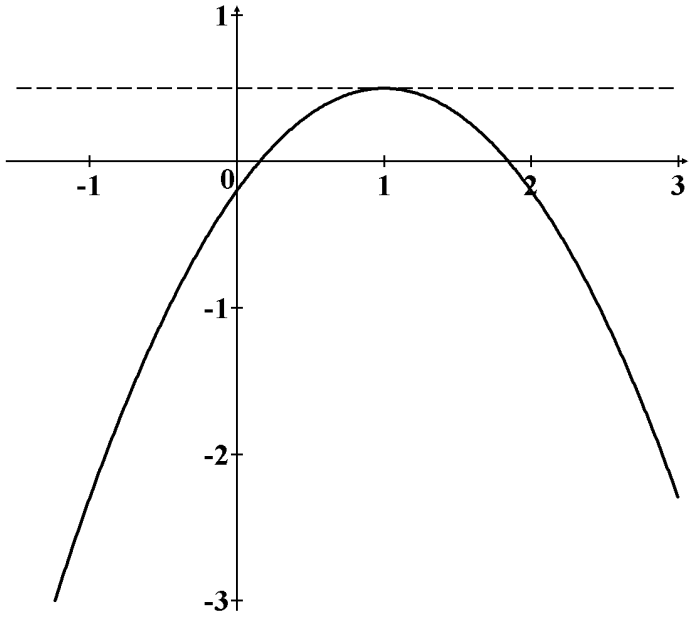


Рис. 8. Ограниченная сверху функция $y = 0.7 \cdot (x-1)^2 - 0.5$. Точка $y = 0.5$ – верхняя грань функции

Определение. Функция $J(u)$ называется **ограниченной сверху** на множестве U, если существует такое число B что $J(u) \le B$ при всех $u \in U$. Функция $J(u)$ **не ограничена сверху** на U, если существует последовательность $\{u_k\} \in U$, для которой $\lim\limits_{k\to\infty} J(u_k) = \infty$. Функцию $J(u)$ называют **ограниченной** на U, если она ограничена на U сверху и снизу [1, с. 13 – 14].

Пример ограниченной сверху функции приведён на рис. 8, а ограниченной функции – на рис. 9.

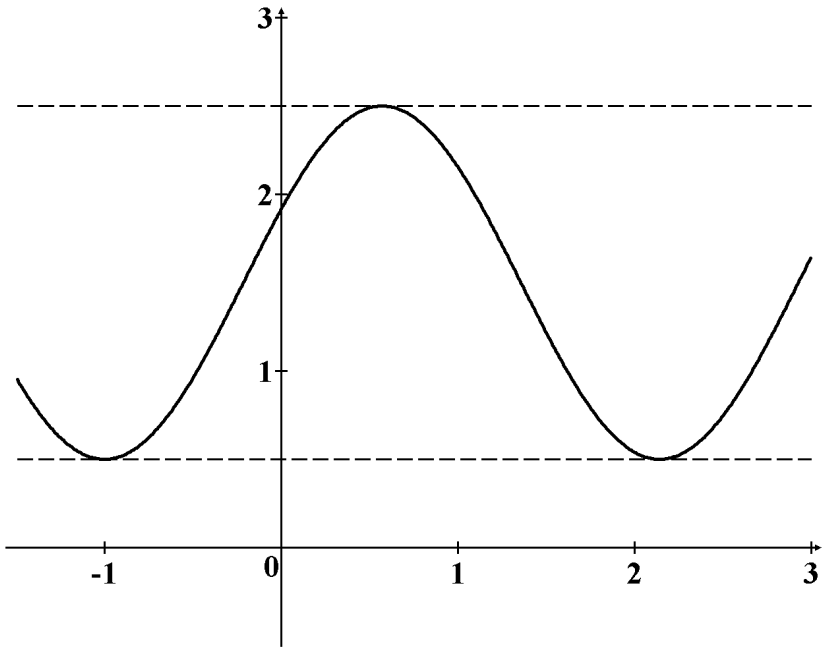


Рис. 9. Ограниченная функция $y = 2 \cdot \sin(x+1)^2 + 0.5$. Точка $y = 2.5$ – верхняя грань функции, а точка $y = 0.5$ – нижняя грань функции

Определение. Если функция $J(u)$ ограничена сверху на U, то число J^* называется **верхней гранью** $J(u)$ на U в том случае, когда: 1) $J(u) \le J^*$ для всех $u \in U$; 2) для любого числа $\varepsilon > 0$ найдется такая точка $u_\varepsilon \in U$, что $J(u_\varepsilon) > J^* - \varepsilon$. Если $J(u)$ не ограничена сверху на U, то по определению принимается $J^* = \infty$. Последовательность

$\{u_k\} \in U$, называется **максимизирующей** для $J(u)$ на U, если $\lim\limits_{k\to\infty} J(u_k) = J^*$. Если существует такая точка $u^* \in U$, что $J(u^*) = J^*$, то u^* называется **точкой максимума** $J(u)$ на U, а величина $J(u^*)$ – **наибольшим** или **максимальным значением** $J(u)$ на U. Множество точек максимума $J(u)$ на U будем обозначать через U^*, верхнюю грань – через $J^* = \sup\limits_{u\in U} J(u)$. [1, с. 13 – 14].

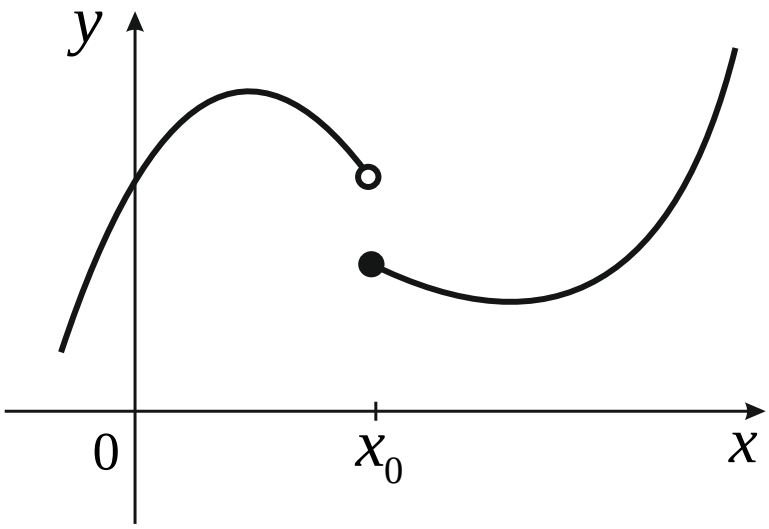


Рис. 10. Полунепрерывная снизу функция

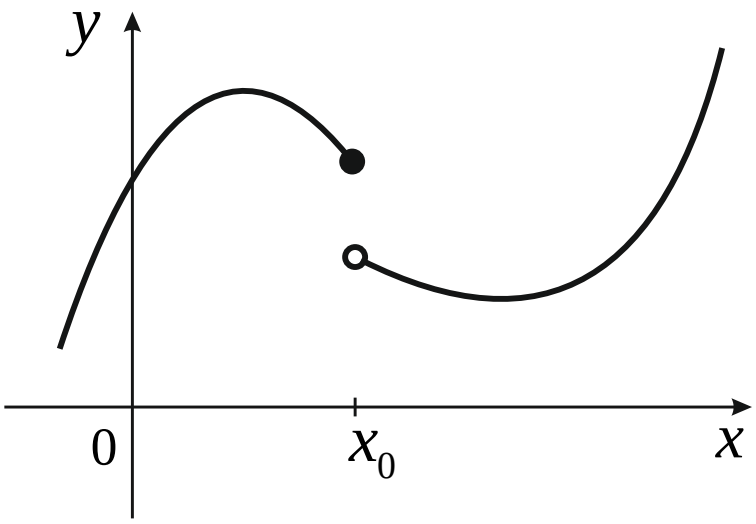


Рис.11. Полунепрерывная сверху функция

Необходимо заметить, что верхняя грань и максимизирующая последовательность всегда существуют, а максимальное значение может не существовать.

Определение. [6, с. 12] Функция $f(x): X \subseteq R^n \to R$ называется полунепрерывной снизу (рис. 10) в точке $x_0 \in X$, если для $\forall \varepsilon > 0 \ \exists \delta > 0$: для $\forall x \in X$, удовлетворяющих $\|x - x_0\| < \delta \Rightarrow f(x) > f(x_0) - \varepsilon$.

Функция $f(x)$ полунепрерывна сверху (рис. 11) в точке $x_0 \in X$, если для $\forall \varepsilon > 0 \ \exists \delta > 0$: для $\forall x \in X$, удовлетворяющих $\|x - x_0\| < \delta \Rightarrow f(x) \le f(x_0) + \varepsilon$.

Функция $f(x)$ непрерывна в точке $x_0 \in X$ тогда и только тогда, когда она в этой точке полунепрерывна и снизу, и сверху.

В этих определениях используются математические символы:

$\in$ – принадлежит;

$\forall$ – для всех, для каждого;

$\exists$ – существует;

$\Rightarrow$ – следовательно;

$\to$ – отсюда следует; «из … в …»,

$X \subseteq R^n$ – означает, что множество X или равно множеству R^n, или входит в него, как часть.

Одними из свойств полунепрерывных функций являются, в частности, следующие:

а) если $f(x)$ – полунепрерывная сверху функция, то $-f(x)$ является полунепрерывной снизу;

б) если $f(x)$ и $g(x)$ – две полунепрерывные снизу (сверху) функции, то их сумма $f(x)+g(x)$ также полунепрерывна снизу (сверху). [6, с. 13]

Обобщенная теорема Вейерштрасса (без доказательства). Полунепрерывная снизу (сверху) функция $f(x): R^n \to R$ достигает глобального минимума (максимума) на всяком компакте $X \subset R^n$. [6, с. 13]

(Компакт, или компактное пространство, в простейшем случае представляет собой отрезок на одномерном множестве).

v4 § 3. Условная и безусловная оптимизация

При решении задач минимизации в каждом конкретном случае, как правило, известна какая-то априорная информация об ожидаемом решении. Чаще всего это ожидаемое решение может принадлежать всей области $x \in X$ (то есть условия (4) и (5) выполняются буквально), но во многих случаях из априорной информации следует, что решение должно отвечать каким-то критериям, например, может быть только положительным, или только целым, или принадлежать каким-то узким областям. Тогда считается, что на задачу наложены **ограничения** и проводится **условная оптимизация** для области **допустимых решений**. Для первого же случая решается задача **без ограничений** методами **безусловной оптимизации**.

Математически это записывается так:

безусловная оптимизация:

$$X = R_n,\ x^* \in R_n; \tag{10}$$

условная оптимизация:

$$x^* \in X, \text{ но } X \neq R_n, \text{ а именно } X \subset R_n, \tag{11}$$

то есть множество допустимых решений X – собственное подмножество пространства R_n.

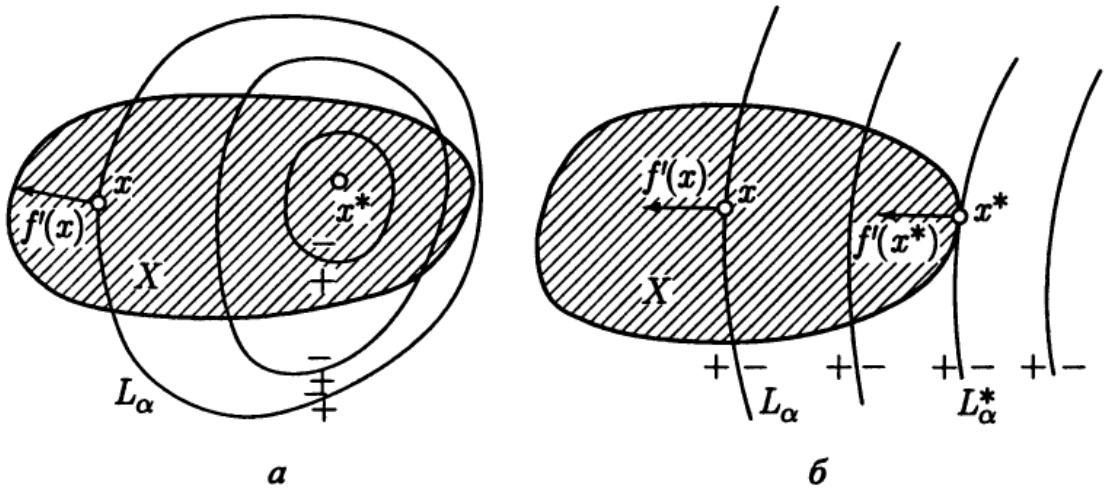


Рис. 12. Геометрическая интерпретация задач оптимизации [3, с. 12].
X – множество допустимых решений, L_α – линии уровня, минимум x^* лежит внутри множества X (а) и снаружи (б)

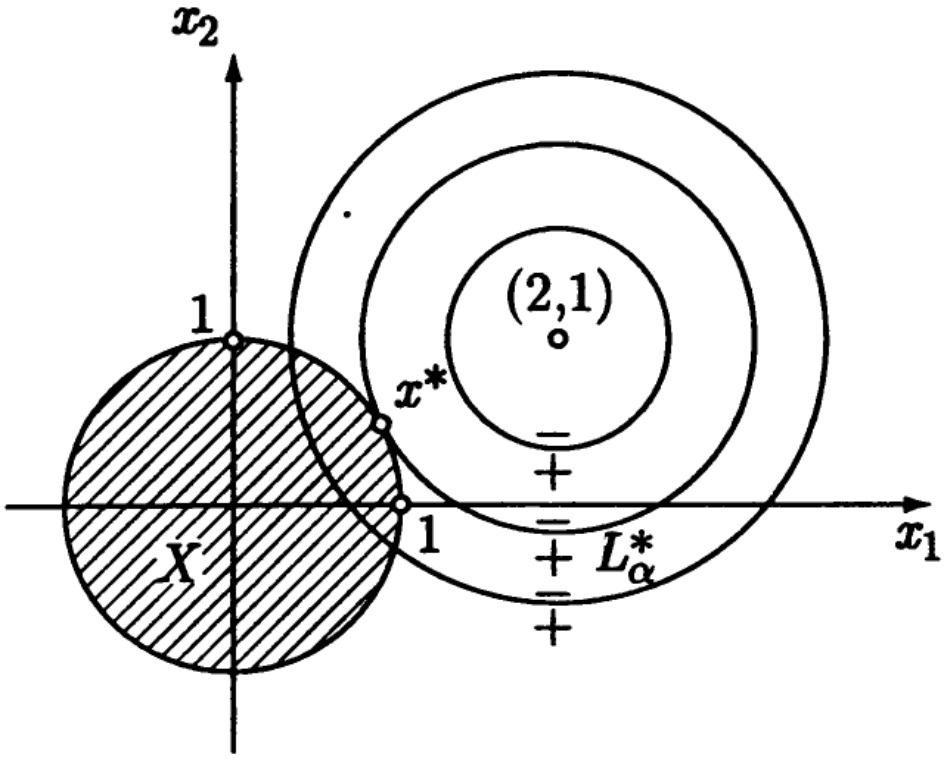


Рис. 13. Геометрическая интерпретация условной минимизации функции двух переменных [3, с.12]. X – множество допустимых решений, L^*_α – линия уровня, содержащая условный минимум x^*

Общая запись для задачи оптимизации (условной и безусловной):

$$x^* \in X,\ X \subseteq R_n \qquad (12)$$

Задачи (10) и (11) имеют различные подходы к решению.

Для задачи (10) могут быть сформулированы сравнительно простые **условия оптимальности** (или **условия экстремума**). Условия, которым должна отвечать точка, являющаяся решением задачи, называются **необходимыми условиями оптимальности**. Условия, из которых следует, что найденное решение является истинным, называются **достаточными условиями оптимальности**. Такая формулировка позволяет теоретически исследовать свойства экстремальных задач указанного типа; на этой основе, в простейших случаях, – явно решить поставленную задачу; обосновать соответствующие численные методы решения более сложных задач.

Задача условной оптимизации (11) предполагает наличие соответствующих ограничений, то есть не всякая точка множества R_n может оказаться решением. При этом описанные условия оптимальности сохраняют свою силу, но их может оказаться недостаточно для корректного решения задачи. Например, для многих условных задач минимум достигается на границе области X, и классический анализ оптимальности неприменим. Таким образом, условные задачи оптимизации почти во всех аспектах более сложные по сравнению с безусловными.

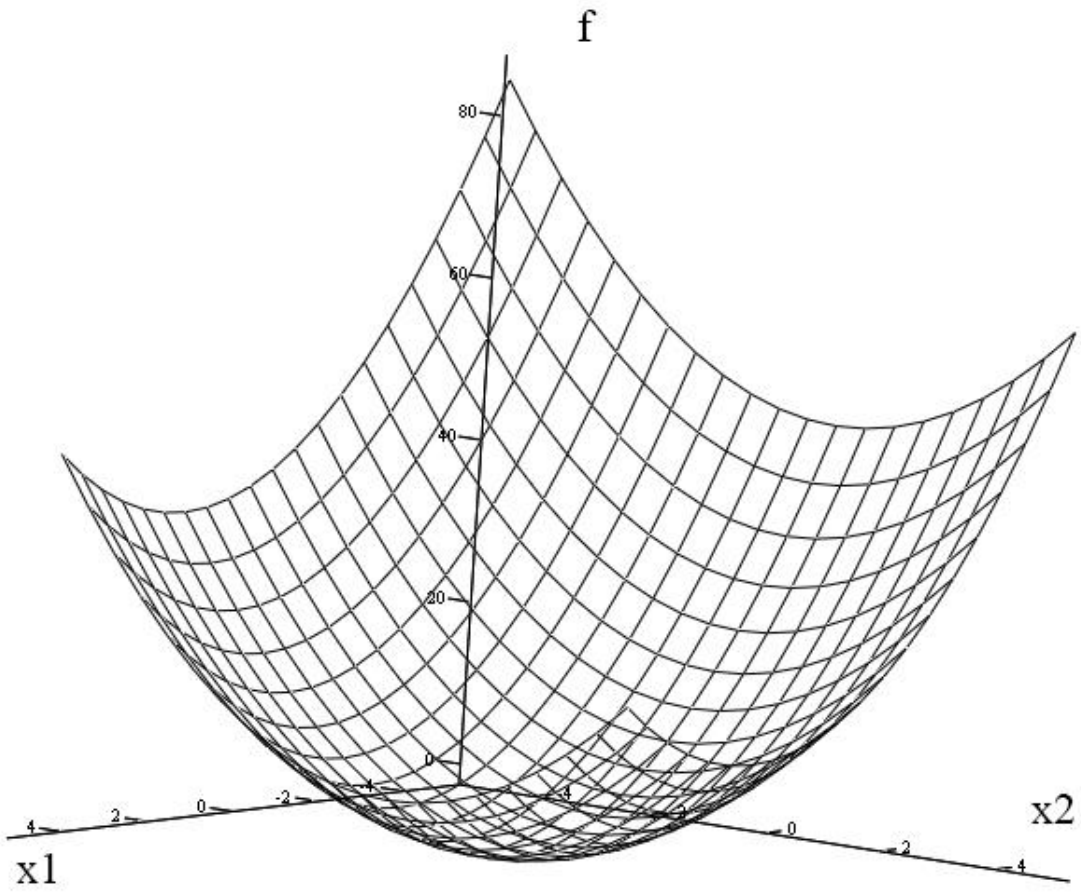


Рис. 14. Функция двух переменных $f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2$

Для **геометрической интерпретации** задач оптимизации (рис. 12) (двумерный случай) полезно изобразить множество допустимых

решений X и несколько линий уровня L_α (см. рис. 5) целевой функции f (здесь α – значение функции на линии уровня).

На рис. 12 «+» и «–» рядом с линиями уровня означают соответственно увеличение функции (значения функции больше α) или её уменьшение. Общим требованием является $x^* \in X$. При условной минимизации возможны два случая: минимум x^* лежит внутри множества X (рис. 12-а), и снаружи его (рис. 12-б). Безусловную минимизацию можно свести к случаю рис. 12-а. Если функция f дифференцируема в точке x, то её градиент, если он ненулевой, ортогонален (то есть в данном случае – перпендикулярен в проекции на плоскость аргументов) к проходящей через x линии уровня и направлен в сторону наибольшего возрастания функции f, т. е. в сторону знака «+».

Рассмотрим это на примере [3, с. 12] – рис. 13.

Здесь множество допустимых решений X – круг единичного радиуса с центром в нуле, а целевая функция – $f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2$ (см. рис 14). Безусловный минимум этой функции соответствует аргументам $x_1 = 2$ и $x_2 = 1$ и равен нулю. То есть линии уровня её L_α представляют собой окружности радиуса $\sqrt{\alpha}$, которые при $x_1 = 2$ и $x_2 = 1$ вырождаются в точку, причём $\alpha \geq 0$. Решением является точка x^*, лежащая на L_α, когда та касается круга X. Решением данной задачи является $x^* = \left(2/\sqrt{5}, 1/\sqrt{5}\right)$; значение целевой функции при этом ~1,528.

v5 § 4. Понятия выпуклого множества и выпуклой функции

В современном аппарате теории оптимизации широко используется такой раздел математики, как **выпуклый анализ**, в котором изучаются **выпуклые множества** и **выпуклые функции**. Используется он для анализа целевой функции на характер и вид экстремумов.

Определение. Множество $X \subseteq R^n$ называется **выпуклым**, если оно содержит всякий отрезок, концы которого принадлежат X, то есть

если для любых точек $x_1, x_2 \in X$ и для любого числа $0 \le \lambda \le 1$ справедливо $\lambda x_1 + (1-\lambda) x_2 \in X$ [2, с. 17]

Другими словами, множество X выпукло, если оно вместе с любыми своими двумя точками x_1 и x_2 содержит и соединяющий их отрезок, т.е. множество вида [3, с.17]

$$[x_1, x_2] = \{x \in R^n \mid x = x_2 + \lambda(x_1 - x_2), \quad 0 \le \lambda \le 1\} \qquad (13)$$

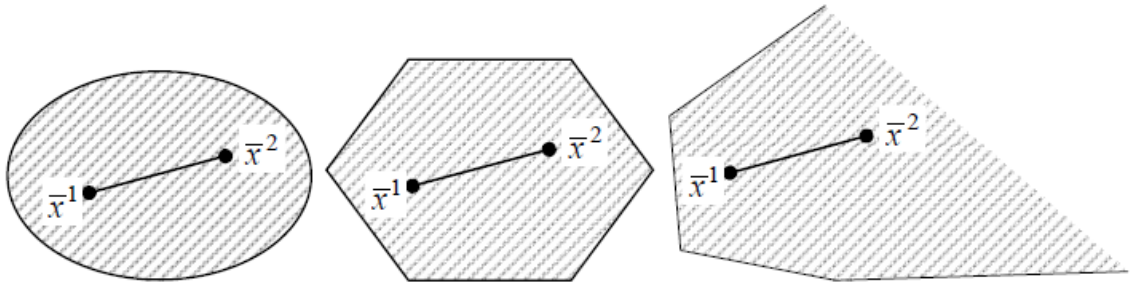


Рис. 15. Выпуклые множества (не содержат впадин, отверстий, промежутков) [7, с. 22]

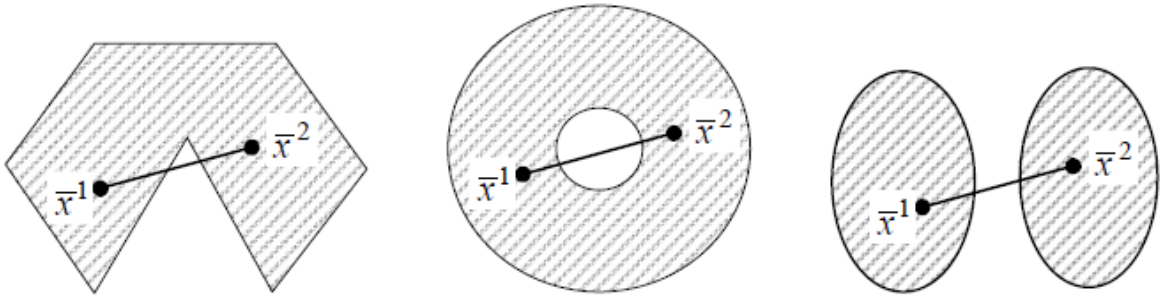


Рис. 16. Невыпуклые множества [7, с. 22]

Проще говоря, выпуклые множества состоят как бы из «одного куска» [2, с. 18], который компактен и не содержит разного рода «дефектов» – впадин, отверстий, промежутков, вмятин, дырок и т. п. На числовой прямой R выпуклыми множествами являются, например, одноточечные множества, интервалы, полуинтервалы, отрезки, полупрямые и, наконец, сама прямая [3, с. 17]. Примерами выпуклых множеств в пространстве R^n служат само пространство, любое его линейное подпространство, одноточечное множество, шар, отрезок,

луч, выходящий из точки и т.п. [3 с. 17]. Примеры выпуклых множеств на плоскости приведены на рис. 15.

На рис. 16 – 18 показаны примеры невыпуклых множеств для пространства R^2. Здесь выбираются две точки, принадлежащие множеству (оно заштриховано), и между ними проводится прямая. Во всех случаях последняя пересекает область, не входящую в это множество. На рис. 17 (слева) множеством является окружность; точки взяты на этой окружности, а прямая, их соединяющая, в окружность не входит.

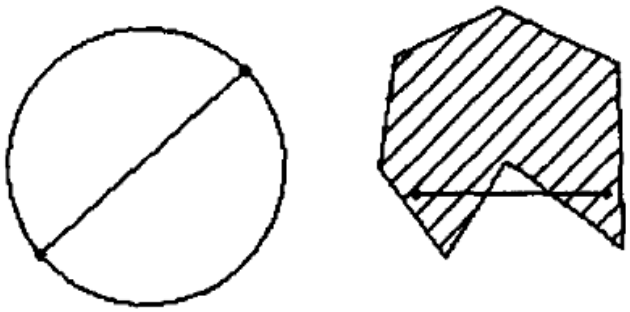


Рис. 17. Примеры невыпуклых множеств [2, с. 17]

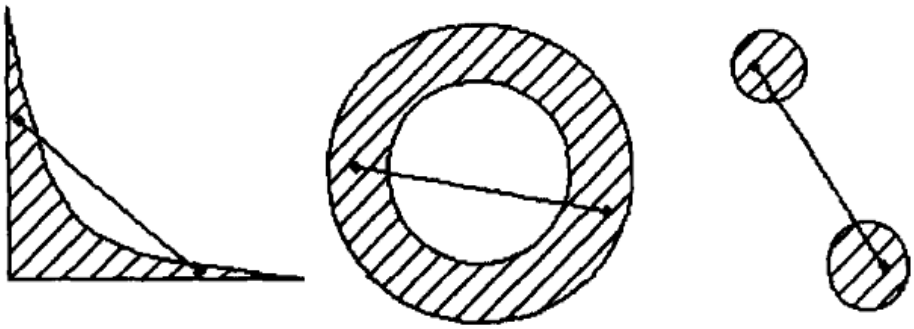


Рис. 18. Примеры невыпуклых множеств [2, с. 17]

Определение. **Функция** $f(x)$, определённая на выпуклом множестве X, называется **выпуклой**, если для любых точек $x_1, x_2 \in X$ и для любого числа $0 \le \lambda \le 1$ справедливо

$$f\left(\lambda x_1 + (1-\lambda) x_2\right) \le \lambda f\left(x_1\right) + (1-\lambda) f\left(x_2\right) \qquad (14)$$

Определение. **Функция** $f(x)$, определённая на выпуклом множестве X, называется **строго выпуклой**, если для любых точек $x_1, x_2 \in X$, $x_1 \neq x_2$ и для любого числа $0 \leq \lambda \leq 1$ справедливо

$$f\left(\lambda x_1 + (1-\lambda)x_2\right) < \lambda f\left(x_1\right) + (1-\lambda) f\left(x_2\right) \qquad (15)$$

(Отличие (15) от (14) в тои, что неравенство строгое).

Определение. [2, с. 18] **Функция** $f(x)$, определённая на выпуклом множестве X, называется **сильно выпуклой** с константой $l > 0$, если для любых точек $x_1, x_2 \in X$, $x_1 \neq x_2$ и для любого числа $0 \leq \lambda \leq 1$ справедливо

$$f\left(\lambda x_1 + (1-\lambda)x_2\right) \leq \lambda f\left(x_1\right) + (1-\lambda) f\left(x_2\right) - \frac{l}{2}\lambda(1-\lambda)\left\|x_1 - x_2\right\|^2 \qquad (16)$$

В выражении (16) добавляется ещё одно слагаемое, величина которого определяется квадратом расстояния между начальной и конечной точками отрезка. При этом степень влияния этого слагаемого определяется величиной константы l, которая не может быть равна нулю [7, с. 23].

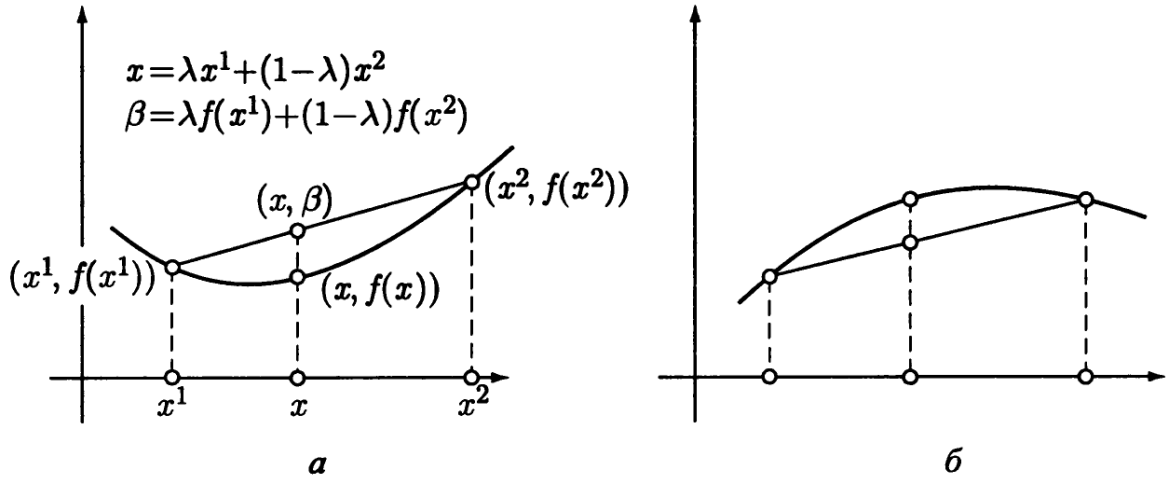


Рис. 19. Выпуклая функция (а), вогнутая функция (б) [3, с. 18]

Таким образом, функция $f(x)$ будет выпуклой, если она целиком лежит не выше отрезка, соединяющего две её произвольные точки. Сильно выпуклая функция одновременно является строго выпуклой и выпуклой, а строго выпуклая – выпуклой. (Выпуклость функции можно определить по матрице Гессе; см. ниже, § 7) [2, с. 18].

Функция $f(x)$ называется **(строго) вогнутой**, если функция $-f(x)$ **(строго) выпукла**. [3, с. 18]

Иногда **выпуклые** функции называются **выпуклыми вниз**, а **вогнутые** – **выпуклыми вверх**.

Геометрическая интерпретация выпуклости представлена на рис. 19. Если для выпуклой функции провести произвольную хорду, то её любая точка лежит не ниже точки графика того же аргумента.

Примеры: функции $f(x)=x^2$ и $f(x)=e^x$ выпуклы на одномерном множестве R, а $f(x)=\ln x$ – вогнута на множестве положительных чисел.

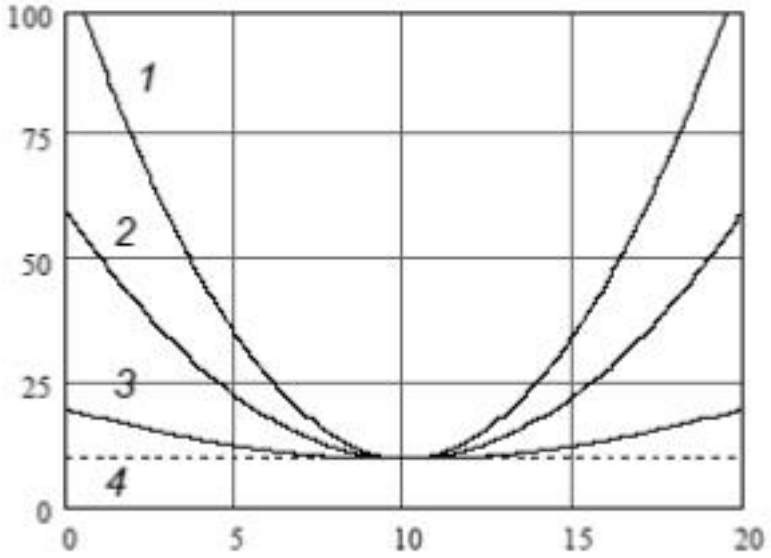


Рис. 20. Функции выпуклые, строго выпуклые и сильно выпуклые [7, с. 24]

На рис. 20 сравниваются выпуклые, строго выпуклые и сильно выпуклые функции. Здесь представлены три графика различной кривизны (1, 2 и 3) и прямая (4). Выпуклыми функциями являются все четыре приведённые, включая прямую линию (это отвечает определению). Строго выпуклые – 1, 2 и 3. Сильно выпуклыми функциями можно считать 1 относительно 2 и 2 относительно 3 (это определяется параметром l, входящим в (16)).

Характер выпуклости функции используется для поиска экстремумов (максимумов или минимумов):

– если функция $f(x)$ выпуклая на выпуклом множестве X, то всякая точка локального минимума (максимума) является точкой её глобального минимума (максимума) на X;

– если выпуклая функция $f(x)$ достигает своего минимума (максимума) в двух различных точках, то она достигает минимума (максимума) во всех точках отрезка, соединяющего эти две точки;

– если функция $f(x)$ – строго выпуклая на выпуклом множестве X, то она может достигать своего глобального минимума (максимума) на X не более чем в одной точке. [7, с. 25]

v6 § 5. Унимодальные функции

При решении задач минимизации функций одним из важнейших является понятие унимодальности функций. Функция будет унимодальной в данной области, если она будет иметь на ней единственный экстремум. Если это требование не выполняется, функция будет мультимодальной.

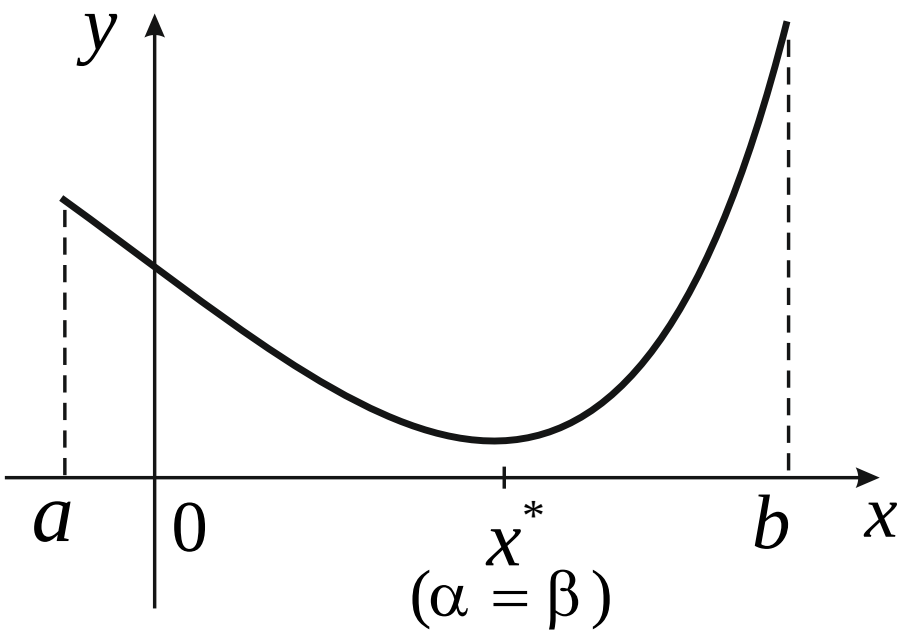


Рис. 21. Строго унимодальная, непрерывная, дифференцируемая функция

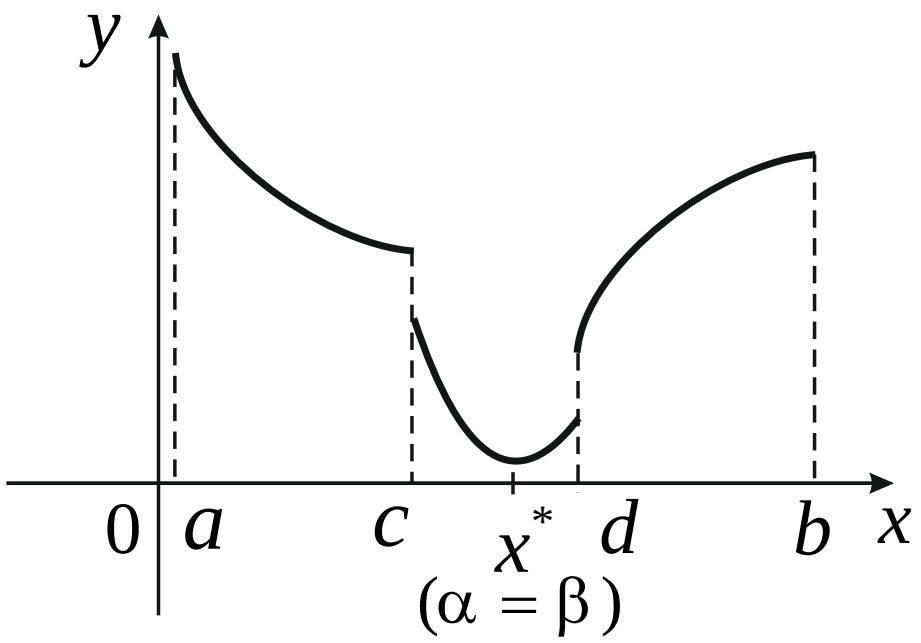


Рис. 22. Строго унимодальная, недифференцируемая функция. $x = c$ и $x = d$ имеется разрыв I-го рода

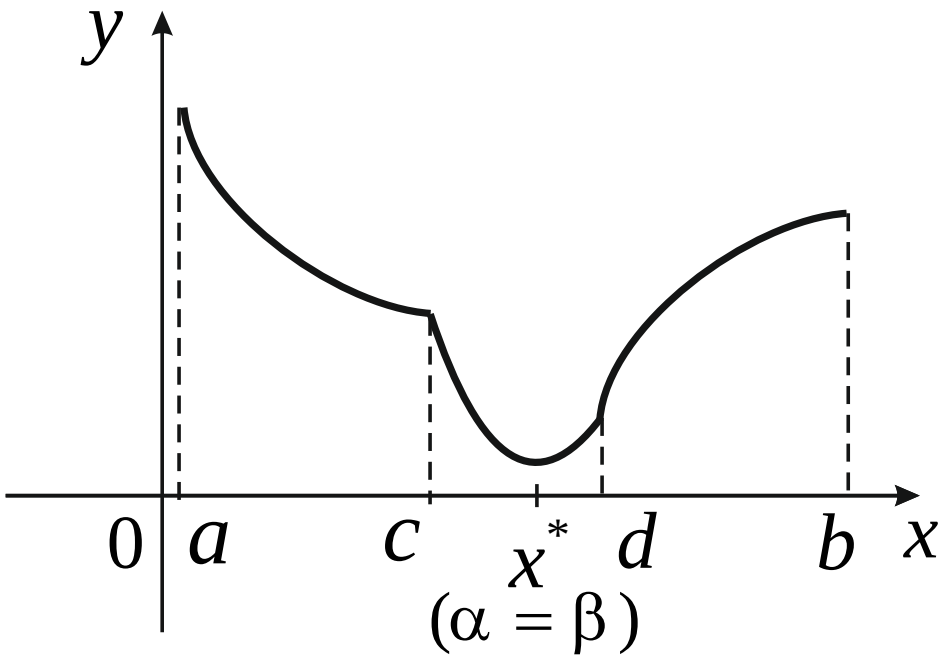


Рис. 23. Строго унимодальная, непрерывная, недифференцируемая функция. При x = c имеет разрыв I-го рода, при $x = c$ и $x = d$ производной у функции $f(x)$ не существует

Определение. **Функция** $f(x)$ **унимодальна** на отрезке $[a,b]$, если существуют точки α,β, для которых выполняется $a \le \alpha \le \beta \le b$ и такие, что: 1) $f(x)$ строго монотонно убывает на $[a,\alpha]$, 2) $f(x)$ строго монотонно возрастает на $[\beta,b]$, 3) для $x \in [\alpha,\beta]$ выполняется $f(x) = \min\limits_{x\in[\alpha,\beta]} f(x)$. Если $\alpha = \beta$, то $f(x)$ **строго унимодальна**. [8, с. 10]

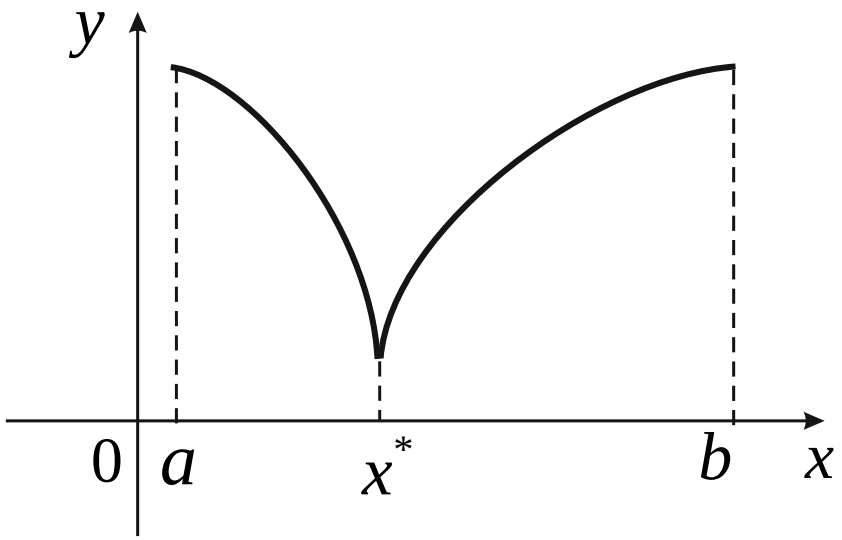


Рис. 24. Строго унимодальная, непрерывная, недифференцируемая функция

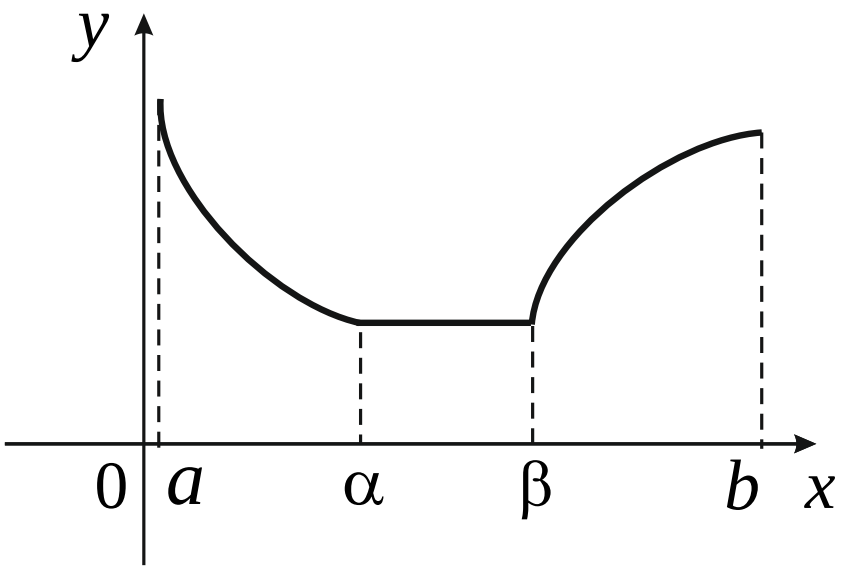


Рис. 25. Нестрого унимодальная, непрерывная, недифференцируемая функция

Это определение отрабатывает ситуацию, показанную на рис. 2 (множество точек между u_8 и u_9 представляет собой минимум функции). Если минимум – единственная точка, то речь должна идти о строгой унимодальности. Для случая максимума мы будем иметь аналогичное определение с небольшими изменениями.

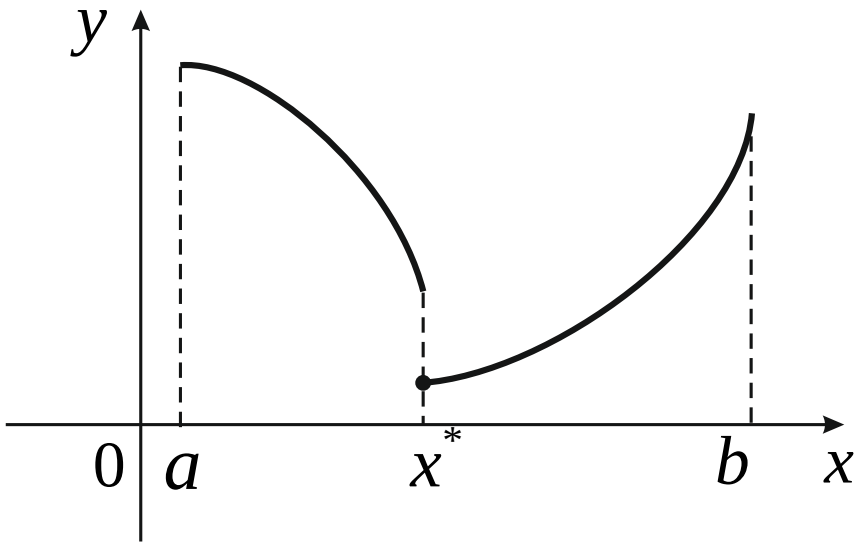


Рис. 26. Унимодальная, недифференцируемая функция; в точке x^* – разрыв I-го рода

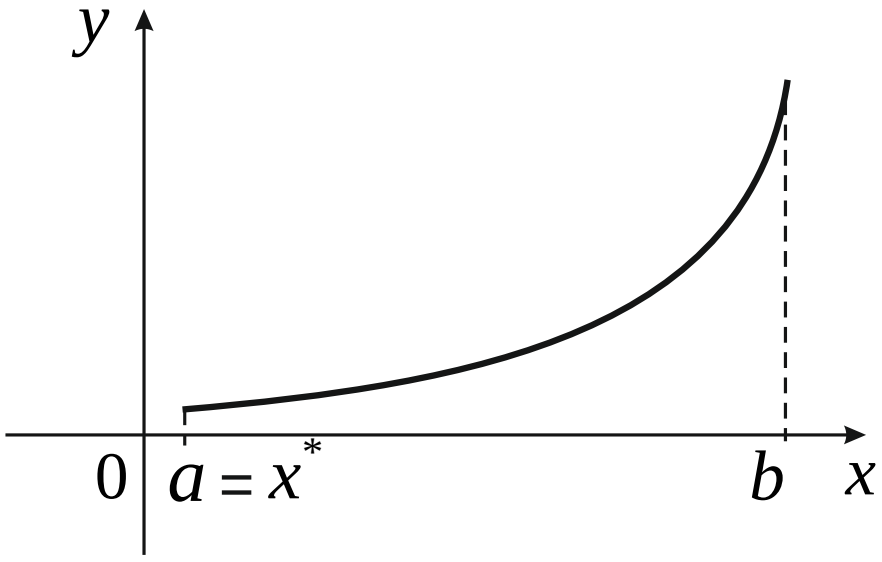


Рис. 27. Строго унимодальная, непрерывная, дифференцируемая функция

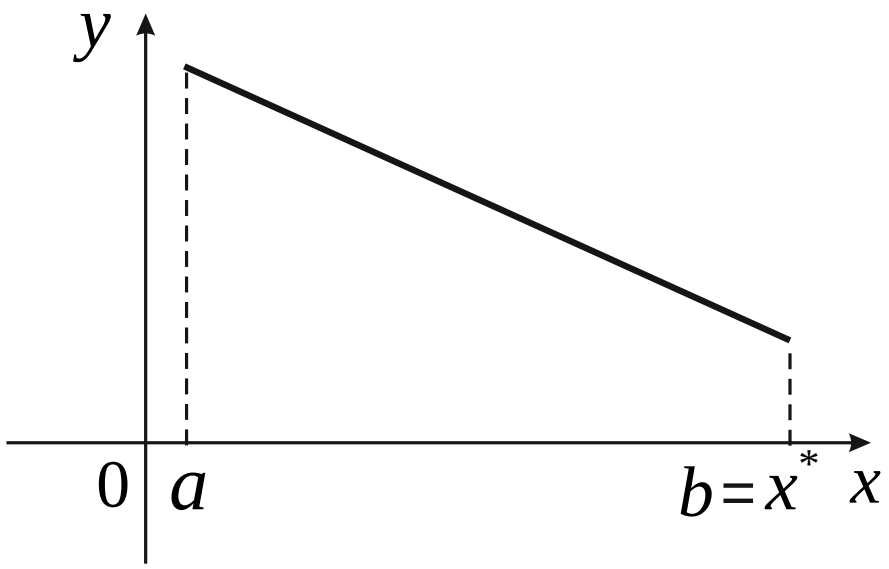


Рис. 28. Строго унимодальная, непрерывная, дифференцируемая функция

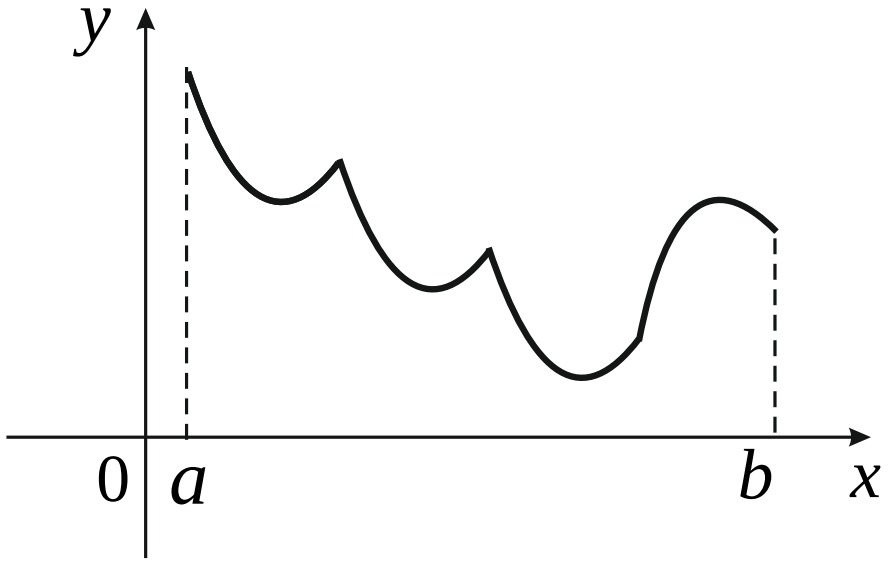


Рис. 29. Мультимодальная функция

Для унимодальных функций могут не выполняться условия непрерывности и дифференцируемости. Левый $[a,\alpha]$ или правый $[\beta,b]$ отрезки могут отсутствовать; тогда экстремум достигается на границе области. Примеры унимодальных функций даны на рис. 21 – 28, пример мультимодальной – на рис. 29.

Свойство унимодальной функции: если $f(x_1) \le f(x_2)$ при $a < x_1 < x_2 < b$, то $x^* < x_2$; если $f(x_1) \ge f(x_2)$ при $a < x_1 < x_2 < b$, то $x^* > x_1$.

Критерии проверки унимодальности функции $f(x)$:

Теорема 1. Если функция $f(x)$ дифференцируема на отрезке $[a,b]$ и производная $f'(x)$ не убывает на этом отрезке, то $f(x)$ унимодальна на $[a,b]$.

Теорема 2. Если функция $f(x)$ дважды дифференцируема на отрезке $[a,b]$ и $f''(x) \ge 0$ при $x \in [a,b]$, то $f(x)$ унимодальна на $[a,b]$. [6, с. 34].

На практике широко используется следующее свойство: если унимодальная функция выпукла, то она имеет глобальный минимум; если – вогнута, то она имеет глобальный максимум [8, с. 12].

Рассмотренные примеры относились только к одномерным функциям, однако всё это без труда можно отнести и к многомерным задачам.

Все существующие вычислительные методы оптимизации пригодны к применению только для унимодальных функций. Таким образом, задачи выявления всех локальных экстремумов, и из них – экстремумов глобальных, на всей числовой оси или на её части, являются отдельной сложной задачей. В настоящем пособии рассматриваются только методы оптимизации для унимодальных функций.

v7 § 6. Общий подход к решению задачи оптимизации

Задача оптимизации является одной из сложнейших математических задач, а её решение невозможно без предварительного математического исследования и глубокого анализа само́й конкретной задачи и сопутствующих условий.

Как уже говорилось, наличие или отсутствие ограничивающих решение условий, позволяет разделить рассматриваемые задачи на безусловную и условную оптимизацию, которые требуют различных

подходов к решению и использования разных, как правило, конкретных методов.

Другие сопутствующие условия чаще всего позволяют получить дополнительную информацию, облегчающую поиск решения.

Сравнительно простой априорный анализ целевой функции, в плане влияющих на неё факторов, позволяет отнести задачу к одномерной или многомерной оптимизации. Это также позволяет выбрать те или иные методы для решения задачи.

Значительно более сложным является вопрос о само́м наличии экстремумов в рамках поставленной задачи минимизации. Математическая теория позволяет провести такой анализ и априорно доказать существование само́го решения или же его отсутствие. В последнем случае можно скорректировать постановку задачи, например – расширить диапазон поиска экстремума по аргументам.

Весьма полезным может оказаться построение графика одномерной или двумерной целевой функции в достаточно широких интервалах, особенно при наличии многих локальных экстремумов. Это позволяет наглядно убедиться в наличии решений и даже подобрать диапазоны по аргументам, содержащие экстремумы, где будут выполняться условия унимодальности функции. Последнее важно в плане использования конкретных методов минимизации. К сожалению, это не гарантирует полного исследования целевой функции на всей числовой оси.

В этих, а также в более сложных многомерных случаях, вместо построения графиков можно использовать алгоритмы сканирования по всем аргументам с выявлением диапазонов, в которых может находиться экстремум. Здесь, опять-таки, нет гарантии полноты исследования.

Однако очень часто задача не является такой сложной, наличие экстремума очевидно, и прямое применение конкретных методов позволяет быстро решить задачу даже без предварительного математического исследования.

В некоторых простых случаях предварительный математический анализ позволяет непосредственно определить экстремумы. Это так называемый **классический метод** (или – аналитический).

Общий анализ включает в себя **необходимые условия** наличия экстремума и **достаточные условия**. Точки, которые «подозреваются»

на экстремальность, должны быть проверены по этим условиям. При этом, если для точки выполняются необходимые условия, то экстремум в этой точке может быть, а может и не быть. Если же эти условия не выполняются, то экстремума точно нет. Затем точки проверяются по критериям достаточных условий, выполнение которых точно подтверждает наличие экстремума. Причём, если достаточные условия не выполняются, это не гарантирует отсутствие экстремума в этой точке. Другими словами, достаточные условия, при их выполнении, могут доказать экстремальность, но не могут её опровергнуть, если не выполняются. Таким образом, использование для анализа только достаточных условий не даст полную картину. Рассмотрение необходимых условий может привести к получению систем уравнений и, иногда, неравенств. Системы эти могут быть решены и получены точки, «подозрительные» на экстремальность. Окончательно подтвердить экстремум можно с помощью достаточных условий. Также можно при этом определить вид экстремума – максимум или минимум.

Необходимые условия подразделяются на **условия первого** и **второго порядков**. Критериями здесь являются **градиент** (для первого порядка) и **матрица Гессе** – **гессиан** (для второго порядка).

При **безусловной оптимизации** решение X ищется на всём числовом пространстве (множестве), то есть $X = R^n$. При **условной оптимизации** на область допустимых решений наложены ограничения, то есть $X \subset R^n$ (то есть допустимые решения представляют собой лишь часть множества R^n). Ограничениями могут быть, например, только положительные числа, целые отрицательные, не превышающие какого-то заданного значения и. д. Эти ограничения, чаще всего, могут быть заданы в виде систем равенств и неравенств. Они могут существенно осложнять минимизацию, или же, наоборот, упрощать решение. Часто бывает, что если ограничения снять, то задача вообще не имеет решения. Необходимые и достаточные условия безусловной оптимизации будут рассмотрены в § 8.

v8 § 7. Краткие сведения о градиенте и матрице Гессе. Некоторые действия с определителями

Градиент определён для непрерывно дифференцируемой функции многих переменных (в том числе – одной) в точке x (x – вектор). Обозначается он $grad\ f(x)$, или, в операторной форме, $\nabla f(x)$, где ∇ – **оператор Гамильтона** (в физике – вектор набла):

$$\nabla = \frac{\partial}{\partial x}\cdot\vec{i} + \frac{\partial}{\partial y}\cdot\vec{j} + \frac{\partial}{\partial z}\cdot\vec{k}, \tag{17}$$

то есть градиент для трёхмерных пространственных координат x, y и z представляет собой сумму первых частных производных по координатам, умноженных на единичные вектора $\vec{i}$, $\vec{j}$ и $\vec{k}$, которые задают направление. Таким образом, **градиент является вектором**, хотя берётся от **скалярной функции**. (В пртивовес этому можно заметить, что похожий оператор дивергенции

$$div = \frac{\partial}{\partial x} + \frac{\partial}{\partial y} + \frac{\partial}{\partial z} \tag{18}$$

является скаляром, а берётся от векторной функции).

Таким образом, градиент в трехмерном пространстве представляется для нашего случая как

$$\nabla f(x) = \frac{\partial f(x)}{\partial x_1}\cdot\vec{i} + \frac{\partial f(x)}{\partial x_2}\cdot\vec{j} + \frac{\partial f(x)}{\partial x_3}\cdot\vec{k} \tag{19}$$

Градиент также может быть представлен для многомерной функции, где координаты – необязательно пространственные, как вектор-столбец первых производных:

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \dots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix} \tag{20}$$

Градиент направлен в сторону наибольшего (то есть наиболее крутого) возрастания функции. Он является критерием необходимых

условий первого порядка, так как используется именно первая производная.

Наряду с градиентом можно определить **вектор антиградиента**, который равен (по модулю) градиенту, но противоположно направлен.

Если целевая функция $f(x)$ **дважды непрерывно дифференцируема** в точке x, то для неё можно построить **матрицу Гессе** (**гессиан**) $H(x)$ – квадратную матрицу вторых частных производных (включая смешанные):

$$H(x)=\begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \dots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & \dots & h_{1n} \\ h_{21} & h_{22} & \dots & h_{2n} \\ \dots & \dots & \dots & \dots \\ h_{n1} & h_{n2} & \dots & h_{nn} \end{pmatrix} \quad (21)$$

(здесь элементы матрицы обозначены h_{ij}).

Поскольку порядок взятия смешанных производных значения не имеет, матрица Гессе – симметричная (относительно главной диагонали).

Исходя из формулы Тэйлора с использованием градиента и матрицы Гессе, приращение функции $f(x)$ в точке x можно записать [2, с. 14]:

$$\Delta f(x) = f(x+\Delta x) - f(x) = \\ = \nabla f(x)^T \Delta x + \frac{1}{2}\Delta x^T H(x)\Delta x + o\left(\|\Delta x\|^2\right), \quad (22)$$

где $o\left(\|\Delta x\|^2\right)$ – сумма всех членов разложения, имеющих порядок выше второго, $\Delta x^T H(x)\Delta x$ – квадратичная форма. Напомним, что x^T – матрица (или вектор), транспонированная из матрицы x, то есть где строки заменены на столбцы; например: если $x=\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, то $x^T=(x_1, x_2)$.

Определение. [2, с. 15-16] Квадратичная форма $\Delta x^T H(x)\Delta x$ (а также соответствующая матрица Гессе $H(x)$) называется:

положительно определённой ($H(x)>0$), если для любого ненулевого Δx выполняется неравенство $\Delta x^T H(x)\Delta x>0$;

отрицательно определённой ($H(x)<0$), если для любого ненулевого Δx выполняется неравенство $\Delta x^T H(x)\Delta x<0$;

положительно полуопределённой ($H(x)\geq 0$), если для любого Δx выполняется неравенство $\Delta x^T H(x)\Delta x\geq 0$ и имеется отличный от нуля вектор Δx, для которого $\Delta x^T H(x)\Delta x=0$;

отрицательно полуопределённой ($H(x)\leq 0$), если для любого Δx выполняется неравенство $\Delta x^T H(x)\Delta x\leq 0$ и имеется отличный от нуля вектор Δx, для которого $\Delta x^T H(x)\Delta x=0$;

неопределённой ($(H(x)>0)\vee(H(x)<0)$), если существуют такие векторы Δx, Δx, что выполняются неравенства $\Delta x^T H(x)\Delta x>0,\ \Delta\left(x\right)^T H(x)\Delta x<0$;

тождественно равной нулю ($H(x)=0$), если для любого Δx выполняется неравенство $\Delta x^T H(x)\Delta x=0$.

Рассмотрим пример вычисления градиента, матрицы Гессе и квадратичной формы для функции $f(x)=x_1^3+x_2^2$.

$$\frac{\partial f(x)}{\partial x_1}=3x_1^2;$$

$$\frac{\partial f(x)}{\partial x_2}=2x_2;$$

$$\frac{\partial^2 f(x)}{\partial x_1^2}=6x_1;$$

$$\frac{\partial^2 f(x)}{\partial x_2^2}=2;$$

$$\frac{\partial^2 f(x)}{\partial x_1\partial x_2}=\frac{\partial^2 f(x)}{\partial x_2\partial x_1}=0.$$

Тогда:

$$\nabla f(x)=\begin{pmatrix}3x_1^2\\2x_2\end{pmatrix};\ H(x)=\begin{pmatrix}6x_1 & 0\\0 & 2\end{pmatrix}.$$

Квадратичная форма:

$$\Delta x^T H(x)\Delta x = \begin{pmatrix}\Delta x_1 & \Delta x_2\end{pmatrix}\begin{pmatrix}6x_1 & 0\\ 0 & 2\end{pmatrix}\begin{pmatrix}\Delta x_1\\ \Delta x_2\end{pmatrix}.$$

Например, для конкретной точки $x = \begin{pmatrix}1\\ 0\end{pmatrix}$ имеем:

$$\nabla f(x) = \begin{pmatrix}3\\ 0\end{pmatrix};\ H(x) = \begin{pmatrix}6 & 0\\ 0 & 2\end{pmatrix};$$

$$\Delta x^T H(x)\Delta x = \begin{pmatrix}\Delta x_1 & \Delta x_2\end{pmatrix}\begin{pmatrix}6 & 0\\ 0 & 2\end{pmatrix}\begin{pmatrix}\Delta x_1\\ \Delta x_2\end{pmatrix} =$$

$$= \begin{pmatrix}6\Delta x_1 & 2\Delta x_2\end{pmatrix}\begin{pmatrix}\Delta x_1\\ \Delta x_2\end{pmatrix} = 6\Delta x_1^2 + 2\Delta x_2^2$$

;

а для точки $x = \begin{pmatrix}3\\ 1\end{pmatrix}$:

$$\nabla f(x) = \begin{pmatrix}27\\ 2\end{pmatrix};\ H(x) = \begin{pmatrix}18 & 0\\ 0 & 2\end{pmatrix};$$

$$\Delta x^T H(x)\Delta x = \begin{pmatrix}\Delta x_1 & \Delta x_2\end{pmatrix}\begin{pmatrix}18 & 0\\ 0 & 2\end{pmatrix}\begin{pmatrix}\Delta x_1\\ \Delta x_2\end{pmatrix} =$$

$$= \begin{pmatrix}18\Delta x_1 & 2\Delta x_2\end{pmatrix}\begin{pmatrix}\Delta x_1\\ \Delta x_2\end{pmatrix} = 18\Delta x_1^2 + 2\Delta x_2^2$$

.

Из полученных выражений видно, что $\forall x \neq 0$ имеем $\Delta x^T H(x)\Delta x > 0,$ следовательно для обеих рассмотренных точек квадратичная форма и матрица Гессе положительно определены.

Выпуклость функции (см. § 4) можно определить по матрице Гессе [2, с. 18]:

если $H(x) \geq 0 \quad \forall x \in R^n$, то функция выпуклая;

если $H(x) > 0 \quad \forall x \in R^n$, то функция строго выпуклая;

если $H(x) \geq lE \quad \forall x \in R^n$, где E – единичная матрица, то функция сильно выпуклая.

Для формулировки необходимых и достаточных условий наличия экстремума требуется не только понятие о само́й матрице

Гессе, но и об определителе матрицы и некоторых действий с ними. рассмотрим это на примере именно матрицы Гессе (21).

Определение. **Определителем матрицы** H (в нашем случае – (21)) является выражение, составленное из элементов этой матрицы. Самое распространённое обозначение его – $\det H(x)$ (для нашего случая). Выражение имеет вид (см. (21)):

$$\det H(x) = \begin{vmatrix} h_{11} & h_{12} & \dots & h_{1n} \\ h_{21} & h_{22} & \dots & h_{2n} \\ \dots & \dots & \dots & \dots \\ h_{n1} & h_{n2} & \dots & h_{nn} \end{vmatrix} \tag{23}$$

Понятие определителя имеет смысл только для квадратных матриц. Определитель (детерминант) представляет собой скалярную величину и может быть подсчитан по определённым правилам (то есть может быть найдено его числовое значение). При этом используется понятие минора определителя.

Определение. **Минором определителя** называется определитель, получаемый из исходного путём вычёркивания одинакового количества строк и столбцов с определёнными номерами.

Для **матрицы первого порядка** (с единственным элементом) (h_{11}) определитель равен самому этому элементу:

$$\det(h_{11}) = |h_{11}| = h_{11} \tag{24}$$

Для **матрицы размерностью** 2×2 определитель вычисляется по следующей схеме:

$$\det H(x) = \begin{vmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{vmatrix} = h_{11}\cdot h_{22} - h_{12}\cdot h_{21} \tag{25}$$

При вычислении **определителя** 3×3 используется его разложение по первой строке; при этом получаются определители $2\times 2,$ которые подсчитываются по формуле (25):

$$\det H(x) = \begin{vmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{vmatrix} = \tag{26}$$

$$= h_{11}\cdot\begin{vmatrix} h_{22} & h_{23} \\ h_{32} & h_{33} \end{vmatrix} - h_{12}\cdot\begin{vmatrix} h_{21} & h_{23} \\ h_{31} & h_{33} \end{vmatrix} + h_{13}\cdot\begin{vmatrix} h_{21} & h_{22} \\ h_{31} & h_{32} \end{vmatrix}$$

То есть каждый элемент первой строки умножается на минор, получаемый вычёркиванием строки и столбца, в которые входит этот элемент. Знак перед каждым слагаемым определяется суммой индексов соответствующего элемента: если сумма чётная, то берётся знак «+», если – нечётная, то берётся знак «–».

Общая формула для вычисления **определителей более высоких порядков**, чем второй, выглядит следующим образом (**разложение по строке**, чаще всего – по первой):

$$\det H(x) = \sum_{j=1}^{n} (-1)^{1+j} h_{1j} M_{1j} \tag{27}$$

где M_{1j} – минор, получаемый вычёркиванием 1-ой строки и j-го столбца. Таким образом, формула (27) позволяет лишь «понизить порядок» определителя; вычисление полученных при этом миноров осуществляется по той же самой формуле (27) (то есть (27) – рекурсивная формула). Разумеется, определитель 3×3 также может быть вычислен по этой формуле.

Определение. **Угловыми минорами** определителя (в нашем случае – (23)) называются определители первого Δ_1. второго Δ_2 и последующих порядков. получаемые из исходного определителя по следующей схеме (см. (23)):

$$\Delta_1 = h_{11}; \tag{28}$$

$$\Delta_2 = \begin{vmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{vmatrix}; \tag{29}$$

$$\Delta_3 = \begin{vmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{vmatrix}; \tag{30}$$

и так далее до:

$$\Delta_n = \begin{vmatrix} h_{11} & h_{12} & \ldots & h_{1n} \\ h_{21} & h_{22} & \ldots & h_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ h_{n1} & h_{n2} & \ldots & h_{nn} \end{vmatrix}. \tag{31}$$

Угловой минор n-го порядка совпадает с исходным определителем.

Определение. **Главными минорами** определителя n-го порядка (в нашем случае – (23)) называются определители m-го порядка $(m \le n)$, которые получаются из исходного определителя вычёркиванием $(n-m)$ строк и $(n-m)$ столбцов с теми же номерами.

Примеры главных миноров определителя 5-го порядка:
определитель

$$\det H(x) = \begin{vmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ h_{31} & h_{32} & h_{33} & h_{34} & h_{35} \\ h_{41} & h_{42} & h_{43} & h_{44} & h_{45} \\ h_{51} & h_{52} & h_{53} & h_{54} & h_{55} \end{vmatrix}; \qquad (32)$$

главные миноры

$$\Delta = \begin{vmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ h_{31} & h_{32} & h_{33} & h_{34} & h_{35} \\ h_{41} & h_{42} & h_{43} & h_{44} & h_{45} \\ h_{51} & h_{52} & h_{53} & h_{54} & h_{55} \end{vmatrix}; \qquad (33)$$

$$\Delta = \begin{vmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ h_{31} & h_{32} & h_{33} & h_{34} & h_{35} \\ h_{41} & h_{42} & h_{43} & h_{44} & h_{45} \\ h_{51} & h_{52} & h_{53} & h_{54} & h_{55} \end{vmatrix}; \qquad (34)$$

$$\Delta = \begin{vmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ h_{31} & h_{32} & h_{33} & h_{34} & h_{35} \\ h_{41} & h_{42} & h_{43} & h_{44} & h_{45} \\ h_{51} & h_{52} & h_{53} & h_{54} & h_{55} \end{vmatrix}; \qquad (35)$$

$$\Delta = \begin{vmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ h_{31} & h_{32} & h_{33} & h_{34} & h_{35} \\ h_{41} & h_{42} & h_{43} & h_{44} & h_{45} \\ h_{51} & h_{52} & h_{53} & h_{54} & h_{55} \end{vmatrix}. \quad (36)$$

Определение. [2, с. 22] **Собственные значения** $\lambda_i, i = 1, ..., n$ матрицы $H\left(x^*\right)$ размера $\left(n \times n\right)$ находятся как корни характеристического уравнения (алгебраического уравнения n-й степени):

$$\left|h\left(x^*\right) - \lambda E\right| = \begin{vmatrix} h_{11} - \lambda & h_{12} & \ldots & h_{1n} \\ h_{21} & h_{22} - \lambda & \ldots & h_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ h_{n1} & h_{n2} & \ldots & h_{nn} - \lambda \end{vmatrix} = 0. \quad (37)$$

То есть рассчитывается определитель при неизвестном λ и приравнивается нулю. Из полученного уравнения находится λ. Необходимо заметить, что собственные значения вещественной симметричной матрицы $H\left(x^*\right)$ вещественны.

v9 § 8. Необходимые и достаточные условия безусловной оптимизации. Классический метод

Классический метод оптимизации основан на дифференциальном исчислении и позволяет аналитически исследовать целевую функцию, найти точки, **подозрительные на экстремум**, и доказать наличие экстремума в них. Как уже отмечалось, при этом рассматриваются необходимые и достаточные условия.

Если функция $f(x)$ кусочно непрерывная и кусочно гладкая на отрезке $[a, b]$, то на этом отрезке существует конечное количество точек, подозрительных на экстремум. Такими точками могут быть точки, в которых 1) первая производная существует и равна нулю (рис. 30); 2) функция непрерывна, но первая производная в этой точке не

существует (рис. 24); 3) функция терпит разрыв первого рода (рис. 26); 4) достигается один из концов отрезка a или b (рис. 27).

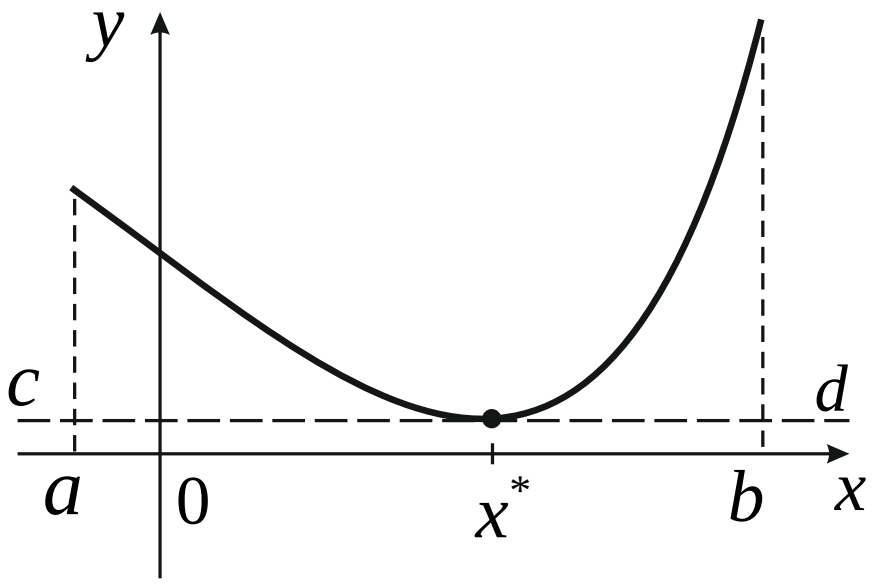


Рис. 30. В точке экстремума x^* функция имеет первую производную (касательная – cd), равную нулю

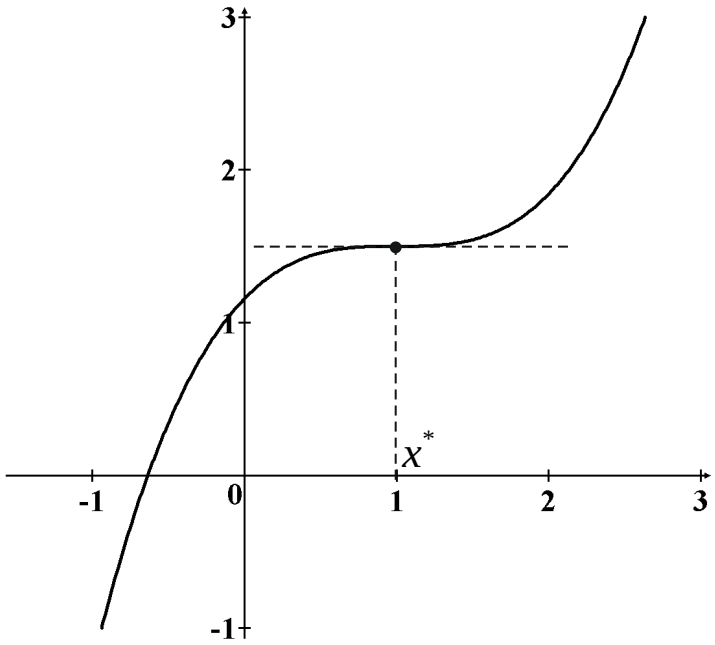


Рис. 31. Стационарная точка x^*, не являющаяся экстремумом (функция $y = (0.7 \cdot (x-1))^3 + 1.5$)

После выявления точек, подозрительных на экстремум, необходимо провести дальнейший математический анализ по доказательству экстремальности этих точек. В некоторых случаях такой анализ удаётся сделать сравнительно легко. Критерии этого приводятся ниже.

Более строгие критерии оптимальности можно применить в случае функций, непрерывно дважды дифференцируемых на отрезке $[a, b]$. Общим подходом к решению здесь является нахождение точек, подозрительных на экстремум, с помощью необходимых условий первого и второго порядков. а, затем, – доказательстао существования в них экстремальности с помощью достаточных условий.

Необходимые условия экстремума первого порядка [2, с. 22]

Пусть $x^* \in R^n$ есть точка локального минимума (максимума) функции $f(x)$ на множестве R^n и $f(x)$ дифференцируема в точке x^*. Тогда градиент функции $f(x)$ в точке x^* равен нулю, то есть

$$\nabla f\left(x^*\right) = 0 \text{ или } \frac{\partial f\left(x^*\right)}{\partial x_i} = 0, \quad i = 1, ..., n. \tag{38}$$

Условие (38) может выполняться не только для экстремальных точек (рис. 31). поэтому оно является лишь необходимым. Все точки, удовлетворяющие (38), называются **стационарными**.

Необходимые условия экстремума второго порядка [2, с. 22]

Пусть точка x^* есть точка локального минимума (максимума) функции $f(x)$ на множестве R^n и функция $f(x)$ дважды дифференцируема в этой точке. Тогда матрица Гессе $H(x^*)$ функции $f(x)$, вычисленная в точке x^*, является положительно полуопределённой (отрицательно полуопределённой), то есть

$$H(x^*) \geq 0 \tag{39}$$

$$\left(H(x^*) \leq 0\right) \tag{40}$$

Достаточные условия экстремума [2, с. 23]

Пусть функция $f(x)$ в точке $x^* \in R^n$ дважды дифференцируема, её градиент равен нулю, а матрица Гессе является положительно определённой (отрицательно определённой), то есть

$$\nabla f\left(x^*\right)=0 \text{ и } H(x^*)>0 \quad (41)$$

$$\left(H(x^*)<0\right) \quad (42)$$

Тогда точка x^* есть точка локального минимума (максимума) функции $f(x)$ на множестве R^n.

Последние два условия требуют детализации. При этом можно использовать два способа.

Первый способ – с использованием угловых и главных миноров.

Критерий проверки достаточных условий экстремума (критерий Сильвестра) [2, с. 23]

1. Для того чтобы матрица Гессе $H(x^*)$ была положительно определённой $\left(H(x^*)>0\right)$ и точка x^* являлась точкой локального минимума, необходимо и достаточно, чтобы знаки угловых миноров были строго положительны:

$$\Delta_1>0, \Delta_2>0, ..., \Delta_n>0 \quad (43)$$

2. Для того чтобы матрица Гессе $H(x^*)$ была отрицательно определённой $\left(H(x^*)<0\right)$ и точка x^* являлась точкой локального максимума, необходимо и достаточно, чтобы знаки угловых миноров чередовались, начиная с отрицательного:

$$\Delta_1<0, \Delta_2>0, \Delta_3<0, ..., (-1)^n \Delta_n>0 \quad (44)$$

Критерий проверки необходимых условий экстремума второго порядка [2, с. 24]

1. Для того чтобы матрица Гессе $H(x^*)$ была положительно полуопределённой $\left(H(x^*)\geq 0\right)$ и точка x^* могла бы являться точкой локального минимума, необходимо и достаточно, чтобы все главные миноры определителя матрицы Гессе были неотрицательны.

2. Для того чтобы матрица Гессе $H(x^*)$ была отрицательно полуопределённой $\left(H(x^*)\leq 0\right)$ и точка x^* могла бы являться точкой локального максимума, необходимо и достаточно, чтобы все главные

миноры чётного порядка были неотрицательны, а все главные миноры нечётного порядка – неположительны.

Второй способ – с использованием собственных значений матрицы Гессе [2, с. 26].

$\lambda_1 > 0, ..., \lambda_n > 0$ – локальный минимум;

$\lambda_1 < 0, ..., \lambda_n < 0$ – локальный максимум;

$\lambda_1 \geq 0, ..., \lambda_n \geq 0$ – возможен локальный минимум, требуется дополнительный анализ;

$\lambda_1 \leq 0, ..., \lambda_n \leq 0$ – возможен локальный максимум, требуется дополнительный анализ;

$\lambda_1 = 0, ..., \lambda_n = 0$ – неопределённость, требуется дополнительный анализ;

λ_i имеют разные знаки – нет экстремумов.

Рассмотренные выше критерии пригодны только для случая непрерывно дифференцируемых функций, когда производная меняет знак в точке локального экстремума (одномерный случай показан на рис. 32). Можно сказать, что для таких функций экстремумы могут быть только в точках. где первая производная обращается в нуль. При этом, если слева и справа от точки, подозрительной на экстремум, производная имеет один и тот же знак, то экстреммума нет.

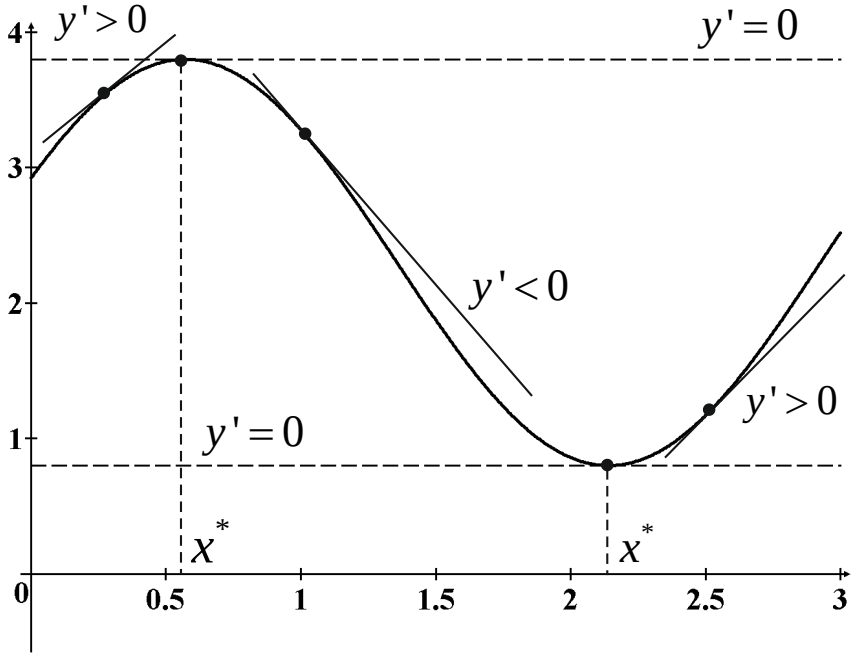


Рис. 32. Непрерывно дифференцируемая функция $y = 3 \cdot \sin(x+1)^2 + 0.8$ с двумя экстремумами x^*. Знак производных меняется в экстремальных точках

Для случая непрерывной функции, дифференцируемой на всём отрезке, за исключением точки, подозрительной на экстремум, (рис. 33, 34) могут быть сформулированы критерии экстремальности. (Для таких функций считается, что они дифференцируемы **в окрестности** точки x^*, но не дифференцируемы в самóй точке x^*).

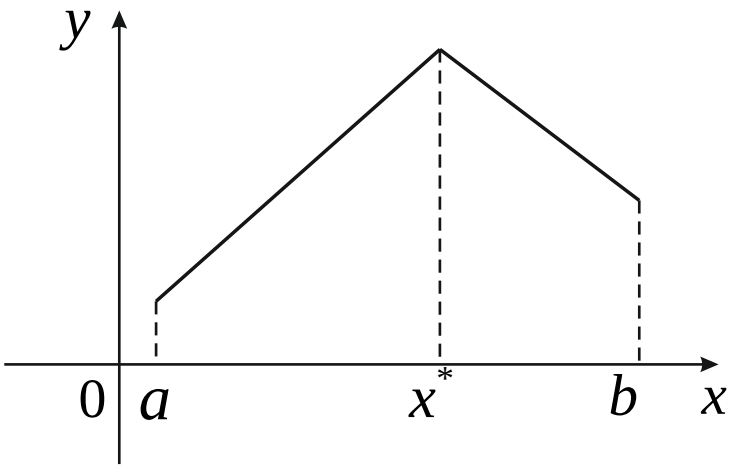


Рис. 33. Непрерывная функция, недифференцируемая в точке максимума x^*

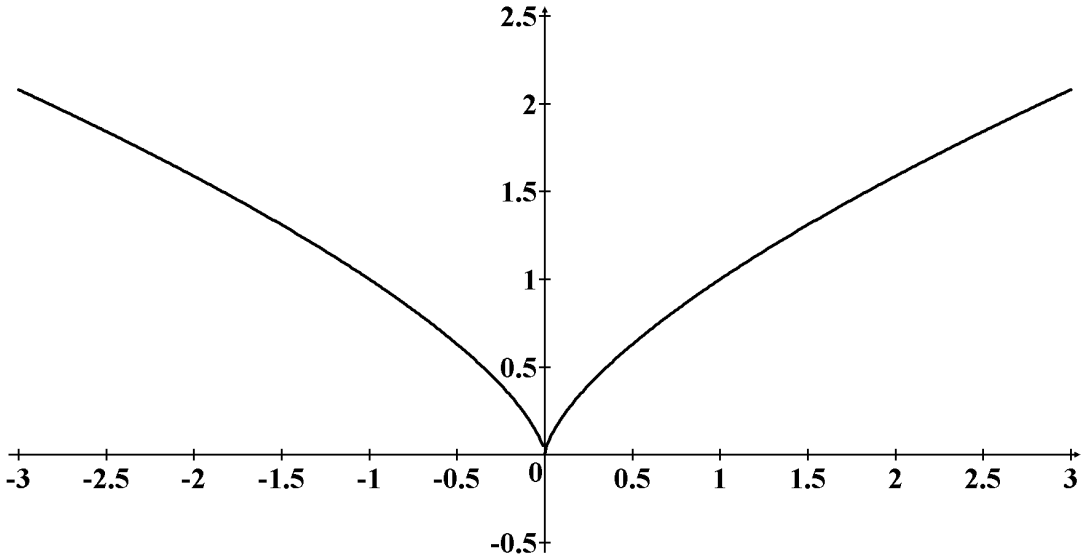


Рис. 34. Непрерывная функция, недифференцируемая в точке минимума $x = 0$ (парабола Нейла $\begin{cases} y = x^{2/3}, & если\ x \geq 0 \\ y = (-x)^{2/3}, & если\ x < 0 \end{cases}$)

Критерий экстремальности для функций, недифференцируемых в точке возможного экстремума [6, с. 19-20].

Если непрерывная функция $f(x)$, не дифференцируема в точке возможного экстремума x^*, но дифференцируема в окрестности этой точки, то:

1) если $f'(x) < 0 \quad \forall x < x^*$ и $f'(x) > 0 \quad \forall x > x^*$, то точка x^* является точкой локального минимума;

2) если $f'(x) > 0 \quad \forall x < x^*$ и $f'(x) < 0 \quad \forall x > x^*$, то точка x^* является точкой локального максимума;

3) если $f'(x)$ имеет один и тот же знак слева и справа от x^*, то экстремума в точке x^* нет.

Как уже отмечалось, к сожалению, применение классического метода решения задачи оптимизации на практике очень ограничено. Это связано, прежде всего, с трудностями дифференцирования функций и нахождения матрицы Гессе. В этом плане исключительную роль играют численные методы оптимизации, которые в настоящее время широко распространены и применяются на практике. Однако многие подходы классического метода сохраняют свою актуальность в плане предварительного математического анализа задач минимизации и установления унимодальности функций.

v10 § 9. Примеры решения задач оптимизации классическим методом

1) Дана функция $y(x_1, x_2) = 2x_1^2 + 2x_2^2$ (рис. 13). Найти экстремум на множестве R^2. Приравниваем нулю частные производные (необходимые условия первого порядка):

$$\frac{\partial y(x_1, x_2)}{\partial x_1} = 4x_1 = 0; \quad \frac{\partial y(x_1, x_2)}{\partial x_2} = 4x_2 = 0;$$

Отсюда находим стационарную точку:

$$x^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Рассчитываем матрицу Гессе:

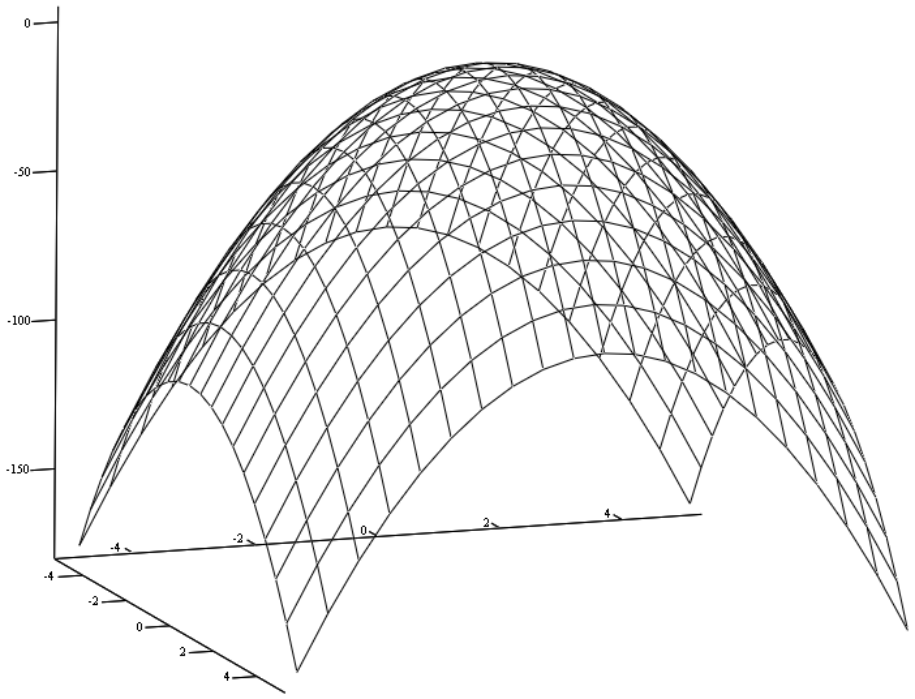
$$H(x^*) = \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}$$

Для проверки достаточных условий рассчитываем угловые миноры (первый способ) и собственные значения (второй способ):

$$\Delta_1 = h_{11} = 4 > 0; \quad \Delta_2 = \begin{vmatrix} 4 & 0 \\ 0 & 4 \end{vmatrix} = 16 > 0.$$

Следовательно, точка x^* – локальный минимум.

$$\begin{vmatrix} 4-\lambda & 0 \\ 0 & 4-\lambda \end{vmatrix} = 0; \qquad (4-\lambda)^2 = 0 \quad \Rightarrow \quad \lambda_1 = \lambda_2 = 4 > 0.$$



f

Рис. 35. Функция $f(x_1, x_2) = -3x_1^2 - 4x_2^2$

Локальный минимум подтверждается. Так как функция $y(x_1, x_2)$ строго выпуклая на множестве R^2, точка x^* – также и глобальный минимум. Значение функции $y(x^*) = 0$.

2) Найти экстремум функции $f(x_1, x_2) = -3x_1^2 - 4x_2^2$ (рис. 35).

Необходимые условия первого порядка:

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = -6x_1 = 0 \quad \frac{\partial f(x_1, x_2)}{\partial x_2} = -8x_2 = 0;$$

Стационарная точка:

$$x^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Матрица Гессе:

$$H(x^*) = \begin{pmatrix} -6 & 0 \\ 0 & -8 \end{pmatrix}$$

Достаточные условия. Угловые миноры (первый способ):

$$\Delta_1 = h_{11} = -6 < 0; \qquad \Delta_2 = \begin{vmatrix} -6 & 0 \\ 0 & -8 \end{vmatrix} = 48 > 0.$$

Точка x^* – локальный максимум.

Собственные значения (второй способ):

$$\begin{vmatrix} -6-\lambda & 0 \\ 0 & -8-\lambda \end{vmatrix} = 0;$$

$$(-6-\lambda)\cdot(-8-\lambda) = 0 \quad \Rightarrow \quad \lambda_1 = -6 < 0; \lambda_2 = -8 < 0.$$

Подтверждение локального максимума. Значение функции $y(x^*) = 0$.

3) Найти экстремум функции $f(x_1, x_2) = 2x_1^2 - 2x_2^2$ – «седловидная» функция (рис. 36).

Необходимые условия первого порядка:

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 2x_1 = 0 \quad \frac{\partial f(x_1, x_2)}{\partial x_2} = -2x_2 = 0;$$

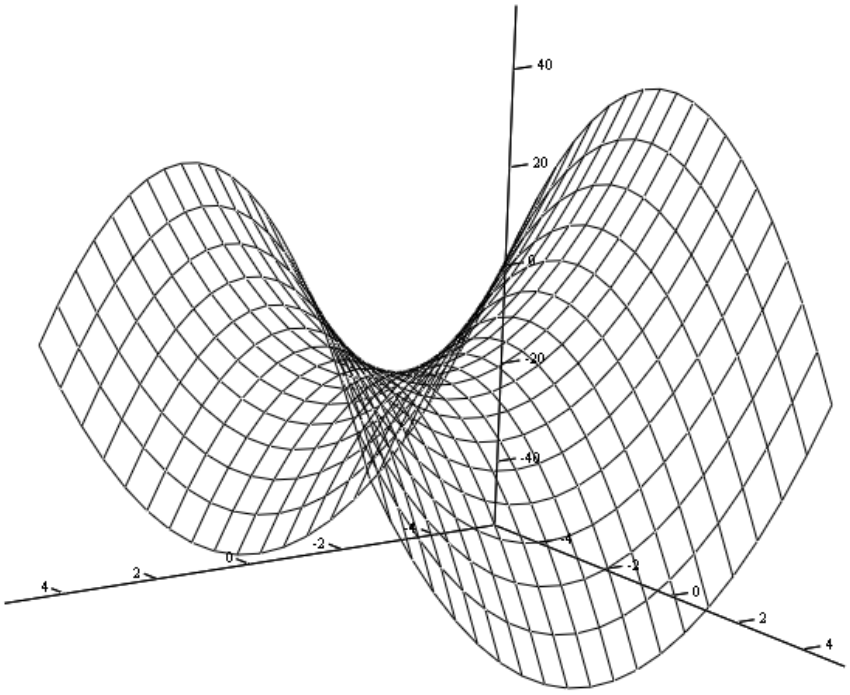
Стационарная точка:

$$x^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Матрица Гессе:

$$H(x^*) = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$$

Достаточные условия. Угловые миноры (первый способ):



f

Рис. 36. Функция $f(x_1, x_2) = 2x_1^2 - 2x_2^2$ – «седловидная»; экстремум отсутствует

$$\Delta_1 = h_{11} = 2 > 0; \qquad \Delta_2 = \begin{vmatrix} 2 & 0 \\ 0 & -2 \end{vmatrix} = -4 < 0.$$

Стационарная точка x^* не является экстремумом.

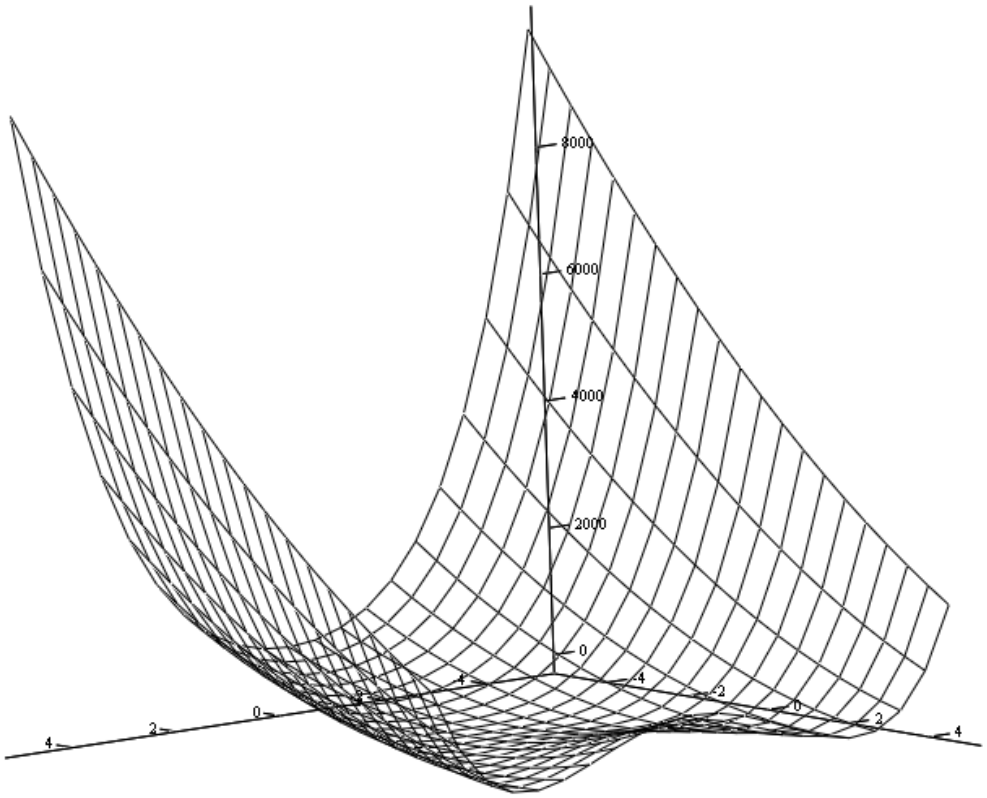
Собственные значения (второй способ):

$$\begin{vmatrix} 2-\lambda & 0 \\ 0 & -2-\lambda \end{vmatrix} = 0;$$

$$(2-\lambda)\cdot(-2-\lambda) = 0 \quad \Rightarrow \quad \lambda_1 = 2 > 0; \lambda_2 = -2 < 0.$$

Подтверждение того, что стационарная точка x^* не является экстремумом. Значение функции в стационарной точке $y(x^*)=0$.

4) Найти экстремум функции $f(x_1, x_2)=2\left(3-x_1\right)^2+8\left(x_1^2-2x_2\right)^2$ (рис. 37).



f

Рис. 37. Функция $f(x_1, x_2)=2\left(3-x_1\right)^2+8\left(x_1^2-2x_2\right)^2$

Необходимые условия первого порядка:

$$\frac{\partial f(x_1, x_2)}{\partial x_1}=-4\left(3-x_1\right)+32\left(x_1^2-2x_2\right)x_1=0;$$

$$\frac{\partial f(x_1,x_2)}{\partial x_2} = -32\left(x_1^2 - 2x_2\right) = 0;$$

Стационарная точка:

$$\begin{cases} -\left(3-x_1\right) + 8\left(x_1^3 - 2x_1x_2\right) = 0 \\ x_1^2 - 2x_2 = 0 \end{cases}$$

$$x_2 = \frac{1}{2}x_1^2; \qquad 8x_1^3 - 8x_1^3 + x_1 - 3 = 0; \qquad x_1 = 3; \qquad x_2 = \frac{9}{2};$$

$$x^* = \begin{pmatrix} 3 \\ 4.5 \end{pmatrix}$$

Вторые частные производные:

$$\frac{\partial^2 f(x_1,x_2)}{\partial x_1^2} = 4\left(1 + 8\left(3x_1^2 - 2x_2\right)\right) = 580;$$

$$\frac{\partial^2 f(x_1,x_2)}{\partial x_2^2} = -32 \cdot \left(-2\right) = 64;$$

$$\frac{\partial^2 f(x_1,x_2)}{\partial x_1 \partial x_2} = 32x_1 \cdot \left(-2\right) = -64x_1 = -192;$$

$$\frac{\partial^2 f(x_1,x_2)}{\partial x_2 \partial x_1} = -32 \cdot \left(2x_1\right) = -64x_1 = -192;$$

Матрица Гессе:

$$H(x^*) = \begin{pmatrix} 580 & -192 \\ -192 & 64 \end{pmatrix}$$

Достаточные условия. Угловые миноры (первый способ):

$$\Delta_1 = h_{11} = 580 > 0; \qquad \Delta_2 = \begin{vmatrix} 580 & -192 \\ -192 & 64 \end{vmatrix} = 256 > 0.$$

Точка x^* – локальный минимум.

Собственные значения (второй способ):

$$\begin{vmatrix} 580-\lambda & -192 \\ -192 & 64-\lambda \end{vmatrix} = 0;$$

$$\left(580-\lambda\right) \cdot \left(64-\lambda\right) - 192^2 = 0;$$

$$-64\lambda + 580 \cdot 64 - 580\lambda + \lambda^2 - 192^2 = 0;$$

$$\lambda^2 - 644\lambda + 256 = 0;$$

$$\lambda_{1,2} = \frac{644}{2} \pm \sqrt{\left(\frac{644}{2}\right)^2 - 256} = 322 \pm 321.6;$$

$$\Rightarrow \quad \lambda_1 = 643.6 > 0; \lambda_2 = 0.4 > 0$$

Подтверждение локального минимума. Значение функции $y(x^*) = 0$.

v11 Глава 2. МЕТОДЫ ОДНОМЕРНОЙ ОПТИМИЗАЦИИ

v12 § 1. Введение. Характеристики численных методов оптимизации

Как уже отмечалось, классический метод решения задач оптимизации применим далеко не всегда. Это связано, в первую очередь, с тем, что очень часто взятие производных функций представляет собой отдельную сложную или вообще не решаемую задачу (в последнем случае – недифференцируемые функции); при математическом анализе условий оптимизации часто приходится решать системы линейных или нелинейных уравнений, а это, само по себе, часто являет собой отдельную нетривиальную задачу. Эти и другие причины делают классический метод ограниченно применимым.

В плане этого исключительная роль принадлежит численным методам, которые позволяют решать одномерные и многомерные задачи оптимизации не только для унимодальных функций, но и, в сочетании с дополнительным анализом, – для недифференцируемых, в некоторых случаях – разрывных, функций, функций, заданных параметрически, а также провести анализ мультимодальных функций (с определением некоторых локальных экстремумов).

Подавляющее большинство таких методов являются приближёнными итерационными. Их реализация состоит в задании требуемой **точности** (чаще всего она обозначается ε), с которой надо получить решение, определения **стратегии поиска**, то есть основного алгоритма построения **минимизирующей последовательности**, которая должна **сходиться** в точке минимума. Выбор класса методов и конкретного метода внутри этого класса диктуется самой задачей и той информацией, которая может быть использована при её решении.

В зависимости от порядка используемых производных методы оптимизации подразделяются на:

1) **методы нулевого порядка**, не требующие информацию о производных;

2) **методы первого порядка**, в которых используется первая производная;

3) **методы второго порядка**, в которых используется вторая производная.

Одна и та же задача оптимизации, вообще говоря, может быть решена разными методами; профессионализм исследователя как раз и состоит в правильном выборе метода, наиболее хорошо подходящего для решения данной конкретной задачи. При этом одним из важнейших факторов является скорость сходимости метода, которая будет определяться, прежде всего, видом целевой функции и значениями соответствующих параметров.

Итерационный процесс решения задачи оптимизации состоит в выстраивании последовательности приближений к решению $\left\{x^{(k)}\right\}$, где k – номер итерации, сходящуюся к экстремуму x^*. Эти приближения будут отличаться друг от друга и никогда не совпадут точно с x^* (за исключением некоторых случаев, например, для целевой функции $f(x)=x^2$). Самым эффективным способом оценки скорости сходимости является сравнение расхождений между соседними приближениями $x^{(k)}$ и $x^{(k+1)}$ с одной стороны, и x^* (за который можно принять конечное решение данной задачи с заданной точностью). В общем случае эти разности будут непостоянными в ходе решения.

Определение. [2, с. 103]. Последовательность $\left\{x^{(k)}\right\}$ называется **сходящейся с порядком** r, если r – максимальное число, для которого

$$0 \le \lim_{k\to\infty} \frac{\left\| x^{(k+1)} - x^* \right\|}{\left\| x^{(k)} - x^* \right\|^r} < \infty \qquad (45)$$

Поскольку итерационный процесс в теории бесконечен, r называется **асимптотической скоростью сходимости**.

Определение. [2, с. 103]. Если последовательность $\left\{x^{(k)}\right\}$ сходится с порядком r, то число

$$c = \lim_{k\to\infty} \frac{\left\| x^{(k+1)} - x^* \right\|}{\left\| x^{(k)} - x^* \right\|^r} \qquad (46)$$

называется **асимптотическим параметром ошибки**.

Определение. [2, с. 103]. Если $r=1, c<1$, то сходимость **линейная**; если $r=2$, то сходимость **квадратичная**, если $r>1$ или $r=1, c=0$, то сходимость **сверхлинейная**.

Определение. [2, с. 110]. **Характеристикой** $R(N)$ **относительного уменьшения начального интервала неопределённости** называется отношение длины интервала, получаемого в результате N вычислений функции, к длине начального интервала неопределённости:

$$R(N)=\frac{|L_N|}{|L_0|} \tag{47}$$

Фактически линейная сходимость имеет скорость геометрической прогрессии, и во многих случаях является вполне удовлетворительной, то есть не требуется обязательно искать для решения методы с квадратичной сходимостью.

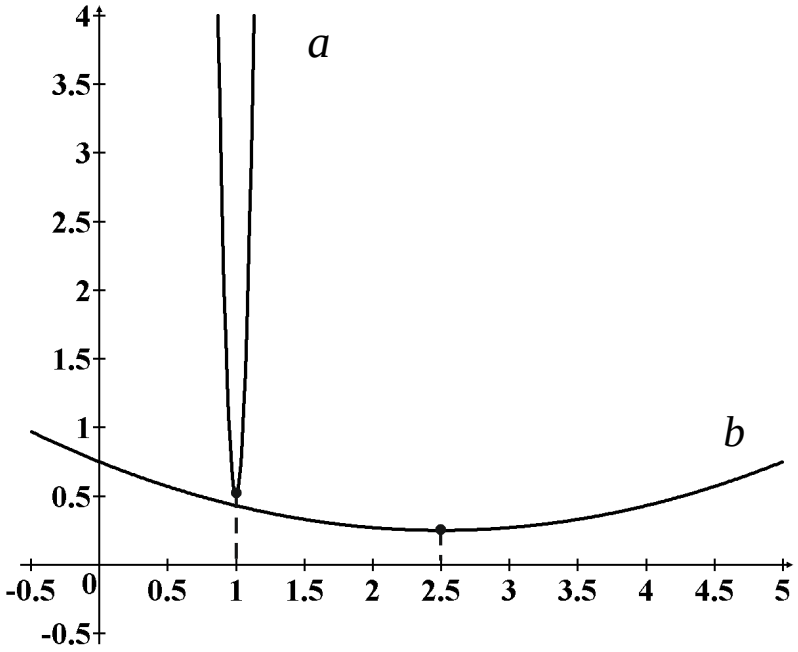


Рис. 38. a – быстрое изменение целевой функции $y=200\cdot(x-1)^2+0.5$; b – пологая целевая функция $y=0.08\cdot(x-2.5)^2+0.25$

Большинство методов одномерной оптимизации применимы лишь для унимодальных функций, поэтому перед их использованием нужно выбрать отрезки числовой оси, где функции унимодальны, то есть содержат только один локальный экстремум. Это может быть сделано путём сканирования функции или построения графиков в широком диапазоне с мелким шагом вычерчивания линии (например, в компьютерных программах MathCad, Advanced Grapher и др.). При этом могут быть ошибки, так как невозможно построить график от $-\infty$ до $+\infty$ или выбрать бесконечно мелкий шаг. Также для выбора интервала неопределённости можно использовать эвристический алгоритм Суонна (см., например, [2, с. 108-109], [7, с. 74]).

Кроме того, на предварительном этапе часто удаётся определить тип локального экстремума – максимум или минимум. Для этого можно использовать некоторые подходы классического метода. Такое определение иногда просто необходимо, так как многие программы нахождения максимума или минимума мало чем отличаются друг от друга, но требуют конкретики.

Также важным является вопрос выбора задаваемой точности ε. Если задача решается в безразмерных координатах, как это по факту имеет место в математике, то возможны два предельных случая, когда в окрестности точки x^* целевая функция может меняться сильно (например резкий провал при минимуме – рис. 38-a) или же, наоборот, очень медленно (пологая функция – рис. 38-b). Тогда, если в первом случае задавать разумную ε по x, то решение по значению целевой функции будет иметь большую неопределённость, а если во втором случае задавать разумную ε по оси ординат (по функции), то большую неопределённость будет иметь решение по аргументу. Выходом из этого может быть задание двух разных точностей по обеим переменным или, при задании одной ε, – выбор критерия окончания по наибольшему изменению одной из переменных. Для реальных физических задач, в которых размерности величин по оси абсцисс и ординат разные, этот вопрос ещё более усложняется.

В § 2 – § 6 рассмотрены методы нулевого порядка, в § 7 – метод первого порядка.

Численные методв одномерной минимизации имеют большое самостоятельное значение, а также широко используются в качестве вспомогательных в задачах многомерной оптимизации.

v13 § 2. Методы сканирования

В этих методах на достаточно большом интервале (отрезке) $[a, b]$ (охватить всю числовую ось от $-\infty$ до $+\infty$ невозможно) на оси абсцисс строится **равномерная сетка**, для узлов которой рассчитываются значения функции. Сетка строится путём задания **числа разбиений** n, при этом получается $n+1$ **узел сетки** и n **интервалов**, имеющих одинаковую длину – **шаг** $h = \frac{b-a}{n}$. Узлы полученной таким образом сетки нумеруются с помощью индекса, например – i, то есть для аргумента x имеем: $x_0 = a, x_1, x_2, ..., x_{i-1}, x_i, ..., x_n = b$, причём значение любого x_i может быть рассчитано по простой формуле: $x_i = a + i \cdot h$. То есть в программной реализации нет необходимости содержать массив аргументов.

Рассчитанные таким образом значения целевой функции представляют собой массив $f(x_i)$, который может быть проанализирован с использованием алгоритмов поиска наименьших и наибольших значений.

При этом может ставиться две задачи. Во-первых, описанный алгоритм позволяет просканировать широкую область аргументов целевой функции и выделить интервалы нахождения локальных экстремумов, которые будут отвечать условиям унимодальности функции. Затем на каждом из этих интервалов может быть решена задача оптимизации соответствующим методом с любой заданной точностью. То есть это – задача предварительного этапа оптимизации. Разумеется, тут возможны ошибки, как уже отмечалось – в плане выбора интервала сканирования и величины шага (числа разбиений). Например, в случае сложных периодических функций возможно наличие узких экстремумов, которые не могут быть обнаружены на заданной сетке, если ширина этих экстремумов будет меньше шага. Как уже отмечалось, рассматриваемую задачу можно решать с помощью построения графиков в имеющихся компьютерных программах (см. предыдущий параграф). Экстремумы при этом можно идентифицировать визуально вручную.

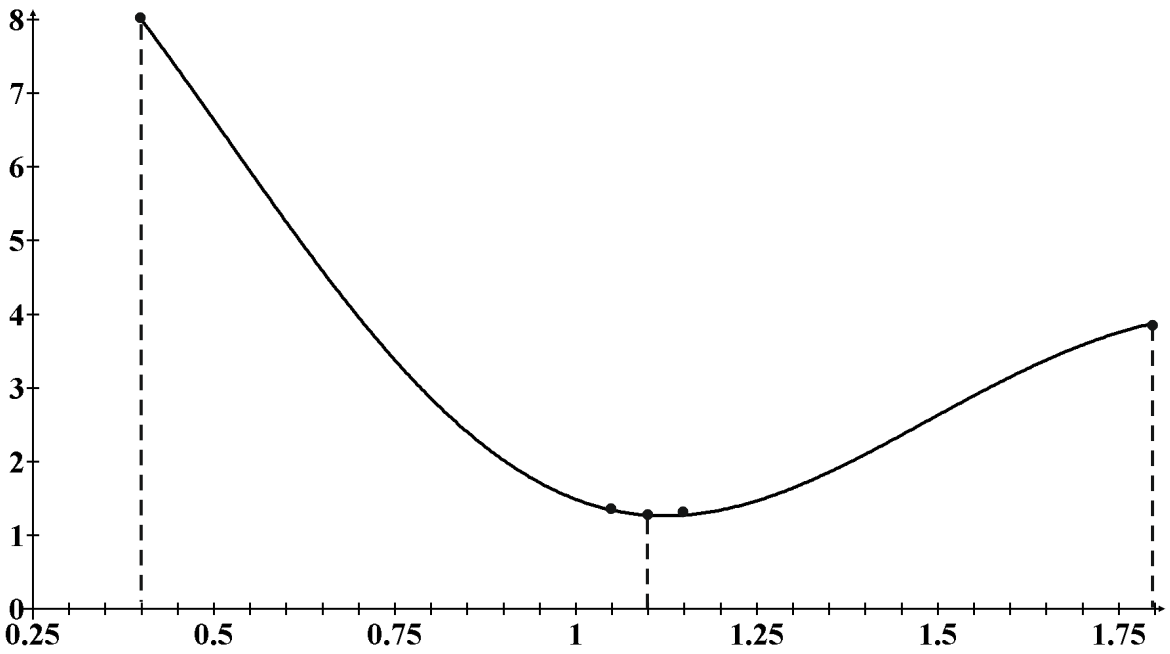


Рис. 39. Минимизация функции $f(x) = 2\sin(4x)/x + 3$ методом перебора. Выбирается наименьшее из всех рассчитанных значений

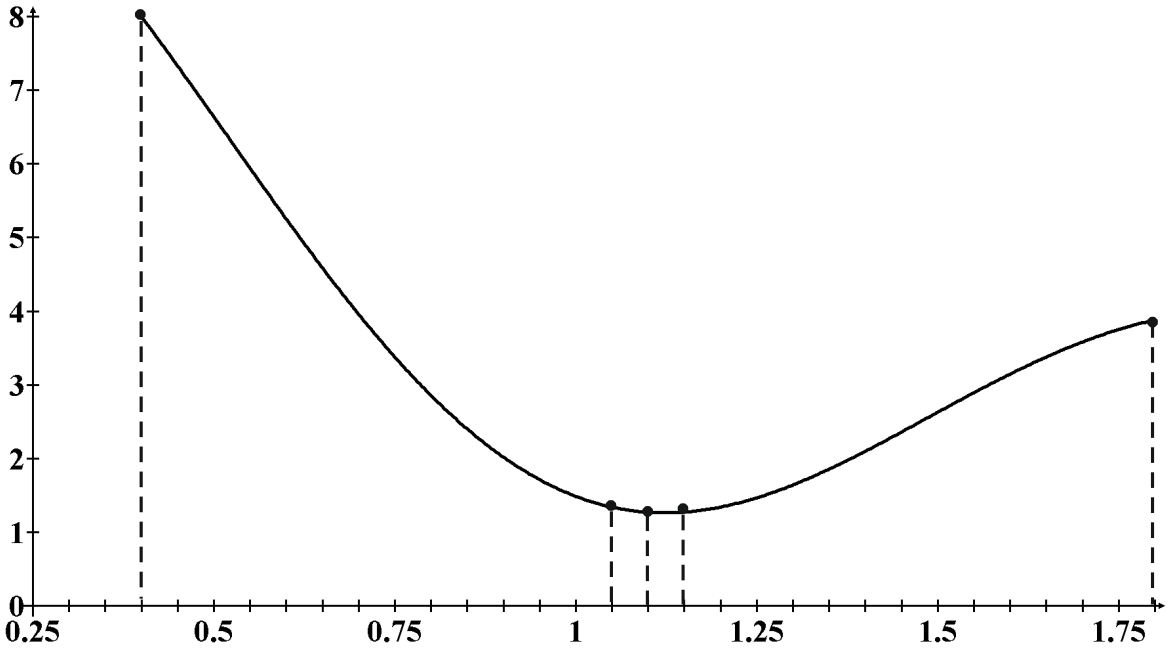


Рис. 40. Минимизация функции $f(x) = 2\sin(4x)/x + 3$ методом равномерного поиска. Выбирается малый интервал, содержащий минимум

Вторая задача может представлять собой непосредственно нахождение экстремума на заданном интервале, на котором целевая функция унимодальна. При минимизации находится либо просто минимальное значение функции в каком-то узле (**метод перебора**) – рис. 39 , либо определяется интервал из двух смежных сегментов (длиной $2h$), в котором точно содержится минимум (**метод равномерного поиска**) – рис. 40 . Если при этом длина $2h$ будет меньше требуемой точности ε, то задачу можно считать решённой. В противном случае можно подобрать более совершенный метод и решить задачу на найденном интервале с любой точностью.

Необходимо отметить, что оба метода используют так называемую **пассивную стратегию поиска**, когда вычисления проводятся в априорно заданных узлах. (В противовес этому, методы, в которых следующие точки выбираются с учётом результатов предыдущих вычислений, относятся к использующим **последовательную стратегию**).

Относительное уменьшение начального интервала неопределённости (то есть $[a, b]$) для метода равномерного поиска вычисляется по формуле:

$$R(N) = \frac{2}{N+1}, \qquad (48)$$

где N – количество вычислений функции [2, с. 111].

В случае унимодальности функции метод равномерного поиска может быть несколько модифицирован. Если имеется только один экстремум, нет необходимости рассчитывать значения функции во всех точках сетки. Можно организовать перебор, например, слева направо (начиная с точки a) и отслеживать поведение функции в нескольких последних точках. Если наблюдается рост функции, речь может идти о поиске максимума, если – спад, – то минимума. Если в следующей рассчитанной точке рост сменится спадом (или спад – ростом), то можно констатировать, что последние два сегмента содержат экстремум, и вычисления можно прекратить. Фактически при этом будет осуществлена последовательная стратегия.

На рис. 41 показана функция $f(x) = 2\sin(4x)/x;\ \ x \neq 0$ на отрезке $[-0.5, 5]$, имеющая несколько экстремумов. В точке $x = 0$ функция терпит разрыв I рода. Ниже приведена программа на языке

Lazarus для консольного приложения, которая по методу сканирования определяет экстремумы функции и их вид (здесь и далее программы с небольшими изменениями подходят и для языка Delphi). Для оптимизации другой функции выражение для целевой функции должно быть заменено (по синтаксису языка).

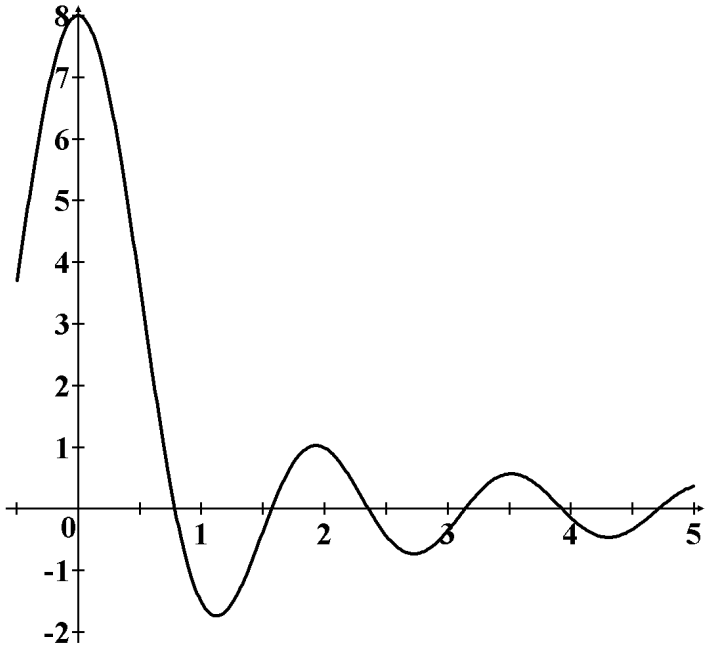


Рис. 41. Функция $f(x) = 2\sin(4x)/x; \quad x \neq 0$ в диапазоне от –0,5 до 5. В точке $x = 0$ функция терпит разрыв I рода

Программа сканирования

```
program Scan1;
var a,b,h,y1,y2: real;// описание концов диапазона a и b,
// шага h, предыдущего и последующего значения
//функции y1 и y2
i,n,k:integer;// описание индекса (номера шага) i,
// числа разбиений n и метки вида экстремума k
//k=1 – min, k=2 – max
function f(x:real):real;// описание целевой функции
begin
if x<>0 then f:=2*sin(4*x)/x else f:=8;
// в точке разрыва x=0 функция возвращает значение 8 –
// – предел при стремлении x → 0
```

```
end;
begin // нвчало основной программы
write('a=');readln(a);// ввод левого конца отрезка
write('b=');readln(b);// ввод правого конца отрезка
write('n=');readln(n);// ввод числа разбиений
h:=(b-a)/n;// расчёт шага
y1:=f(a);// расчёт предыдущего значения функции
y2:=f(a+h);// расчёт последующего значения функции
if y2<y1 then k:=1 else k:=2;// установление метки экстремума,
// если функция возрастает, ищется максимум,
// если убывает – минимум
for i:=2 to n do begin
// перебор всех значений функции, начиная с третьего
// (индекс i = 2) до конца
y1:=y2;// переприсваивание последующего значения функции
// предыдущему в начале цикла обработки следующего шага
y2:=f(a+i*h);// расчёт нового значения y2
if (k=1)and(y2>y1)then begin writeln('Local minimum from x=',a+(i-
2)*h,' to x=',a+i*h,' f(x)=',y1);k:=2;end;
// анализ поведения функции: если при поиске минимума
// значение функции начинает возрастать (y2>y1), то минимум
// пройден, о чём выводится соответствующая информация
if (k=2)and(y2<y1)then begin writeln('Local maximum from x=',a+(i-
2)*h,' to x=',a+i*h,' f(x)=',y1);k:=1;end;
// анализ поведения функции: если при поиске максимума
// значение функции начинает убывать (y2<y1), то максимум
// пройден, о чём выводится соответствующая информация
end;// конец цикла сканирования
readln;// задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

```
a=-0.5
b=5
```

n=10000
Local maximum from x=-5.9999999999998943E-004
to x= 5.0000000000005596E-004
f(x)= 7.9999999466666676E+000
Local minimum from x= 1.1230500000000001E+000
to x= 1.1241500000000002E+000
f(x)=-1.7378681732394781E+000
Local maximum from x= 1.9310000000000000E+000
to x= 1.9321000000000002E+000
f(x)= 1.0269959666033273E+000
Local minimum from x= 2.7257500000000001E+000
to x= 2.7268500000000002E+000
f(x)=-7.3060119783400457E-001
Local maximum from x= 3.5161000000000007E+000
to x= 3.5171999999999999E+000
f(x)= 5.6730762882815466E-001
Local minimum from x= 4.3048000000000002E+000
to x= 4.3059000000000003E+000
f(x)=-4.6377432238209060E-001

(Напомним, что в синтаксисе Lazarus'а выражение E<число> означает $10^{<число>}$, то есть, например, –4.6377E-001 означает $-4.6377 \cdot 10^{-1}$). Таким образом, все локальные экстремумы корректно определены.

На рис. 42 показана функция $f(x) = 2\sin(4x)/x;\quad x \neq 0$, унимодальная на отрезке $[0, 0.75]$, имеющая минимум. (В точке $x = 0$ функция терпит разрыв I рода). Ниже приведена программа на языке Lazarus для консольного приложения, которая по методу равномерного поиска с остановкой перебора определяет экстремум непрерывной унимодальной функции и его вид.

Программа метода равномерного поиска

```
program scan2;
var a,b,h,y1,y2: real;// описание концов диапазона a и b,
// шага h, предыдущего и последующего значения
//функции y1 и y2
```

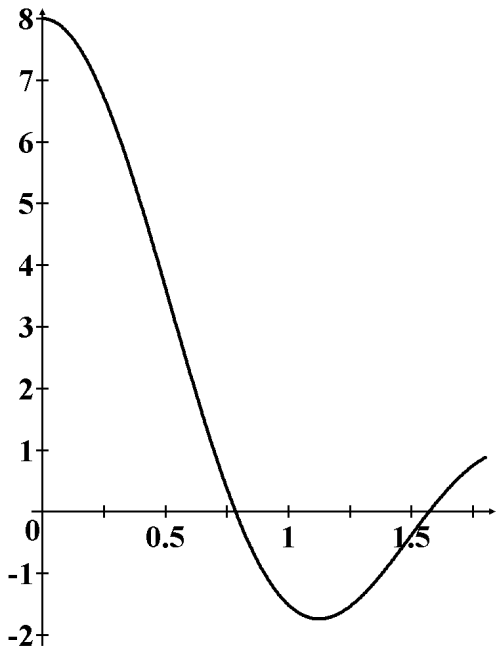


Рис. 42. Функция $f(x) = 2\sin(4x)/x; \quad x \neq 0$ в диапазоне от 0 до 1,75. В точке $x = 0$ функция терпит разрыв I рода

```
i,n:integer;// описание индекса (номера шага) i,
// и числа разбиений n
function f(x:real):real;// описание целевой функции
begin
if x<>0 then f:=2*sin(4*x)/x else f:=8;
// в точке разрыва x=0 функция возвращает значение 8 –
// – предел при стремлении x → 0
end;
begin // нвчало основной программы
write('a=');readln(a);// ввод левого конца отрезка
write('b=');readln(b);// ввод правого конца отрезка
write('n=');readln(n);// ввод числа разбиений
h:=(b-a)/n;// расчёт шага
y1:=f(a);// расчёт предыдущего значения функции
y2:=f(a+h);// расчёт последующего значения функции
// если функция в начале интервала убывает, то есть y2<y1,
// ведётся поиск минимума, иначе – ведётся поиск максимума:
if y2<y1 then
for i:=2 to n do begin
```

```
// перебор всех значений функции, начиная с третьего
// (индекс i = 2) до конца
y1:=y2; ;// переприсваивание последующего значения функции
// предыдущему в начале цикла обработки следующего шага
y2:=f(a+i*h);// расчёт нового значения y2
// анализ поведения функции: если при поиске минимума
// значение функции начинает возрастать (y2>y1), то минимум
// пройден, о чём выводится соответствующая информация
if y2>y1 then begin writeln('Minimum from x=',a+(i-2)*h,' to x=',a+i*h);
writeln('f(x)=',y1);// вывод значения функции в минимуме
writeln('number of steps ',i,' from ',n);// вывод количества
// произведённых шагов и общего числа узлов сетки
break;// досрочный выход из цикла
end;
end
else // аналогично для случая y2>y1
for i:=2 to n do begin
y1:=y2;
y2:=f(a+i*h);
if y2<y1 then begin writeln('Maximum from x=',a+(i-2)*h,' to x=',a+i*h);
writeln('f(x)=',y1);
writeln('number of steps ',i,' from ',n);
break;
end;
end;// завершение внешнего оператора if
readln;// задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

```
a=0.25
b=1.75
n=10000
```

Minimum from x= 1.1231499999999999E+000 to x= 1.1234500000000001E+000
f(x)=-1.7378689875661586E+000
number of steps 5823 from 10000

v14 § 3. Метод дихотомии (деления отрезка пополам)

В методе дихотомии исходный отрезок $[a,b]$, на котором функция унимодальна, делится пополам и от полученной середины влево и вправо откладываются отрезки малой длины δ, которая, как правило, сравнима с задаваемой точностью ε – рис. 43. Таким образом, в середине исходного интервала получаются две точки $x_1 = \frac{a+b}{2} - \delta$ и $x_2 = \frac{a+b}{2} + \delta$, в которых вычисляются значения функции $y_1 = f(x_1)$ и $y_2 = f(x_2)$.

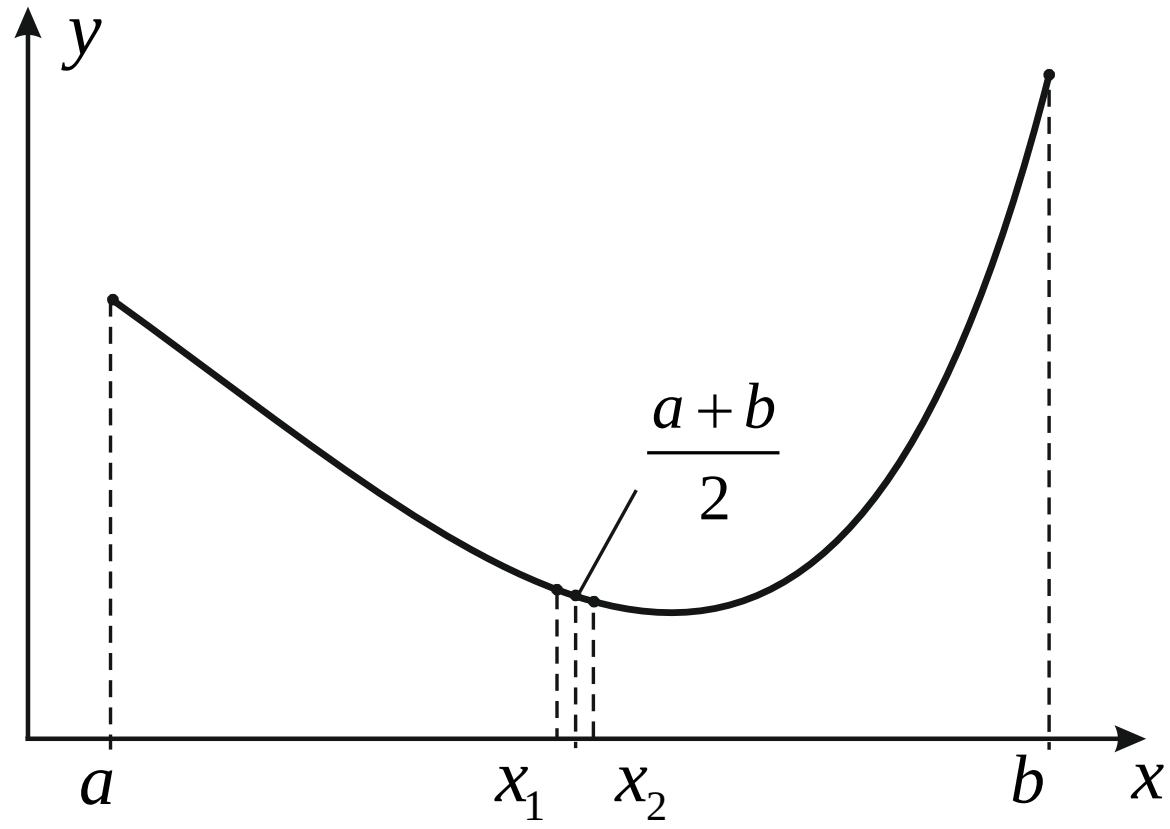


Рис. 43. Принцип метода дихотомии, x_1 и x_2 отстоят от середины на δ

При этом в задаче минимизации возможны два варианта (рис. 44): если $y_1 > y_2$, то возможный ход унимодальной функции может быть только таким, как показано на рис. 44-a (качественно), если же $y_1 < y_2$, то возможен только вид рис. 44-b. Таким образом, мы получаем критерий нахождения минимума либо в левой половине исходного отрезка, либо в правой, и можно отбросить соответствующую часть исходного отрезка, уменьшив тем самым интервал неопределённости. Итерации с уменьшением интервала неопределённости можно продолжать до тех пор, пока этот интервал не станет меньше заданной точности. Недостатком метода является постоянство величины δ; если она маленькая, сравнимая или меньше точности ε, то возможны большие погрешности при сравнении вычисляемых функций y_1 и y_2.

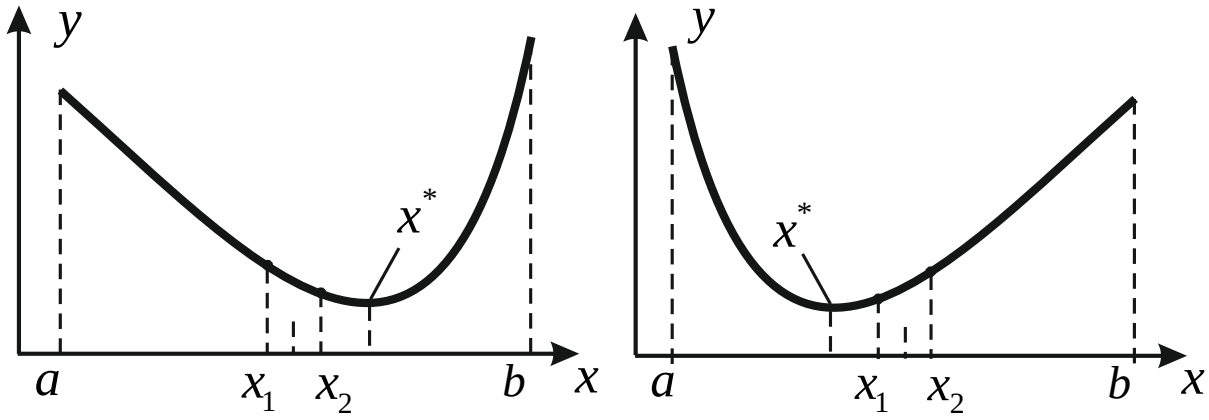


Рис. 43. a – При $y_1 > y_2$ функция имеет минимум в правой половине исходного отрезка; b – при $y_1 < y_2$ – в левой

Ниже приведена программа на языке Lazarus для консольного приложения, которая по методу дихотомии минимизирует функцию $f(x) = 2\sin(4x)/x + 3$ на интервале унимодальности $[0.4, 1.8]$ (рис. 39). При использовании программы для других случаев нужно заменить целевую функцию.

Программа метода дихотомии

```
program Dihotomia;
```

```
var a,b,dt,eps: real; // описание концов диапазона a и b,
// малого числа dt и точности eps
function f(x:real):real; // описание целевой функции
begin
if x<>0 then f:=2*sin(4*x)/x+3 else f:=8;
// в точке разрыва x=0 функция возвращает значение 8 –
// – предел при стремлении x → 0
end;
begin // нвчало основной программы
write('a=');readln(a);// ввод левого конца отрезка
write('b=');readln(b);// ввод правого конца отрезка
write('dt=');readln(dt);// ввод малого числа
write('eps=');readln(eps);// ввод точности
if (f((a+b)/2)<f(a)) or (f((a+b)/2)<f(b)) then
// установление вида экстремума; при выполнении этого условия
// имеет место минимум
begin // начало минимизации
repeat // открытие итерационного цикла
// проверка критерия сужения интервала неопределённости:
if f((a+b)/2-dt)<f((a+b)/2+dt)
then b:=(a+b)/2 // отбрасывание правого отрезка
else a:=(a+b)/2; // отбрасывание левого отрезка
until abs(b-a)<eps; // критерий выхода из цикла
// вывод результатов – значения функции и аргумента:
writeln('Minimum f(x)=',f((a+b)/2),' for x=',(a+b)/2);
end // конец минимизации
else
// при выполнении обратного условия имеет место максимум
begin// начало поиска максимума
repeat// открытие итерационного цикла
// проверка критерия сужения интервала неопределённости:
if f((a+b)/2-dt)<f((a+b)/2+dt)
then a:=(a+b)/2 // отбрасывание левого отрезка
else b:=(a+b)/2; // отбрасывание правого отрезка
until abs(b-a)<eps; // критерий выхода из цикла
// вывод результатов – значения функции и аргумента:
writeln('Maximum f(x)=',f((a+b)/2),' for x=',(a+b)/2);
```

```
end; // конец поиска максимума
readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

```
a=0.4
b=1.8
dt=0.00001
eps=0.00004
Minimum f(x)= 1.2621309750543019E+000
for x= 1.1233596801757815E+000
```

Та же функция имеет максимум на отрезке [1.5, 2.5] (см. похожую функцию рис. 41). Результат работы программы для этого интервала:

```
a=1.5
b=2.5
dt=0.00001
eps=0.00004
Maximum f(x)= 4.0269964277775792E+000
for x= 1.9313201904296875E+000
```

В методе дихотомии относительное уменьшение начального интервала неопределённости равно:

$$R(N) = \frac{1}{2^{N/2}}, \qquad (49)$$

где N – количество вычислений функции [2, с. 118].

v15 § 4. Метод деления отрезка на четыре части

(Метод описан в [2, с. 112 – 116] как метод деления интервала пополам). В этом методе исходный отрезок делится на четыре равные части точками $x_1 = \frac{3a+b}{4}$, $x_2 = \frac{a+b}{2}$ и $x_3 = \frac{a+3b}{4}$, которым будут

соответствовать значения целевой функции $y_1 = f\left(\frac{3a+b}{4}\right)$, $y_2 = f\left(\frac{a+b}{2}\right)$ и $y_3 = f\left(\frac{a+3b}{4}\right)$. При этом возможны следующие варианты прохождения унимодальных функций (качественно), в зависимости от взаимного расположения значений y_1, y_2 и y_3 – рис. 44 – 47.

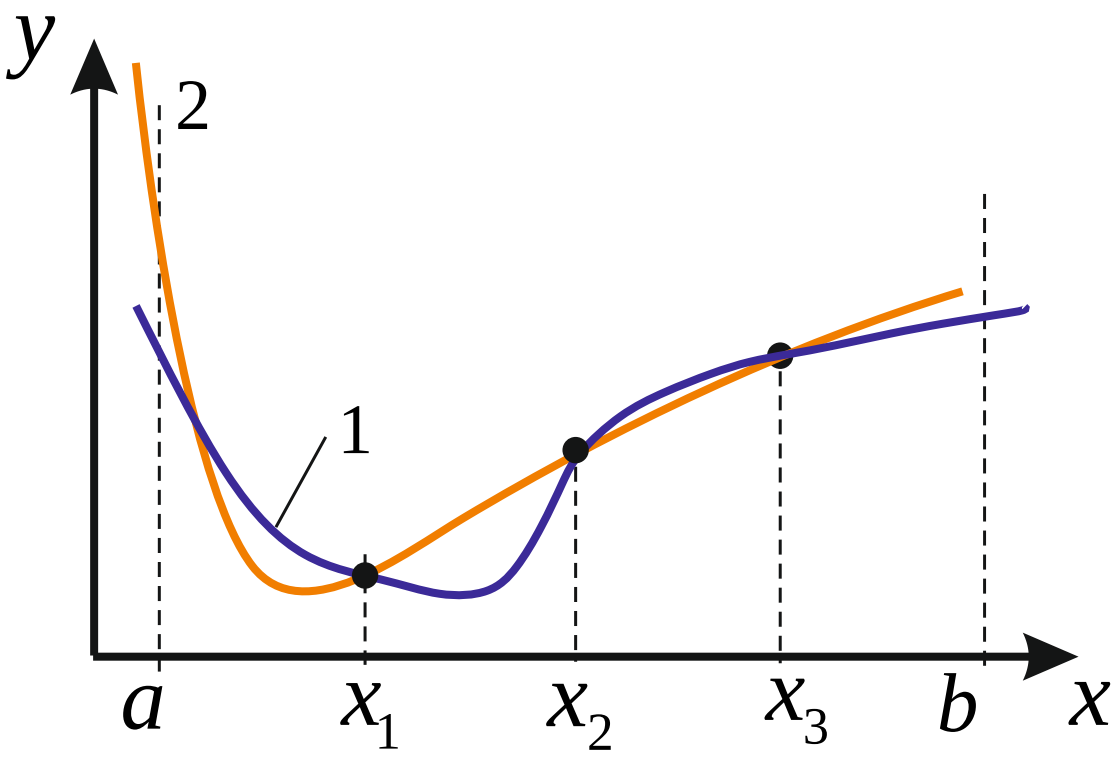


Рис. 44. Возможный вид унимодальных функций при $y_1 < y_2 < y_3$. Минимумы могут находиться на $[a, x_1]$ (2) или на $[x_1, x_2]$ (1)

1) $y_1 < y_2 < y_3$ (рис. 44). Если функции унимодальны, они могут иметь минимум или на отрезке $[a, x_1]$ или на отрезке $[x_1, x_2]$. Другие положения минимума будут нарушать унимодальность. Следовательно, можно отбросить отрезок $[x_2, b]$ посредством присваивания $b := x_2$. Перед заходом на следующую итерацию необходимо сделать переприсваивание $y_2 := y_1$ и вычислить новые значения y_1 и y_3 в новых точках x_1 и x_3.

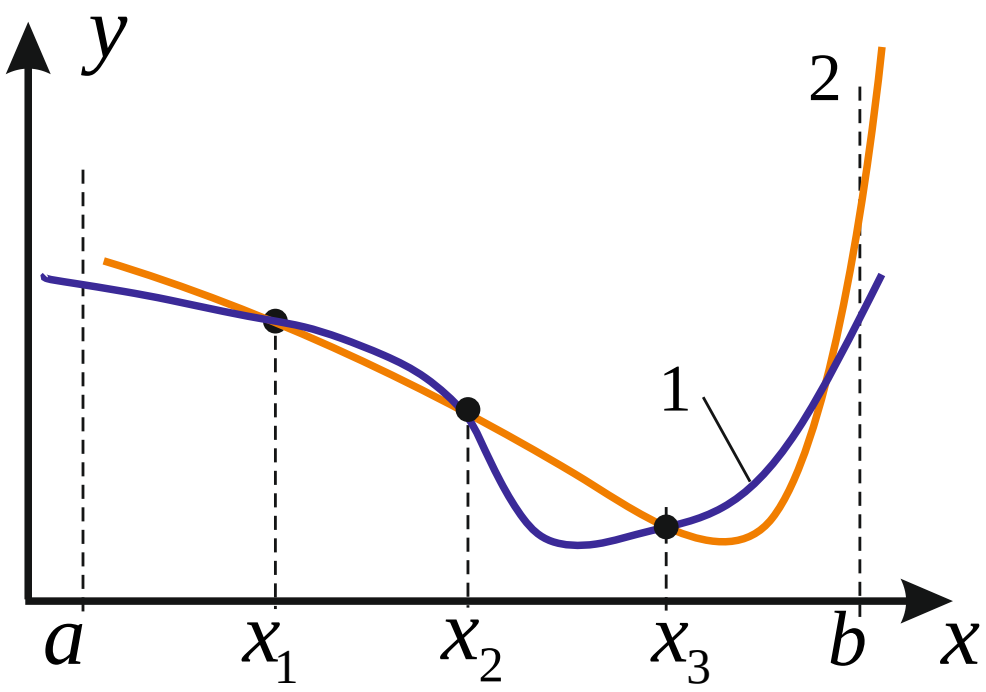


Рис. 45. Возможный вид унимодальных функций при $y_1 > y_2 > y_3$.
Минимумы могут находиться на $[x_2, x_3]$ (1) или на $[x_3, b]$ (2)

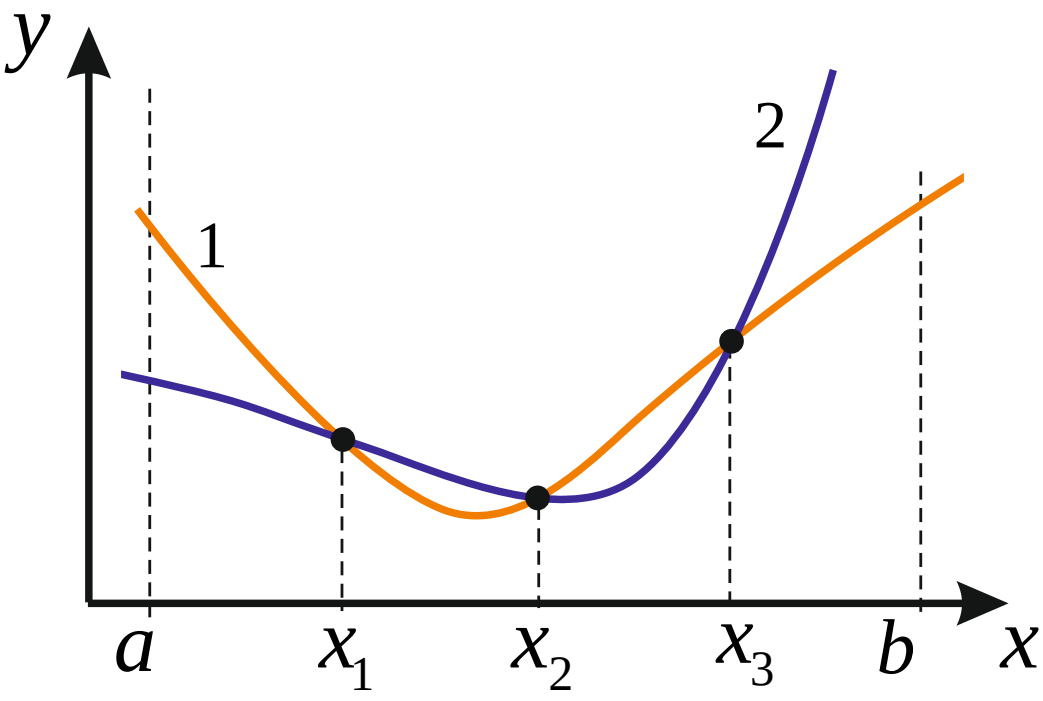


Рис. 46. Возможный вид унимодальных функций при $y_2 < y_1 < y_3$.
Минимумы могут находиться на $[x_1, x_2]$ (1) или на $[x_2, x_3]$ (2)

2) $y_1 > y_2 > y_3$ (рис. 45). Аналогичная, но зеркальная ситуация. Минимумы могут находиться на отрезках $[x_2, x_3]$ и $[x_3, b]$. Отбрасывается отрезок $[a, x_2]$. Переприсваивания: $a := x_2$, $y_2 := y_3$. Новые значения функции y_1 и y_3 также вычисляются в новых точках x_1 и x_3.

3) $y_2 < y_1 < y_3$ (рис. 46). Минимумы могут находиться только на двух средних отрезках, поэтому отбрасывать следует отрезки $[a, x_1]$ и $[x_3, b]$ (их общая длина составляет половину исходного отрезка $[a, b]$, как и в первых двух случаях). Для этого следует сделать присваивания: $a := x_1$, $b := x_3$, и вычислить новые значения функции y_1 и y_3 в новых точках x_1 и x_3 (значение y_2 сохраняется).

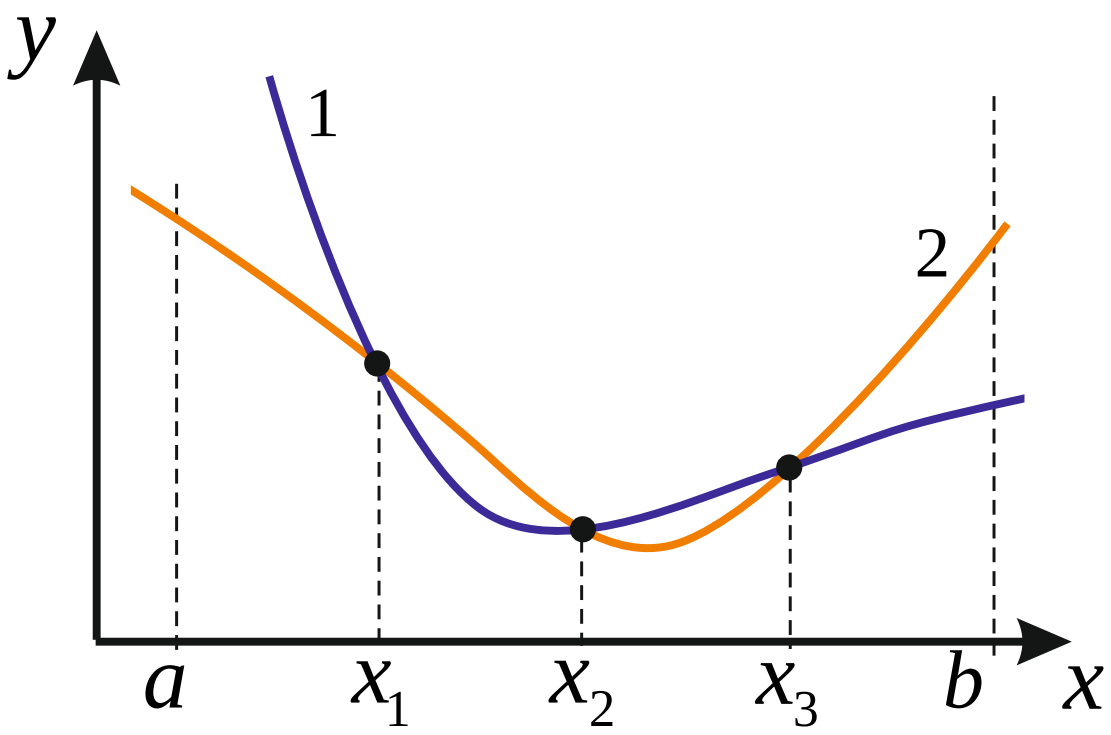


Рис. 47. Возможный вид унимодальных функций при $y_2 < y_3 < y_1$. Минимумы могут находиться на $[x_1, x_2]$ (1) или на $[x_2, x_3]$ (2)

4) $y_2 < y_3 < y_1$ (рис. 47). Ситуация по своим выводам совершенно аналогична пункту 3).

Ниже представлена программа на языке Lazarus (консольное приложение) для нахождения минимума или максимума (в последнем

случае условия 1) – 4) скорректированы соответствующим образом). Для апробации программы использовались функции $f(x)=2\sin(4x)/x+3;\quad x\neq 0$ (рис. 39, 40; см. также рис. 41, 42), локон Аньези (точнее – агвинея Ньютона) $f(x)=3/(2+(x+4)^2)+0.5$ и её же форма с минусом $f(x)=-3/(2+(x-2)^2)+0.5$ – рис. 48, распределение Гаусса (нормальное распределение) $f(x)=1.5\cdot\exp(-(x-4)^2)$ и её же форма с минусом $f(x)=-2\cdot\exp(-(x+3)^2)-0.5$ – рис. 49.

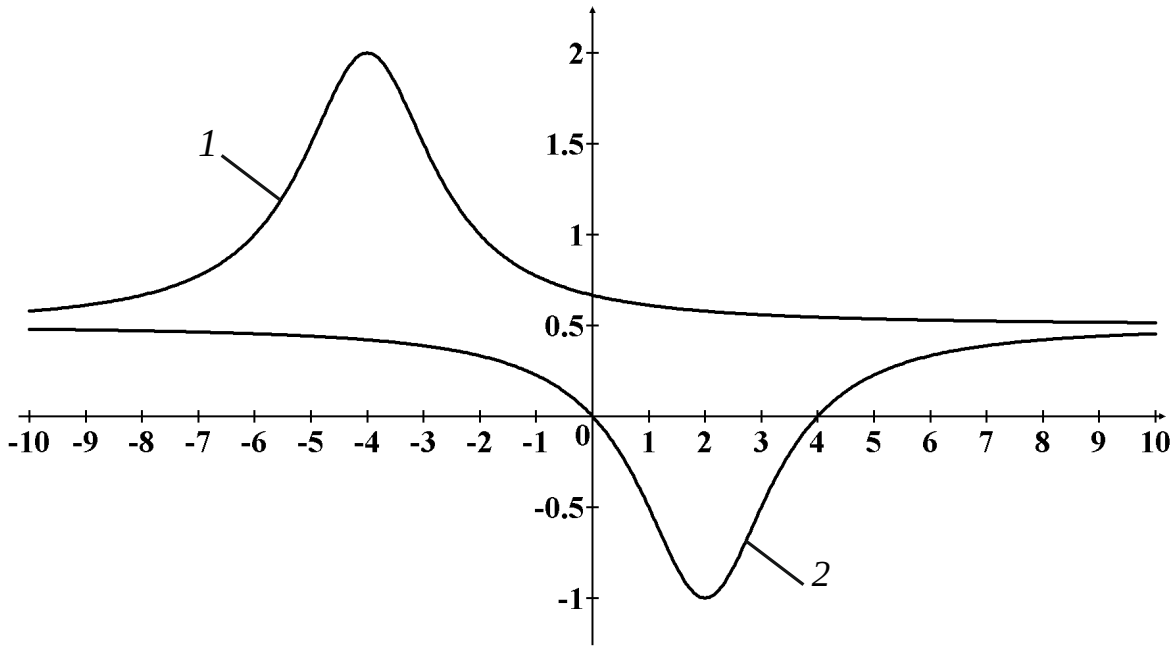


Рис. 48. 1 – функция локон Аньези $f(x)=3/(2+(x+4)^2)+0.5$;
2 – функция локон Аньези с обратным знаком
$f(x)=-3/(2+(x-2)^2)+0.5$

Программа метода деления отрезка на четыре части

```
program TripleDivision;
var a,b,y1,y2,y3,eps: real; // описание концов диапазона a и b,
// функций y1, y2, y3 в трёх внутренних точках x1, x2, x3
// и точности eps
//Далее описываются пять функций, по которым проводилась
```

// апробация, четыре из которых закомментированы скобками

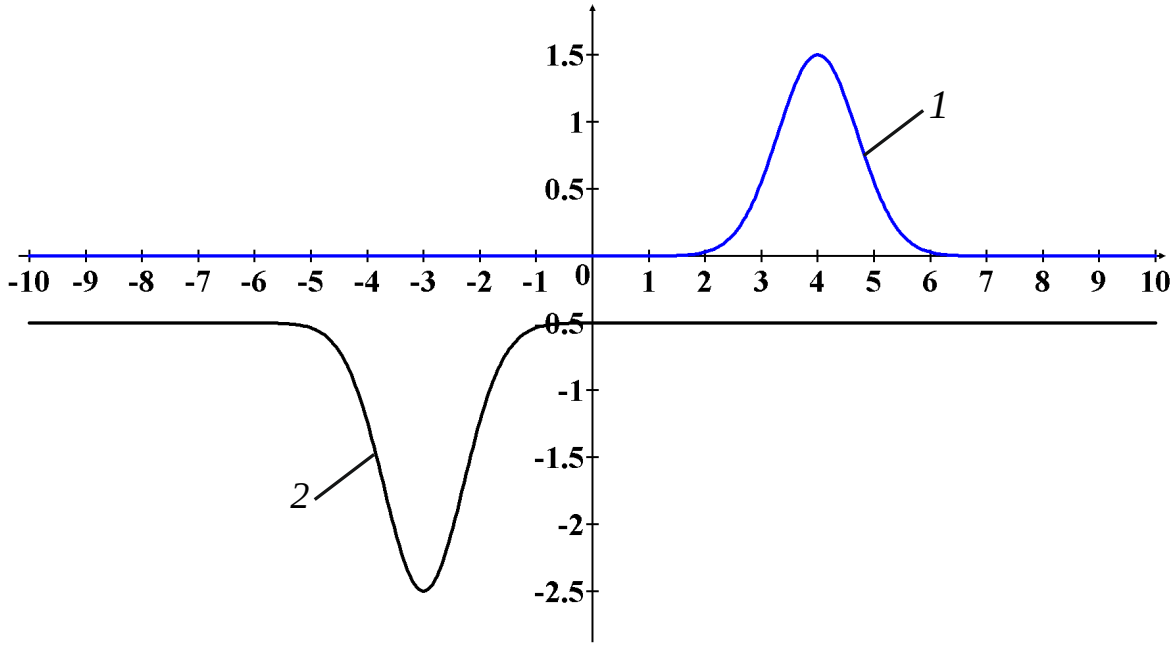


Рис. 49. 1 – функция распределение Гаусса $f(x) = 1.5 \cdot \exp(-(x-4)^2)$

2 – функция распределение Гаусса с обратным знаком

$$f(x) = -2 \cdot \exp(-(x+3)^2) - 0.5$$

```
// { и }, см. текст
{function f(x:real):real;
begin
if x<>0 then f:=2*sin(4*x)/x+3 else f:=8;
end;}
{function f(x:real):real;
begin
f:=3/(2+sqr(x+4))+0.5
end; }
{function f(x:real):real;
begin
f:=-3/(2+sqr(x-2))+0.5
end;}
{function f(x:real):real;
begin
```

```
f:=1.5*exp(-sqr(x-4))
end;}
function f(x:real):real;
begin
f:=-2*exp(-sqr(x+3))-0.5
end;
begin // начало основной программы
write('a=');readln(a); // ввод левого конца отрезка
write('b=');readln(b); // ввод правого конца отрезка
write('eps=');readln(eps); // ввод точности
y1:=f((3*a+b)/4); // расчёт функции в левой точке
y2:=f((a+b)/2); // расчёт функции в средней точке
y3:=f((a+3*b)/4); // расчёт функции в правой точке
if (y2<f(a)) or (y2<f(b)) then
// установление вида экстремума; при выполнении этого условия
// имеет место минимум
begin // начало минимизации
repeat // открытие итерационного цикла
if(y1<y2)and(y2<y3) // выполнение условия y1 < y2 < y3 (рис. 44)
then begin // необходимые действия согласно алгоритму:
b:=(a+b)/2; // отбрасывание двух правых отрезков
y2:=y1;
y1:=f((3*a+b)/4); // расчёт нового значения функции
y3:=f((a+3*b)/4); // расчёт нового значения функции
end
else
if(y1>y2)and(y2>y3) // выполнение условия y1 > y2 > y3 (рис. 45)
then
begin // необходимые действия согласно алгоритму:
a:=(a+b)/2; // отбрасывание двух левых отрезков
y2:=y3;
y1:=f((3*a+b)/4); // расчёт нового значения функции
y3:=f((a+3*b)/4); // расчёт нового значения функции
end
else // иначе – выполнение условий y2 < y1 < y3 (рис. 46)
// или y2 < y3 < y1 (рис. 47)
```

```
begin // необходимые действия согласно алгоритму:
// отбрасывание двух крайних малых отрезков:
a:=(3*a+b)/4;
b:=(a+3*b)/4;
y1:=f((3*a+b)/4); // расчёт нового значения функции
y3:=f((a+3*b)/4); // расчёт нового значения функции
end
until abs(b-a)<eps; // критерий выхода из цикла
// вывод результатов – значения функции и аргумента:
writeln('Minimum f(x)=',f((a+b)/2),' for x=',(a+b)/2);
end // конец минимизации
else
// при выполнении обратного условия имеет место максимум
begin // начало поиска максимума
repeat // открытие итерационного цикла
if(y1>y2)and(y2>y3)
then begin // необходимые действия согласно алгоритму:
b:=(a+b)/2; // отбрасывание двух правых отрезков
y2:=y1;
y1:=f((3*a+b)/4); // расчёт нового значения функции
y3:=f((a+3*b)/4); // расчёт нового значения функции
end
else
if(y1<y2)and(y2<y3)
then
begin // необходимые действия согласно алгоритму:
a:=(a+b)/2; // отбрасывание двух левых отрезков
y2:=y3;
y1:=f((3*a+b)/4); // расчёт нового значения функции
y3:=f((a+3*b)/4); // расчёт нового значения функции
end
else // иначе – выполнение обратных условий
begin // необходимые действия согласно алгоритму:
// отбрасывание двух крайних малых отрезков:
a:=(3*a+b)/4;
b:=(a+3*b)/4;
y1:=f((3*a+b)/4); // расчёт нового значения функции
```

```
y3:=f((a+3*b)/4); // расчёт нового значения функции
end
until abs(b-a)<eps; // критерий выхода из цикла
// вывод результатов – значения функции и аргумента:
writeln('Maximum f(x)=',f((a+b)/2),' for x=',(a+b)/2);
end; // конец поиска максимума
readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

1) Функция $f(x) = 2\sin(4x)/x;\ \ x \neq 0$ для двух разных интервалов; см. похожие функции рис. 41 и 42.

```
a=0.4
b=1.8
eps=0.0004
Minimum f(x)= 1.2621309744914821E+000
for x= 1.1233487537770996E+000

a=1.5
b=2.5
eps=0.00004
Maximum f(x)= 4.0269964281787756E+000
for x= 1.9313148190308311E+000
```

2) Функция $f(x) = 3/(2+(x+4)^2) + 0.5$ – рис. 48-1.

```
a=-10
b=10
eps=0.000001
Maximum f(x)= 1.9899631882967483E+000
for x=-3.8839286283135595E+000
```

3) Функция $f(x) = -3/(2+(x-2)^2) + 0.5$ – рис. 48-2.

a=-10
b=10
eps=0.000001
Minimum f(x)=-9.6064277846595014E-001
for x= 2.2321427931472462E+000

4) Функция $f(x) = 1.5 \cdot \exp(-(x-4)^2)$ – рис. 49-1.

a=-10
b=10
eps=0.000001
Maximum f(x)= 1.4999999624167102E+000
for x= 4.0001582893770156E+000

5) Функция $f(x) = -2 \cdot \exp(-(x+3)^2) - 0.5$ – рис. 49-2.

a=-10
b=10
eps=0.000001
Minimum f(x)=-2.4999999874800078E+000
for x=-2.9999208798637018E+000

Эффективность метода дихотомии деления отрезка на четыре части примерно одинакова при малых ε [2, с. 118]. Относительное уменьшение начального интервала неопределённости также находится по формуле (49). При заданной $R(N)$ количество вычислений функции, требующееся для обеспечения заданной точности ε можно оценить, как [2, с. 114]:

$$N \geq \frac{2 \ln R(N)}{\ln 0.5}. \qquad (50)$$

v16 § 5. Метод золотого сечения

При реализации алгоритмов оптимизации одним из основных источников замедления счёта и накопления погрешности считается

вычисление значения целевой функции. То есть, чем большее количество раз приходится вычислять функцию, тем медленнее работает программа. Метод, использующий золотое сечение для определения точек внутри отрезка неопределённости, в которых будут вычисляться значения функции для анализа, позволяет уменьшить количество таких вычислений.

Термин «золотое сечение» имеет древнее происхождение; оно используется во многих областях науки и техники (например, в строительстве), а также в искусстве (изобразительное искусство, музыка и т. д.).

Золотым сечением отрезка $[a, b]$ (рис. 50-1) называется точка (на рисунке – x_1), которая делит его в **пропорции золотого сечения**, то есть когда **длина ме́ньшего отрезка относится к длине бо́льшего отрезка так же, как длина бо́льшего отрезка относится к длине всего отрезка**:

$$\frac{x_1 - a}{b - x_1} = \frac{b - x_1}{b - a} \tag{51}$$

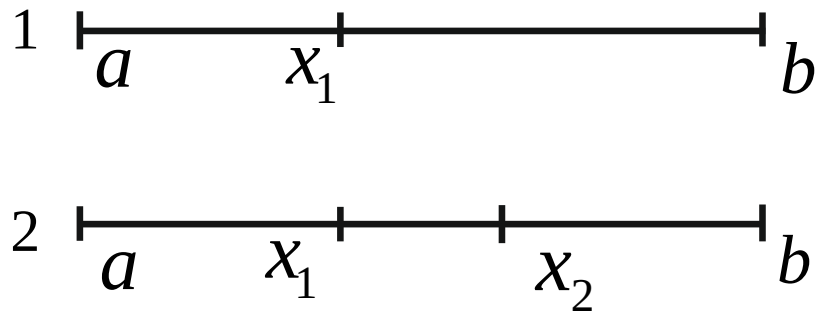


Рис. 50. 1 – x_1 – золотое сечение отрезка $[a, b]$; 2 – двустороннее золотое сечение

Основное свойство золотого сечения. Если сделать **двустороннее золотое сечение** отрезка $[a, b]$ (рис. 50-2), то **полученная точка** (x_2) **будет делить бо́льший отрезок** $(b - x_1)$, **в пропорции золотого сечения**.

При построении двустороннего сечения точка x_2 выбирается таким образом, чтобы соблюдалась пропорция золотого сечения, но как бы с другого края:

$$\frac{b-x_2}{x_2-a}=\frac{x_2-a}{b-a} \tag{52}$$

Из рис. 50-2 видно, что:

$$x_1-a=b-x_2 \tag{53}$$

и

$$x_2-a=b-x_1 \tag{54}$$

Тогда все четыре дроби в (51) и (52) равны между собой и должны быть равны какому-то положительному числу $\frac{1}{\tau}$:

$$\frac{x_1-a}{b-x_1}=\frac{b-x_1}{b-a}=\frac{b-x_2}{x_2-a}=\frac{x_2-a}{b-a}=\frac{1}{\tau}. \tag{54}$$

Если несколько преобразовать исходные пропорции (51) и (52), то можно получить аналогичное соотношение:

$$\frac{b-a}{b-x_1}=\frac{b-x_1}{x_1-a}=\frac{b-a}{x_2-a}=\frac{x_2-a}{b-x_2}=\frac{1}{\tau}. \tag{55}$$

Тогда из первой дроби имеем:

$$x_1=b-\tau\left(b-a\right), \tag{56}$$

а из третьей:

$$x_2=a+\tau\left(b-a\right). \tag{57}$$

Из последней дроби (55) и из (53) можно получить:

$$\frac{1}{\tau}=\frac{x_2-a}{x_1-a}. \tag{58}$$

Комбинируя (58), (56) и (57), имеем:

$$\frac{x_2-a}{x_1-a}=\frac{a+\tau\left(b-a\right)-a}{b-\tau\left(b-a\right)-a}=\frac{\tau\left(b-a\right)}{b-a-\tau\left(b-a\right)}=$$
$$=\frac{\tau\left(b-a\right)}{(b-a)(1-\tau)}=\frac{\tau}{1-\tau}=\frac{1}{\tau} \tag{59}$$

Из последнего равенства (59) получаем квадратное уравнение:

$$\tau^2+\tau-1=0, \tag{60}$$

корнями которого являются:

$$\tau_{1,2}=-\frac{1}{2}\pm\frac{\sqrt{5}}{2}. \tag{61}$$

Так как τ должна быть положительной, выбираем значение:

$$\tau = \frac{\sqrt{5}-1}{2}. \qquad (62)$$

Выражение (56) можно преобразовать $x_1 = a + b - a - \tau(b-a) = a + (1-\tau)(b-a)$.

Окончательно, имеем формулы для расчёта x_1 и x_2:

$$x_1 = a + \left(1 - \frac{\sqrt{5}-1}{2}\right)(b-a), \qquad (63)$$

$$x_2 = a + \frac{\sqrt{5}-1}{2}(b-a). \qquad (64)$$

Алгоритм минимизации представляет собой итерационный процесс уменьшения интервала неопределённости, начиная с исходного отрезка $[a, b]$, который включает в себя двустороннее золотое сечение этого интервала, сравнение между собой функций в полученных точках и отбрасывание отрезка, в котором корень не содержится.

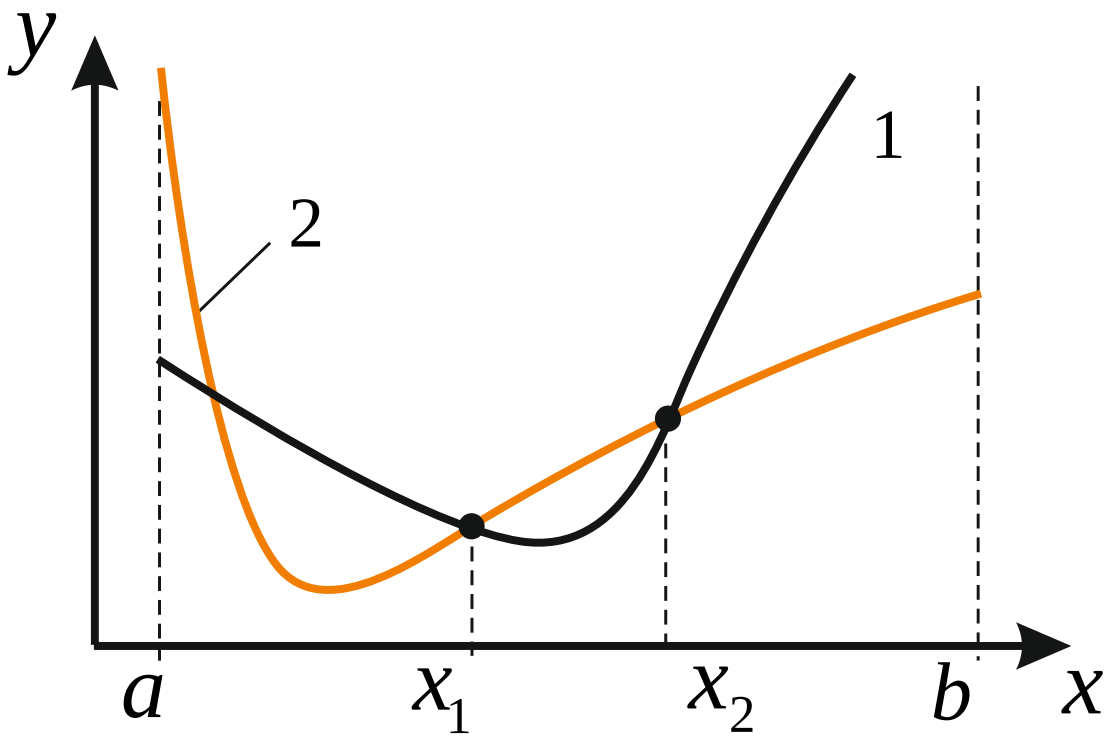


Рис. 51. Возможный вид унимодальных функций при $y_1 < y_2$.
Минимумы могут находиться на $[x_1, x_2]$ (1) или на $[a, x_1]$ (2)

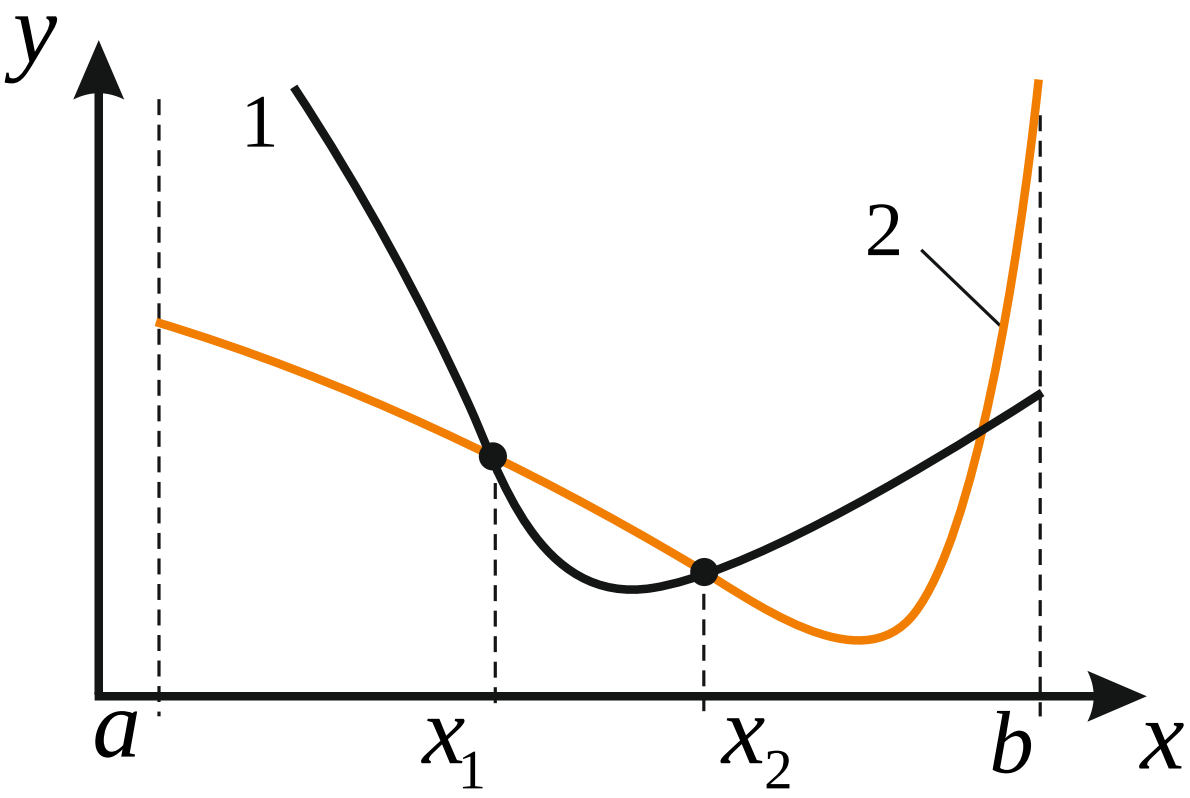


Рис. 52. Возможный вид унимодальных функций при $y_1 > y_2$. Минимумы могут находиться на $[x_1, x_2]$ (1) или на $[x_2, b]$ (2)

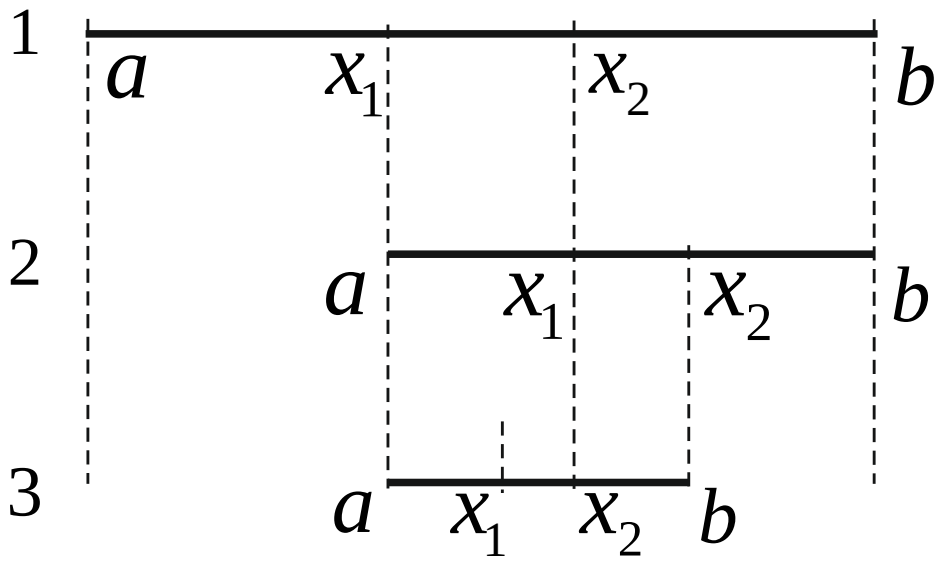


Рис. 53. Схема итераций метода золотого сечения; 1 – исходный отрезок; 2 – отбрасывание левого отрезка; 3 – отбрасывание правого отрезка

Анализ сводится к следующему. Если значения функций в средних точках $y_1 < y_2$ (рис. 51), то отрезок $[x_2, b]$ экстремум не содержит, так как невозможно провести через точки x_1 и x_2 унимодальную функцию, имеющую на нём минимум. Аналогично, в случае $y_1 > y_2$ (рис. 52), минимум не содержит отрезок $[a, x_1]$.

Перед началом итераций по формулам (63) и (64) рассчитываются точки x_1 и x_2 и значения функций в этих точках y_1 и y_2 (рис. 53). Затем начинаются итерации минимизации. В начале каждой итерации y_1 и y_2 сравниваются между собой и определяется отрезок, который необходимо отбросить (см. рис. 51 и 52). Допустим, что нужно отбросить левый отрезок (рис. 53-2). Для этого надо сделать присваивания $a := x_1$ (перенос a в точку x_1), $x_1 := x_2$ и, соответственно, $y_1 := y_2$, b остаётся прежним. Заново рассчитываются значения x_2 (по (64)) и y_2 (по $y_2 = f(x_2)$). После этого можно проводить следующую итерацию, если не достигнут критерий окончания процесса.

Если же нужно отбросить правый отрезок (рис. 53-3), то делаются присваивания $b := x_2$ (перенос b в точку x_2), $x_2 := x_1$ и $y_2 := y_1$, a остаётся прежним. Заново рассчитываются значения x_1 (по (63)) и y_1 (по $y_1 = f(x_1)$).

Нужно понимать, что отбрасывания правого и левого отрезков производится не поочерёдно, как это показано на рис. 53, а в соответствии с результатом анализа.

Таким образом, на каждой итерации производится только один новый расчёт функции, простые переприсваивания значений – операции очень быстрые, поэтому во многих случаях можно получить большой выигрыш во времени. В сравнении с методом деления отрезка на четыре части, где проводится два вычисления функции за итерацию, это более рационально, но при этом длина отбрасываемого в методе золотого сечения отрезка существенно меньше.

Критерием окончания итерационного процесса является остаточная длина интервала неопределённости в сравнении с заданной точностью:

$$|b - a| < \varepsilon. \qquad (65)$$

Ниже представлена программа на языке Lazarus (консольное приложение) для нахождения минимума или максимума (в последнем случае алгоритм скорректирован соответствующим образом). Для апробации программы использовалась функция $y = x^2 + \exp(x)$ (рис. 54), а также функции из предыдущего параграфа – $f(x) = 2\sin(4x)/x + 3;\quad x \neq 0$ (рис. 39, 40; см. также рис. 41, 42), локон Аньези (точнее – агвинея Ньютона) $f(x) = 3/(2 + (x+4)^2) + 0.5$ и её же форма с минусом $f(x) = -3/(2 + (x-2)^2) + 0.5$ – рис. 48, распределение Гаусса (нормальное распределение) $f(x) = 1.5 \cdot \exp(-(x-4)^2)$ и её же форма с минусом $f(x) = -2 \cdot \exp(-(x+3)^2) - 0.5$ – рис. 49.

Программа метода золотого сечения

```
program ZolotSech;
var a,b,eps,x1,x2,fx1,fx2:real; // описание концов диапазона a и b,
// точек двустороннего золотого сечения x1 и x2,
//значений функции в этих точках fx1 и fx2 и точности eps
// Функция расчёта левой точки золотого сечения:
function f1:real;begin f1:=a+(1-(sqrt(5)-1)/2)*(b-a);end;
// Функция расчёта правой точки золотого сечения:
function f2:real;begin f2:=a+(sqrt(5)-1)/2*(b-a);end;
// (Функции f1 и f2 – без параметров. так как они «видят» a и b
// в основной программе)
//Далее описываются шесть функций, по которым проводилась
// апробация, пять из которых закомментированы скобками
// { и }, см. текст
{function f(x:real):real;
begin f:=sqr(x)+exp(x);
end;}
{function f(x:real):real;
begin
if x<>0 then f:=2*sin(4*x)/x+3 else f:=8;
end;}
{function f(x:real):real;
begin
```

```
f:=3/(2+sqr(x+4))+0.5
end;}
{function f(x:real):real;
begin
f:=-3/(2+sqr(x-2))+0.5
end;}
{function f(x:real):real;
begin
f:=1.5*exp(-sqr(x-4))
end;}
function f(x:real):real;
begin
f:=-2*exp(-sqr(x+3))-0.5
end;
begin // начало основной программы
write('a=');readln(a); // ввод левого конца отрезка
write('b=');readln(b); // ввод правого конца отрезка
write('eps=');readln(eps); // ввод точности
x1:=f1;x2:=f2; // расчёт точек золотого сечения
fx1:=f(x1);fx2:=f(x2); // расчёт значений функции в этих точках
if (f((a+b)/2)<f(a)) or (f((a+b)/2)<f(b)) then
// установление вида экстремума; при выполнении этого условия
// имеет место минимум
begin // начало минимизации
repeat // открытие итерационного цикла
if fx1>fx2 // анализ поведения целевой функции; см. рис. 51 и 52
then // отбрасывание левого отрезка;
// необходимые действия согласно алгоритму:
begin
a:=x1;
x1:=x2;
fx1:=fx2;
x2:=f2;
fx2:=f(x2); // расчёт нового значения функции
end
else // отбрасывание правого отрезка;
// необходимые действия согласно алгоритму:
```

```
begin
b:=x2;
x2:=x1;
fx2:=fx1;
x1:=f1;
fx1:=f(x1); // расчёт нового значения функции
end
until (abs(b-a)<eps); // критерий выхода из цикла
// вывод результатов – значения функции и аргумента:
writeln('Minimum f(x)=',f((a+b)/2),' for x=',(a+b)/2);
end // конец минимизации
else
// при выполнении обратного условия имеет место максимум
begin // начало поиска максимума
repeat // открытие итерационного цикла
if fx1<fx2 // анализ поведения целевой функции; см. рис. 51 и 52
then // отбрасывание левого отрезка;
// необходимые действия согласно алгоритму:
begin
a:=x1;
x1:=x2;
fx1:=fx2;
x2:=f2;
fx2:=f(x2); // расчёт нового значения функции
end
else // отбрасывание правого отрезка;
// необходимые действия согласно алгоритму:
begin
b:=x2;
x2:=x1;
fx2:=fx1;
x1:=f1;
fx1:=f(x1); // расчёт нового значения функции
end
until abs(b-a)<eps; // критерий выхода из цикла
// вывод результатов – значения функции и аргумента:
writeln('Maximum f(x)=',f((a+b)/2),' for x=',(a+b)/2);
```

```
end; // конец поиска максимума
readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

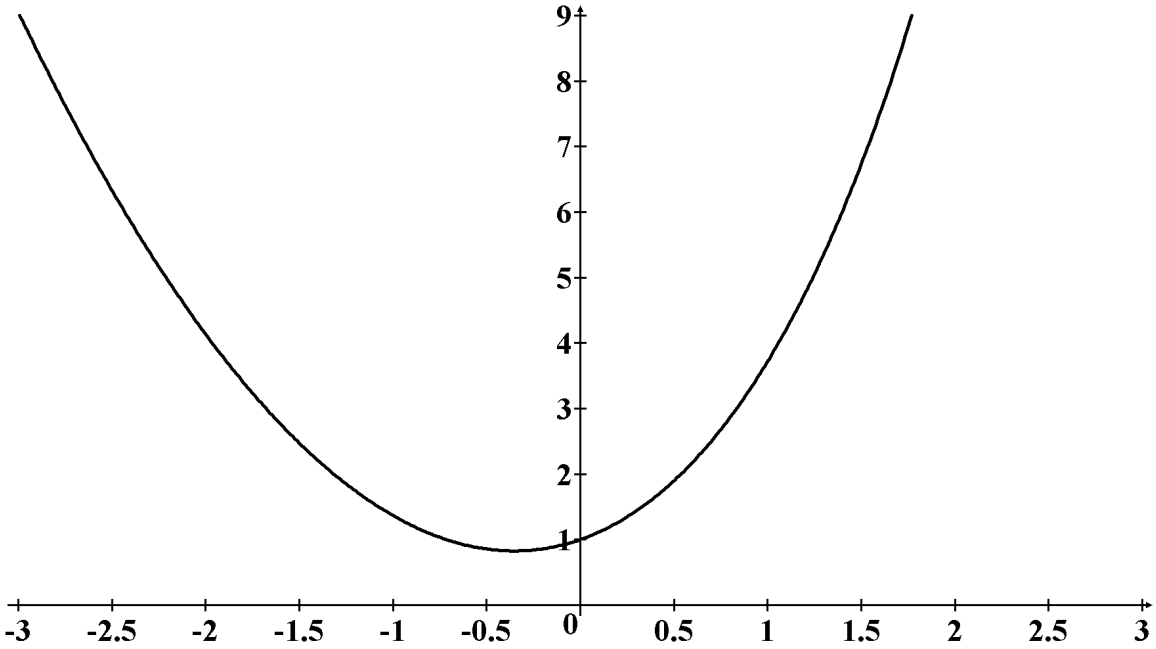


Рис. 54. Функция $y = x^2 + \exp(x)$

Результаты работы программы:

1) Функция $y = x^2 + \exp(x)$

```
a=-1
b=0
eps=0.000001
Minimum f(x)= 8.2718402612755892E-001
for x=-3.5173387126929057E-001
```

2) Функция $f(x) = 2\sin(4x)/x;\ \ x \neq 0$ для двух разных интервалов; см. похожие функции рис. 41 и 42.

a=0.4
b=1.8
eps=0.0004
Minimum f(x)= 1.2621309845667885E+000
for x= 1.1233252035942398E+000

a=1.5
b=2.5
eps=0.00004
Maximum f(x)= 4.0269964278965737E+000
for x= 1.9313068105184570E+000

3) Функция $f(x) = 3/(2+(x+4)^2) + 0.5$ – рис. 48-1.

a=-10
b=10
eps=0.000001
Maximum f(x)= 1.9999999999999694E+000
for x=-3.9999997973977877E+000

4) Функция $f(x) = -3/(2+(x-2)^2) + 0.5$ – рис. 48-2.

a=-10
b=10
eps=0.000001
Minimum f(x)=-9.9999999999993672E-001
for x= 2.0000002907929293E+000

5) Функция $f(x) = 1.5 \cdot \exp(-(x-4)^2)$ – рис. 49-1.

a=-10
b=10
eps=0.000001
Maximum f(x)= 1.4999999999999383E+000
for x= 3.9999997973977872E+000

6) Функция $f(x) = -2 \cdot \exp(-(x+3)^2) - 0.5$ – рис. 49-2.

a=-10
b=10
eps=0.000001
Minimum f(x)=-2.4999999999999498E+000
for x=-3.0000001585068565E+000

В методе золотого сечения относительное уменьшение начального интервала неопределённости составляет [2, с. 121]:

$$R(N) = \left(\frac{\sqrt{5}-1}{2} \right)^{N-1}, \quad (66)$$

где N – количество вычислений функции.

На каждой итерации происходит сокращение длины интервала неопределённости в $\frac{\sqrt{5}-1}{2}$ раза.

Если задана величина $R(N)$, то при заданной точности вычислений потребуется число вычислений функции [2, с. 121]:

$$N \geq 1 + \frac{\ln R(N)}{\ln \frac{\sqrt{5}-1}{2}}. \quad (67)$$

v17 § 6. Метод чисел Фибоначчи

Метод чисел Фибоначчи (часто используется сокращённое название «метод Фибоначчи») по алгоритму напоминает метод золотого сечения; в нём также интервал неопределённости делится на три отрезка и на каждом шаге отбрасывается левый или правый отрезок, а перед заходом на следующий шаг вычисляется только одно новое значение функции. Но процесс расчёта в этом методе не итерационный, как в методе золотого сечения, а выполняется строго задаваемое количество шагов.

Разбиение исходного и последующих отрезков на три части происходит с использованием **чисел Фибоначчи**, которые представляют собой **числовую последовательность**, которая

выстраивается следующим образом: **первые два числа – 0 и 1, а все последующие числа равны сумме двух предыдущих чисел**. Первые двадцать чисел выглядят так (иногда 0 не берётся):

0 1 1 2 3 5 8 13 21 34 55 89
144 233 377 610 987 1597 2584 4181

Таким образом, числа Фибоначчи определяются по формуле:

$$F_0 = F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \quad n = 2, 3, 4, \ldots \tag{68}$$

Также непосредственно число Фибоначчи с номером n может быть вычислено по формуле Бине:

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}. \tag{69}$$

Алгоритм сводится к следующему. Сначала задаётся количество вычислений функции N и заполняется массив чисел Фибоначчи $F_0, F_1, ..., F_N$. Затем по формулам

$$x_1 = a + \frac{F_{N-2}}{F_N} \cdot (b - a) \tag{70}$$

$$x_2 = a + \frac{F_{N-1}}{F_N} \cdot (b - a) \tag{71}$$

вычисляются внутренние точки x_1 и x_2 и функции в них, которые затем сравниваются между собой. По результатам этого анализа отбрасывается соответствующий отрезок, и вычисления повторяются. На каждом новом шаге k внутренние точки вычисляются по формулам:

$$x_1 = a + \frac{F_{N-k-3}}{F_{N-k-1}} \cdot (b - a) \tag{72}$$

$$x_2 = a + \frac{F_{N-k-2}}{F_{N-k-1}} \cdot (b - a) \tag{73}$$

Таким образом, число шагов строго определяется числом N.

Ниже представлена программа на языке Lazarus (консольное приложение) для нахождения минимума или максимума (в последнем случае алгоритм скорректирован соответствующим образом). Для

апробации программы использовалась функция $y = x^2 + \exp(x)$ (рис. 54), а также функции $f(x) = 2\sin(4x)/x + 3;\ \ x \neq 0$ (рис. 39, 40; см. также рис. 41, 42), локон Аньези (точнее – агвинея Ньютона) $f(x) = 3/(2 + (x+4)^2) + 0.5$ и её же форма с минусом $f(x) = -3/(2 + (x-2)^2) + 0.5$ – рис. 48, распределение Гаусса (нормальное распределение) $f(x) = 1.5 \cdot \exp(-(x-4)^2)$ и её же форма с минусом $f(x) = -2 \cdot \exp(-(x+3)^2) - 0.5$ – рис. 49.

Программа метода чисел Фибоначчи

```
program Fibonachio;
const n=30; // задание количества чисел Фибоначчи
type arfb=array[0..n] of longint; // описание типа массива
//чисел Фибоначчи
var fb:arfb; // описание массива чисел Фибоначчи
a,b,x1,x2:real; // описание концов диапазона a и b
// и двух внутренних точек x1, x2
i,k:integer; // описание индекса организации цикла i
// и номера шага k
// Далее описываются функции для вычисления внутренних точек
// x1 и x2 в зависимости от номера шага k
function f1(k:integer):real;begin f1:=a+fb[n-k-3]/fb[n-k-1]*(b-a);end;
function f2(k:integer):real;begin f2:=a+fb[n-k-2]/fb[n-k-1]*(b-a);end;
// Процедура расчёта массива чисел Фибоначчи:
procedure FFb(var fb:arfb);
var i:integer; // описание индекса для внутреннего цикла
begin
fb[0]:=0;fb[1]:=1;
for i:=2 to n do fb[i]:=fb[i-1]+fb[i-2];
end;
//Далее описываются пять функций, по которым проводилась
// апробация, четыре из которых закомментированы скобками
// { и }, см. текст
{function f(x:real):real;
begin f:=sqr(x)+exp(x);
```

```
end;}
{function f(x:real):real;
begin
if x<>0 then f:=2*sin(4*x)/x+3 else f:=8;
end;}
{function f(x:real):real;
begin
f:=3/(2+sqr(x+4))+0.5
end;}
{function f(x:real):real;
begin
f:=-3/(2+sqr(x-2))+0.5
end;}
{function f(x:real):real;
begin
f:=1.5*exp(-sqr(x-4))
end;}
function f(x:real):real;
begin
f:=-2*exp(-sqr(x+3))-0.5
end;
begin // начало основной программы
FFb(fb); // расчёт массива чисел Фибоначчи
for i:=0 to n do writeln(fb[i]); // вывод чисел Фибоначчи
// (не обязательно)
write('a=');readln(a); // ввод левого конца отрезка
write('b=');readln(b); // ввод правого конца отрезка
if (f((a+b)/2)<f(a)) or (f((a+b)/2)<f(b)) then
// установление вида экстремума; при выполнении этого условия
// имеет место минимум
begin // начало минимизации
x1:=f1(0); // расчёт левой внутренней точки
x2:=f2(0); // расчёт правой внутренней точки
for k:=0 to n-2 do begin // начало цикла пошаговых вычислений
if f(x1)<f(x2) // проверка условия отбрасывания
// правого отрезка
then
```

```
begin // необходимые действия согласно алгоритму:
b:=x2; // отбрасывания правого отрезка
x2:=x1;
x1:=f1(k); // расчёт нового значения левой внутренней точки
end
else // иначе – условие отбрасывания левого отрезка
begin // необходимые действия согласно алгоритму
a:=x1; // отбрасывания левого отрезка
x1:=x2;
x2:=f2(k); // расчёт нового значения правой внутренней точки
end;
end;
// вывод результатов – значения функции и аргумента:
writeln('Minimum f(x)=',f((a+b)/2),' for x=',(a+b)/2);
end // конец минимизации
else
// при выполнении обратного условия имеет место максимум
begin // начало поиска максимума
x1:=f1(0); // расчёт левой внутренней точки
x2:=f2(0); // расчёт правой внутренней точки
for k:=0 to n-2 do begin // начало цикла пошаговых вычислений
if f(x1)<f(x2) // проверка условия отбрасывания
// правого отрезка
then
begin // необходимые действия согласно алгоритму:
b:=x2; // отбрасывания правого отрезка
x2:=x1;
x1:=f1(k); // расчёт нового значения левой внутренней точки
end
else // иначе – условие отбрасывания левого отрезка
begin // необходимые действия согласно алгоритму
a:=x1; // отбрасывания левого отрезка
x1:=x2;
x2:=f2(k); // расчёт нового значения правой внутренней точки
end;
end;
// вывод результатов – значения функции и аргумента:
```

```
writeln('Maximum f(x)=',f((a+b)/2),' for x=',(a+b)/2);
end; // конец поиска максимума
readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

1) Функция $y = x^2 + \exp(x)$

```
a=-1
b=0
Minimum f(x)= 8.2718402612752495E-001
for x=-3.5173369068900551E-001
```

2) Функция $f(x) = 2\sin(4x)/x; \quad x \neq 0$ для двух разных интервалов; см. похожие функции рис. 41 и 42.

```
a=0.4
b=1.8
Minimum f(x)= 1.2621309743390809E+000
for x= 1.1233538051071155E+000

a=1.5
b=2.5
Maximum f(x)= 4.0269964281983048E+000
for x= 1.9313139993274500E+000
```

3) Функция $f(x) = 3/(2+(x+4)^2) + 0.5$ – рис. 48-1.

```
a=-10
b=10
Maximum f(x)= 1.9999999998916638E+000
for x=-3.9999879813215058E+000
```

4) Функция $f(x) = -3/(2+(x-2)^2) + 0.5$ – рис. 48-2.

a=-10
b=10
Minimum f(x)=-9.9999999989166444E-001
for x= 2.0000120186392980E+000

5) Функция $f(x) = 1.5 \cdot \exp(-(x-4)^2)$ – рис. 49-1.

a=-10
b=10
Maximum f(x)= 1.4999999997833289E+000
for x= 4.0000120186269541E+000

6) Функция $f(x) = -2 \cdot \exp(-(x+3)^2) - 0.5$ – рис. 49-2.

a=-10
b=10
Minimum f(x)=-2.4999999997111031E+000
for x=-2.9999879813338213E+000

В методе Фибоначчи относительное уменьшение начального интервала неопределённости составляет [2, с. 125]:

$$R(N) = \frac{1}{F_N}, \tag{74}$$

где N – количество вычислений функции.

На k-ой итерации происходит сокращение длины интервала неопределённости согласно $\frac{F_{N-k-1}}{F_{N-k}}$ [2, с. 125].

Метод Фибоначчи считается самым оптимальным среди однотипных, но его недостатком является то, что он не итерационный, а пошаговый; чтобы получить заданную точность, часто приходится проводить несколько вычислений с разным N и выбирать наиболее приемлемый результат. Как вариант, можно эти расчёты заложить в одной программе с автоматическим анализом результатов.

v18 § 7. Метод квадратичной интерполяции

В этом методе в качестве приближения к искомому минимуму используется минимум параболы (рис. 55), проведённой через три точки целевой функции $f(x_1)$, $f(x_2)$ и $f(x_3)$, причём точки x_1, x_2 и x_3 выбираются по определённому критерию, а значения целевой функции y_1, y_2 и y_3 могут быть рассчитаны и сравнены между собой. (Другие названия метода: метод Пауэлла, метод парабол, метод квадратичной аппроксимации; метод входит в группу методов полиномиальной интерполяции, в которую также входит более точный метод кубической интерполяции. Термин «интерполяция» подходит больше, чем термин «аппроксимация», так как задаваемые точки считаются заданными точно).

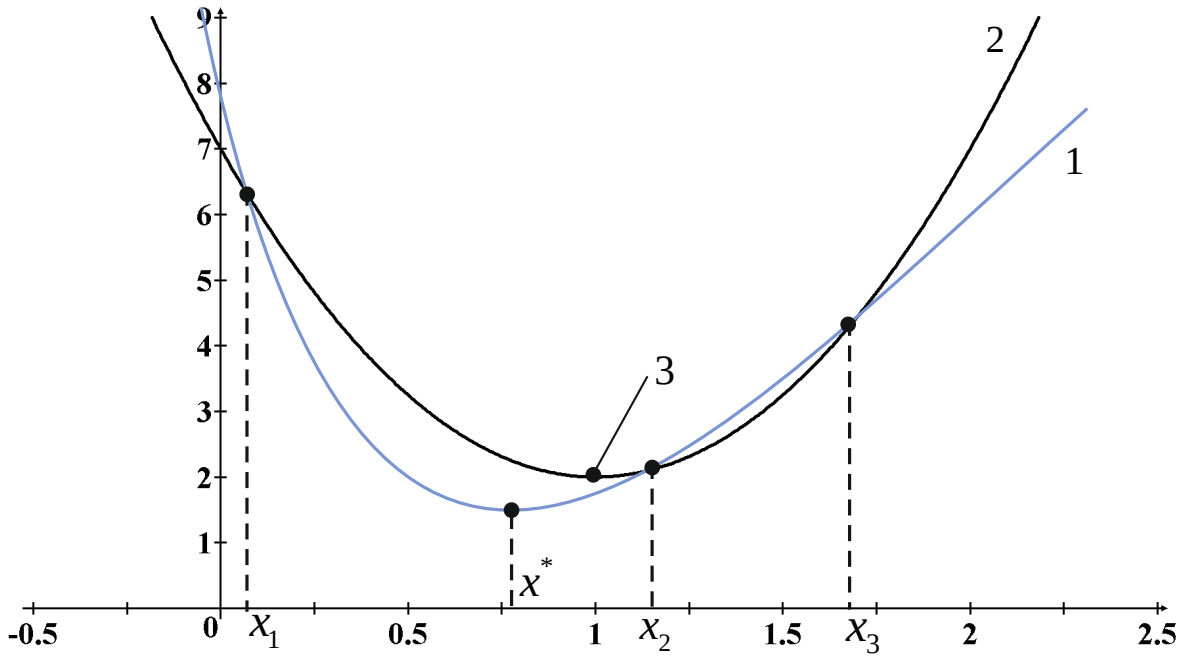


Рис. 55. Метод квадратичной интерполяции. 1 – целевая функция; 2 – интерполирующая парабола; x_1, x_2 и x_3 – выбранные точки; x^* – искомый минимум; 3 – минимум параболы

Алгоритм сводится к следующему. Выбирается начальная точка x_1 – начальное приближение к минимуму x^*, малый шаг Δx, точность по оси x ε. Точка x_2 находится по формуле:

$$x_2 = x_1 + \Delta x \tag{75}$$

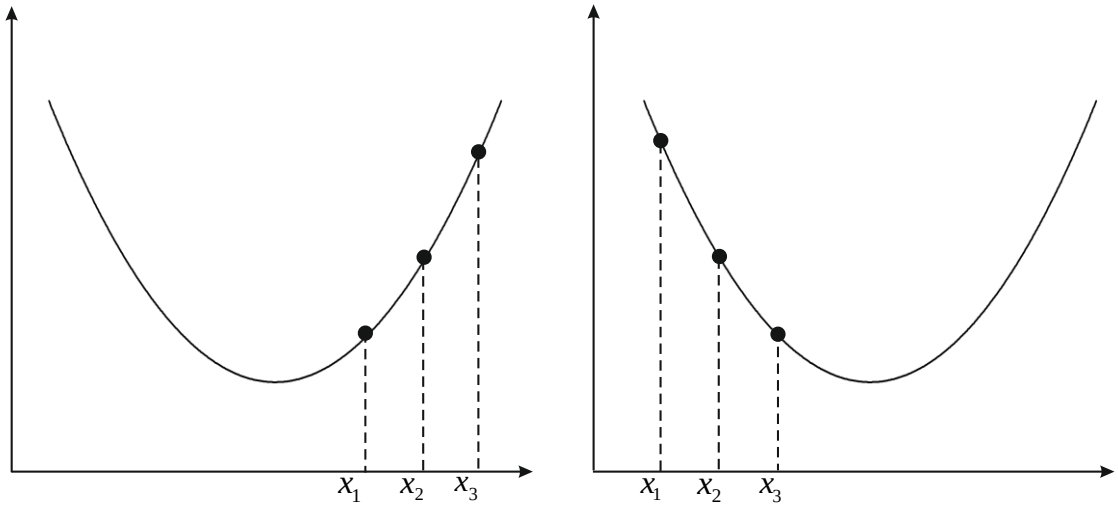
В зависимости от хода целевой функции третья точка должна оказаться либо справа от x_2 (когда $f(x_1) > f(x_2)$, эта ситуация имеет место на рис. 55), либо – слева от x_1 (когда $f(x_1) < f(x_2)$). В первом случае третья точка вычисляется как:

$$x_3 = x_1 + 2\Delta x, \tag{76}$$

а во втором – как:

$$x_3 = x_1 - \Delta x \tag{77}$$

Чтобы сохранить общность подхода, можно эти точки переобозначить по порядку следования.



а) б)

Рис. 56. Построение интерполирующей параболы; x_1, x_2 и x_3 – выбранные точки; а) – минимум параболы лежит слева от точек; б) – минимум параболы лежит справа от точек

Далее вычисляются значения целевой функции $f(x_1)$, $f(x_2)$ и $f(x_3)$, и через полученные точки проводится интерполирующая парабола. Так как через три точки, не лежащие на одной прямой, можно провести только одну параболу, при этом построении возможны четыре случая, показанные на рис. 56 и 57.

Минимум параболы может лежать слева от трёх точек (рис. 56-а), справа от них (рис. 55-б), между точками x_1, и x_2 или между x_2 и x_3 (рис. 57).

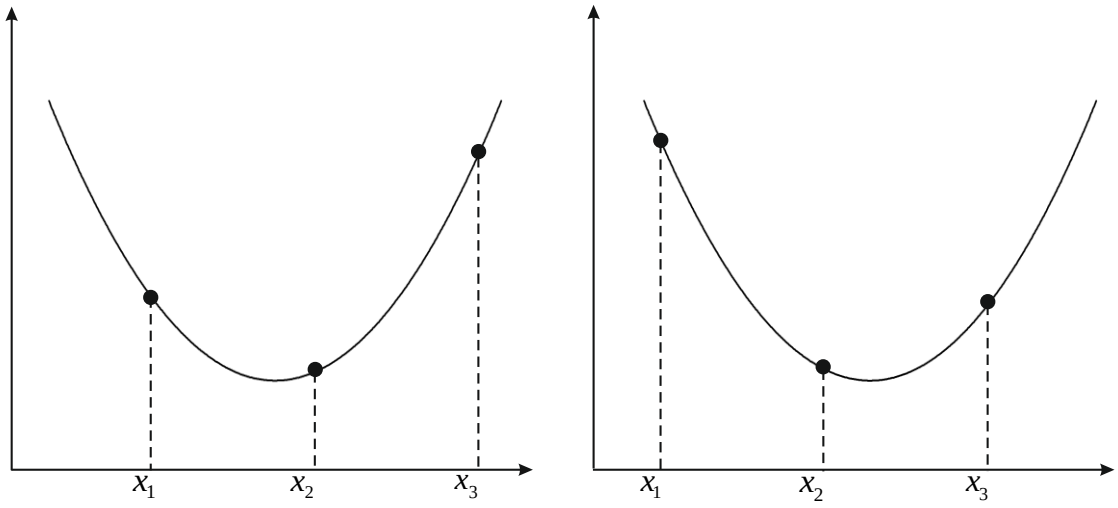


Рис. 57. Построение интерполирующей параболы; x_1, x_2 и x_3 – выбранные точки; минимум параболы лежит внутри интервала точек, два варианта

Во всех случаях парабола может быть построена по одной и той же формуле, причём нет смысла получать уравнение само́й параболы, достаточно найти её минимум. Этот минимум на следующей итерации служит исходной точкой x_1.

Уравнение параболы имеет вид:

$$y = ax^2 + bx + c, \quad a \neq 0 \tag{78}$$

Для трёх точек имеем систему:

$$\begin{cases} ax_1^2 + bx_1 + c = y_1 \\ ax_2^2 + bx_2 + c = y_2 \\ ax_3^2 + bx_3 + c = y_3 \end{cases} \tag{79}$$

Её определитель

$$\begin{vmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{vmatrix} = (x_1 - x_2)(x_1 - x_3)(x_2 - x_3) \tag{80}$$

представляет собой определитель Вандермонда, который вычисляется указанным способом. По формуле Крамера решение системы (79) относительно a, b и c может быть представлено следующим образом:

$$a=\frac{\begin{vmatrix} y_1 & x_1 & 1 \\ y_2 & x_2 & 1 \\ y_3 & x_3 & 1 \end{vmatrix}}{\begin{vmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{vmatrix}};\quad b=\frac{\begin{vmatrix} x_1^2 & y_1 & 1 \\ x_2^2 & y_2 & 1 \\ x_3^2 & y_3 & 1 \end{vmatrix}}{\begin{vmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{vmatrix}};\quad c=\frac{\begin{vmatrix} x_1^2 & x_1 & y_1 \\ x_2^2 & x_2 & y_2 \\ x_3^2 & x_3 & y_3 \end{vmatrix}}{\begin{vmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{vmatrix}} \tag{81}$$

Если определители в числителях разложить по первому, второму и третьему столбцу соответственно, то, с учётом (80), имеем:

$$a=\frac{y_1\left(x_2-x_3\right)-y_2\left(x_1-x_3\right)+y_3\left(x_1-x_2\right)}{\left(x_1-x_2\right)\left(x_1-x_3\right)\left(x_2-x_3\right)}=$$

$$=\frac{y_1\left(x_2-x_3\right)+y_2\left(x_3-x_1\right)+y_3\left(x_1-x_2\right)}{\left(x_1-x_2\right)\left(x_1-x_3\right)\left(x_2-x_3\right)};$$

$$b=\frac{-y_1\left(x_2^2-x_3^2\right)+y_2\left(x_1^2-x_3^2\right)-y_3\left(x_1^2-x_2^2\right)}{\left(x_1-x_2\right)\left(x_1-x_3\right)\left(x_2-x_3\right)}= \tag{82}$$

$$=-\frac{y_1\left(x_2^2-x_3^2\right)+y_2\left(x_3^2-x_1^2\right)+y_3\left(x_1^2-x_2^2\right)}{\left(x_1-x_2\right)\left(x_1-x_3\right)\left(x_2-x_3\right)};$$

$$c=\frac{y_1\left(x_2-x_3\right)x_2x_3-y_2\left(x_1-x_3\right)x_3x_1+y_3\left(x_1-x_2\right)x_1x_2}{\left(x_1-x_2\right)\left(x_1-x_3\right)\left(x_2-x_3\right)};$$

Необходимым условием экстремума параболы является:

$$y'=2ax+b=0, \tag{83}$$

откуда можно выразить стационарную точку (минимум параболы):

$$x_{par}^{*}=-b/\left(2a\right) \tag{84}$$

Тогда окончательно имеем:

$$x_{par}^{*}=\frac{y_1\left(x_2^2-x_3^2\right)+y_2\left(x_3^2-x_1^2\right)+y_3\left(x_1^2-x_2^2\right)}{y_1\left(x_2-x_3\right)+y_2\left(x_3-x_1\right)+y_3\left(x_1-x_2\right)} \tag{85}$$

Таким образом, для расчёта минимума параболы требуются только точки x_1, x_2, x_3 и значения целевой функции в них y_1, y_2, y_3.

Если вычисленное значение целевой функции в точке x^*_{par} окажется меньше любой из y_1, y_2, y_3, то эту точку можно выбрать в качестве начальной точки x_1 на следующей итерации. Но может так оказаться, что оно будет больше какой-то из точек. Тогда в качестве нового значения x_1 на следующей итерации выбирается минимальная точка (точнее – с минимальным значением функции), шаг Δx уменьшается (например – в два раза) и после этого проводится следующая итерация. Что касается шага Δx, то в начале минимизации он не может быть слишком маленьким, сравнимым с точностью ε, так как, если начальное приближение x_1 в первой итерации выбрано неудачно, то для решения задачи потребуется очень большое количество итераций. А вот при приближении к решению x^* шаг можно уменьшать, и, когда его величина станет меньше ε, итерации можно прекращать. Таким образом, критерием окончания итерационного процесса можно считать:

$$\Delta x < \varepsilon \qquad (86)$$

Программа, реализующая метод квадратичной интерполяции, представлена на языке Lazarus (консольное приложение) Для апробации программы использовалась функция $y = x^2 + \exp(x)$ (рис. 54), а также функции $f(x) = 2\sin(4x)/x + 3;\ \ x \neq 0$ (рис. 39, 40; см. также рис. 41, 42), локон Аньези (точнее – агвинея Ньютона) $f(x) = 3/(2 + (x+4)^2) + 0.5$ и её же форма с минусом $f(x) = -3/(2 + (x-2)^2) + 0.5$ – рис. 48, распределение Гаусса (нормальное распределение) $f(x) = 1.5 \cdot \exp(-(x-4)^2)$ и её же форма с минусом $f(x) = -2 \cdot \exp(-(x+3)^2) - 0.5$ – рис. 49.

Программа метода квадратичной интерполяции

```
program SquareInterpol;
var a,b,eps,h,xp,min,max:real; // описание концов диапазона a и b,
// функций y1, y2, y3 в трёх внутренних точках x1, x2, x3,
// точности eps и шага h, и двух вспомогательных переменных
// для поиска минимума и максимума min и max
i,imin,imax:integer; // описание целочисленных переменных:
```

```
// i – для организации цикла, imin и imax – номера (индексы)
// элементов массива при поиске минимума и максимума
x,y:array[1..3] of real; // описание массивов из трёх
// элементов каждый: x – три опорные точки,
// y – значения целевой функции в них
//Далее описываются пять функций, по которым проводилась
// апробация, четыре из которых закомментированы скобками
// { и }, см. текст
{function f(x:real):real;
begin f:=sqr(x)+exp(x);
end;}
{function f(x:real):real;
begin
if x<>0 then f:=2*sin(4*x)/x+3 else f:=8;
end;}
{function f(x:real):real;
begin
f:=3/(2+sqr(x+4))+0.5
end;}
{function f(x:real):real;
begin
f:=-3/(2+sqr(x-2))+0.5
end;}
{function f(x:real):real;
begin
f:=1.5*exp(-sqr(x-4))
end;}
function f(x:real):real;
begin
f:=-2*exp(-sqr(x+3))-0.5
end;
begin // начало основной программы
write('a=');readln(a); // ввод левого конца отрезка
write('b=');readln(b); // ввод правого конца отрезка
write('h=');readln(h); // ввод шага
write('eps=');readln(eps); // ввод точности
x[1]:=(a+b)/2; // выбор первой опорной точки
```

```
// как середины диапазона унимодальности
if (f((a+b)/2)<f(a)) or (f((a+b)/2)<f(b)) then
// установление вида экстремума; при выполнении этого условия
// имеет место минимум
begin // начало минимизации
repeat // открытие итерационного цикла
x[2]:=x[1]+h; // расчёт второй опорной точки
// Далее следует анализ и расчёт третьей опорной точки
// и, если потребуется, переназначение номеров опорных точек
if f(x[1])>f(x[2])
then x[3]:=x[1]+2*h
else begin
x[3]:=x[2];
x[2]:=x[1];
x[1]:=x[1]-h;
end;
// Далее в опорных точках рассчитываются
// значения целевой функции:
y[1]:=f(x[1]);
y[2]:=f(x[2]);
y[3]:=f(x[3]);
// Расчёт точки минимума параболы по формуле (85):
xp:=(y[1]*(sqr(x[2])-sqr(x[3]))+y[2]*(sqr(x[3])-
sqr(x[1]))+y[3]*(sqr(x[1])-sqr(x[2])))/
(y[1]*(x[2]-x[3])+y[2]*(x[3]-x[1])+y[3]*(x[1]-x[2]));
// Далее следует поиск минимума целевой функции в точках
// x[1], x[2], x[3 и xp
min:=f(xp);imin:=0; // начальное присваивание минимуму
// и его местоположению
for i:=1 to 3 do // цикл поиска минимума
if min>y[i] then begin
min:=y[i];imin:=i;end;
if imin=0 then
x[1]:=xp // если минимум – в точке xp,
// то она становится новой первой опорной точкой
else begin // если же минимум – среди опорных точек,
// то он становится новой первой опорной точкой,
```

```
        // и при этом уполовинивается шаг h:
        x[1]:=x[imin];
        h:=h/2;
        end;
        until (2*h)<eps; // критерий выхода из цикла
        // вывод результатов – значения функции и аргумента:
        writeln('Minimum f(x)=',f(x[1]),' for x=',x[1]);
        end // конец минимизации
        else
        // при выполнении обратного условия имеет место максимум
        begin // начало поиска максимума
        repeat // открытие итерационного цикла
        x[2]:=x[1]+h; // расчёт второй опорной точки
        // Далее следует анализ и расчёт третьей опорной точки
        // и, если потребуется, переназначение номеров опорных точек
        if f(x[1])<f(x[2])
        then x[3]:=x[1]+2*h
        else begin
        x[3]:=x[2];
        x[2]:=x[1];
        x[1]:=x[1]-h;
        end;
        // Далее в опорных точках рассчитываются
        // значения целевой функции:
        y[1]:=f(x[1]);
        y[2]:=f(x[2]);
        y[3]:=f(x[3]);
        // Расчёт точки максимума параболы по формуле (85):
        xp:=(y[1]*(sqr(x[2])-sqr(x[3]))+y[2]*(sqr(x[3])-
sqr(x[1]))+y[3]*(sqr(x[1])-sqr(x[2])))/
        (y[1]*(x[2]-x[3])+y[2]*(x[3]-x[1])+y[3]*(x[1]-x[2]));
        // Далее следует поиск максимума целевой функции в точках
        // x[1], x[2], x[3 и xp
        max:=f(xp);imax:=0; // начальное присваивание минимуму
        // и его местоположению
        for i:=1 to 3 do // цикл поиска максимума
        if max<y[i] then begin
```

```
max:=y[i];imax:=i;end;
if imax=0 then
x[1]:=xp // если максимум – в точке xp,
// то она становится новой первой опорной точкой
else begin // если же максимум – среди опорных точек,
// то он становится новой первой опорной точкой,
// и при этом уполовинивается шаг h:
x[1]:=x[imax];
h:=h/2;
end;
until (h*2)<eps; // критерий выхода из цикла
// вывод результатов – значения функции и аргумента:
writeln('Maximum f(x)=',f(x[1]),' for x=',x[1]);
end;
readln;
end.
```

Примечание. В программе можно использовать не четыре переменные min, max, imin, imax, а две, назвав их, например, extr и iextr, или сохранив любую пару прежних переменных.

Результаты работы программы:

1) Функция $y = x^2 + \exp(x)$

```
a=-1
b=0
h=0.2
eps=0.000001
Minimum f(x)= 8.2718402612765662E-001
for x=-3.5173339843750001E-001
```

2) Функция $f(x) = 2\sin(4x)/x; \quad x \neq 0$ для двух разных интервалов; см. похожие функции рис. 41 и 42.

```
a=0.4
```

b=1.8
h=0.2
eps=0.0004
Minimum f(x)= 1.2621310750745280E+000
for x= 1.1234374999999999E+000

a=1.5
b=2.5
h=0.2
eps=0.0004
Maximum f(x)= 4.0269963956394825E+000
for x= 1.9312499999999999E+000

3) Функция $f(x) = 3/(2+(x+4)^2)+0.5$ – рис. 48-1.

Программа работает **некорректно!** (пологие участки)
a=-10
b=10
h=0.2
eps=0.000001
Maximum f(x)= 7.0053468572392741E-001
for x=-3.9999923706054691E-001

a=-5.5
b=-3
h=0.2
eps=0.000001
Maximum f(x)= 2.0000000000000000E+000
for x=-4.0000000000000000E+000

4) Функция $f(x) = -3/(2+(x-2)^2)+0.5$ – рис. 48-2.

Программа работает **некорректно!** (пологие участки)
a=-10
b=10
h=0.2

eps=0.000001
Minimum f(x)=-3.7209209485503869E-001
for x= 7.9999847412109382E-001

a=-1
b=4
h=0.2
eps=0.000001
Minimum f(x)=-1.0000000000000000E+000
for x= 2.0000000000000000E+000

5) Функция $f(x)=1.5\cdot\exp(-(x-4)^2)$ – рис. 49-1.

В данном случае программа сработала корректно:
a=-10
b=10
h=0.2
eps=0.000001
Maximum f(x)= 1.4999999999999776E+000
for x= 4.0000001223772310E+000

a=2
b=5
h=0.2
eps=0.000001
Maximum f(x)= 1.5000000000000000E+000 for
x= 4.0000000000000000E+000

6) Функция $f(x)=-2\cdot\exp(-(x+3)^2)-0.5$ – рис. 49-2.

Программа работает **некорректно!** (пологие участки)
a=-10
b=10
h=0.2
eps=0.000001
Minimum f(x)=-5.0231844914984070E-001

for x=-3.9999923706054691E-001

a=-5
b=-2
h=0.2
eps=0.000001
Minimum f(x)=-2.5000000000000000E+000
for x=-3.0000000000000000E+000

Как показали расчёты, для функций 3), 4), 6) (см. рис. 48 и 49) программа даёт неверные результаты; это связано с наличием пологих участков целевой функции, на которых построение параболы часто происходит некорректно. Главный вывод: метод непригоден для оптимизации пологих функций.

v19 Глава 3. МЕТОДЫ МНОГОМЕРНОЙ ОПТИМИЗАЦИИ

v20 § 1. Введение

Пожалуй, самыми распространёнными на практике задачами оптимизации являются задачи безусловной оптимизации функций многих переменных. В настоящее время разработано множество методов для численного решения этих задач, охватывающих практически все возможные на практике специфические случаи и учитывающие множество нюансов. Профессионализм пользователей как раз и состоит в правильном выборе метода или методов для решения своей конкретной задачи. Выбор этот будет определяться, прежде всего, видом и характером целевой функции, возможностью или невозможностью взять от неё первую и вторую производные, возможностью установления унимодальности функции на выбранной области, другой дополнительной информацией.

Как уже отмечалось, методы подразделяются на методы нулевого порядка (когда достаточно иметь аналитическое выражение самой функции для вычисления её значения в нужных точках), первого порядка (когда кроме самой функции требуется знать первую производную) и второго порядка (требуется знание и второй производной). Как правило, методы первого и второго порядка имеют более быструю сходимость (часто – квадратичную), но требуют предварительную математическую проработку – взятие производных и так далее. Методы нулевого порядка не требуют предварительный математический анализ и могут применяться автоматически как вспомогательные методы в каких-то сложных алгоритмах. Кроме того, они могут применяться в случае недифференцируемых функций или когда взятие производных очень сложно. Недостаток медленной сходимости в настоящее время нивелируется высокой скоростью действия современных компьютеров.

Во многих методах многомерной оптимизации используются как вспомогательные, методы оптимизации одномерной.

Из рассмотренных методов методы покоординатного спуска и Нелдера-Мида являются методами нулевого порядка, а градиентные методы, в том числе – Флетчера-Ривса, – первого.

Все рассмотренные методы и программы относятся к случаю целевых функций двух аргументов; это сделано для наглядности. Но все алгоритмы могут быть распространены на многомерный случай.

Представленные программы апробировались на большом количестве одних и тех же функций, что позволяет сравнить результаты. При сравнении следует помнить, что полученные приближённые числа должны округляться в соответствии с заданной точностью. Например, результаты

1.6718301852488904E-007
-6.4078969063486751E-010
1.9229229880270118E-007
-2.9997156654710097E-008
-6.6754800779508075E-007
-4.5314363950917557E-012
7.1008032012561376E-010
3.2310453165753202E-006

представляют собой одно и то же число, равное нулю.

v21 § 2. Метод покоординатного спуска

Алгоритм метода сводится к следующему. Задаётся начальная точка в виде координат x1 и x2, которая должна лежать в области унимодальности функции. Также задаётся шаг и требуемая точность; если в реальной задаче размерность осей x1 и x2 различна (то есть – это разные физические величины), то шаг и точность задаются по каждой оси отдельно (h1, h2, eps1 и eps2). Шаг должен быть не слишком большим, но и не слишком маленьким, иначе потребуется очень большое количество итераций.

На первой итерации от центральной начальной точки делаются шаги по направлению осей координат – вверх, направо, вниз и налево. В полученных точках вычисляется значение функции и среди них определяется минимальное значение (если производится минимизация). На рис. 58 внизу показан возможный порядок вычисления. В данном примере минимальное значение – в точке под номером 1. В эту точку на следующей итерации следует перенести центральную точку и повторить алгоритм. Однако при приближении к минимуму возможен «перескок» экстремальной точки, если шаг

окажется слишком большим. Поэтому в конце каждой итерации проверяется **монотонность**, то есть необходимость того, чтобы вновь найденная минимальная точка была ниже центральной. В примере рис. 58 (внизу) монотонность соблюдается. Если же монотонность нарушается, то центральная точка остаётся на месте, шаг уменьшается (например – наполовину) и алгоритм повторяется заново.

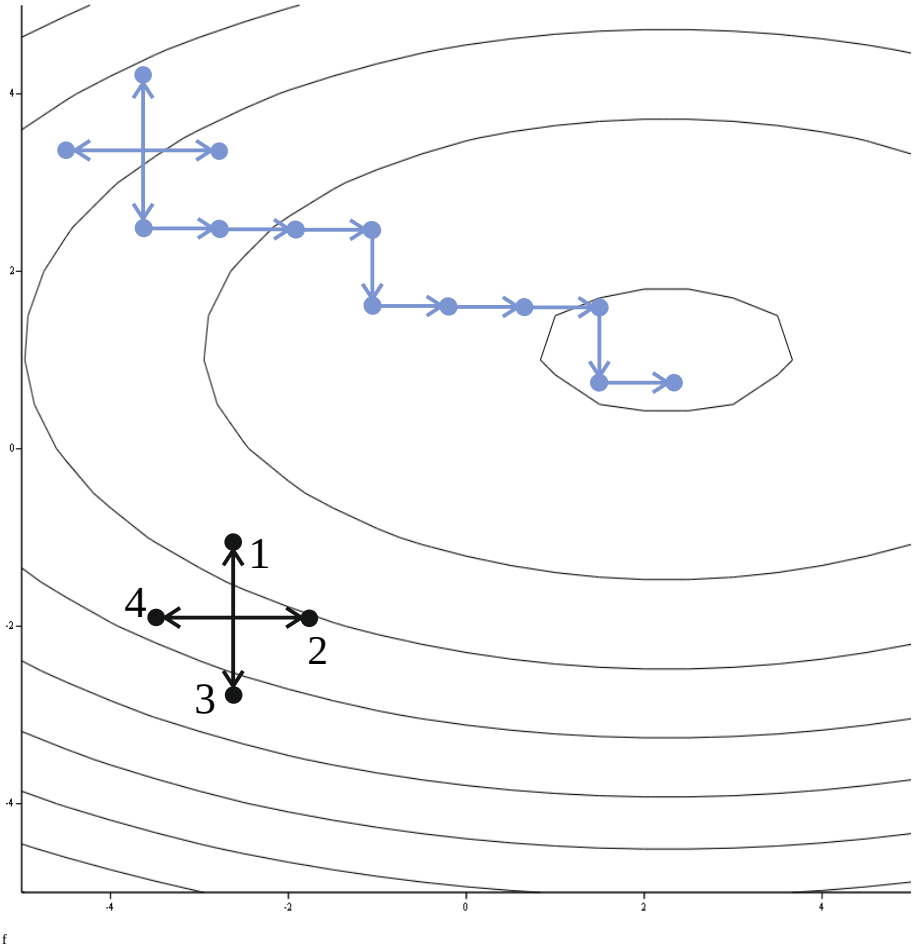


Рис. 58. Линии уровня функции $f(x1, x2) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2$.
Принцип алгоритма метода покоординатного спуска (см. текст)

В верхней части рис. 58 показано движение центральной точки от начального «креста» – 10 итераций. Далее возможно нарушение монотонности.

Итерации можно прекращать, когда длина шага станет меньше заданной точности.

Программа, реализующая метод покоординатного спуска, представлена на языке Lazarus (консольное приложение) Для апробации программы использовались функции:

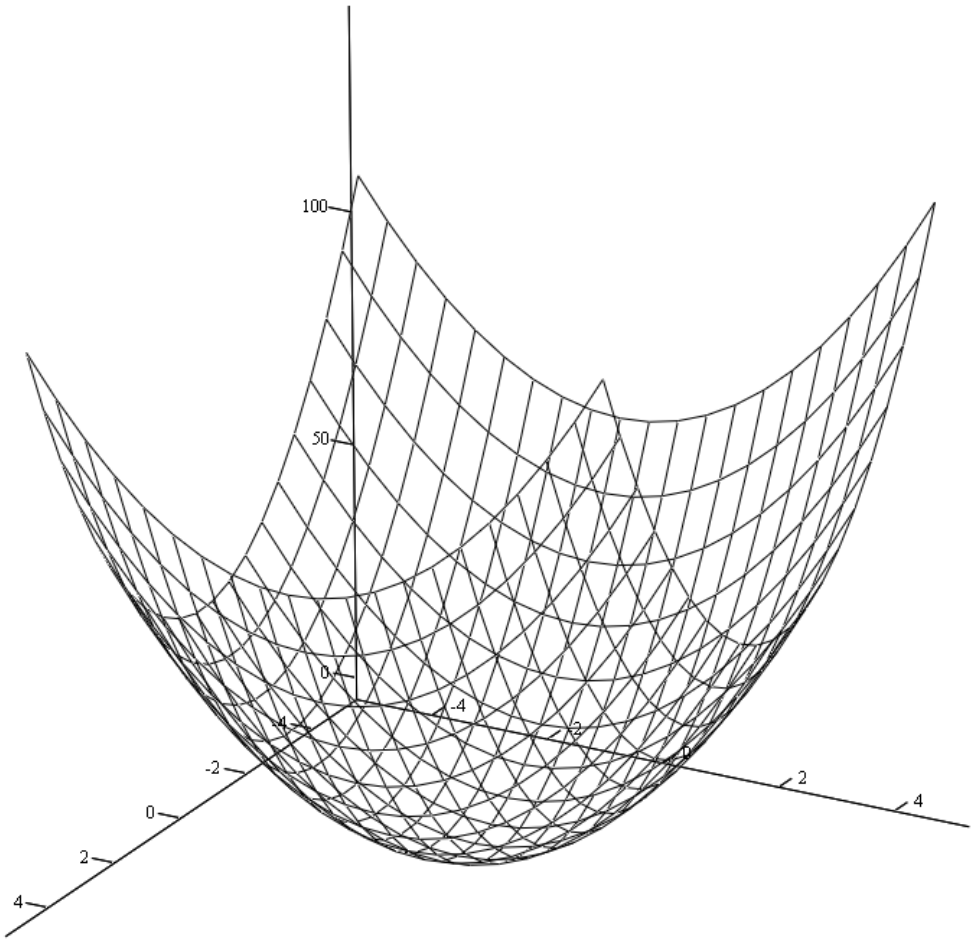
1. $f(x1, x2) = 3x_1^2 + 2x_2^2 + x_1 + 2x_2$ (рис. 59),
2. $f(x1, x2) = (x_1 - 2)^2 + (x_2 - 1)^2 + x_1 + 2x_2$ (рис. 14),
3. $f(x1, x2) = -3(x_1 - 2)^2 - 4(x_2 - 3)^2$ (max) (рис. 60),
4. $f(x1, x2) = -2(x_1 - 2)^2 - 8(x_2 - 1)^2 + x_1 + 2x_2$ (max) (рис. 61),
5. $f(x1, x2) = 2(3 - x_1)^2 + 8(x_1^2 - 2x_2)^2$ (рис. 37),
6. $f(x1, x2) = 2x_1 - 3.5x_2 + \exp(x_1^2 + x_2^2)$ (рис. 62),
7. $f(x1, x2) = x_1 - 1.4x_2 + \exp(0.01x_1^2 + 0.11x_2^2)$ (рис. 63).

(87)

Программа метода покоординатного спуска

```
program PocoordSpusk;
var
x1,x2,
// описание координат приближения экстремума;
h1,h2,eps1,eps2,
// описание шагов и точностей по осям x1 и x2 (если размерности
// разные)
min: real;
// описание вспомогательной переменной для поиска
// минимального или максимального значения
m, // описание местоположения минимального или
// максимального значения
k:integer;
// описание метки вида экстремума: k=1 – минимум,
// k=0 – максимум;
//Далее описываются 7 функций, по которым проводилась
// апробация программы, 6 из которых закомментированы
// скобками { и }
```

```
{function f(x1,x2:real):real;                    // №1
begin f:=3*sqr(x1)+2*sqr(x2)+x1+ 2*x2;end;}
```



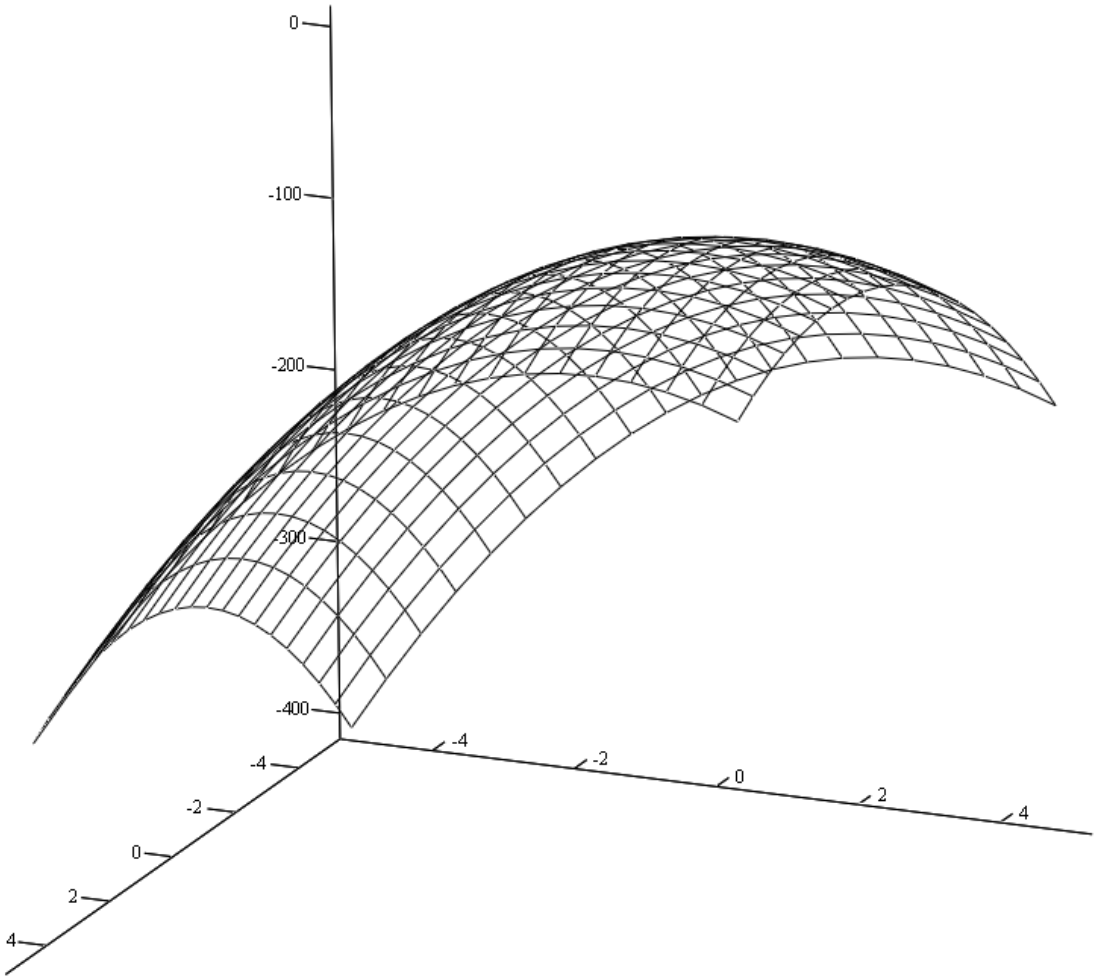
f

Рис. 59. Функция $f(x_1, x_2) = 3x_1^2 + 2x_2^2 + x_1 + 2x_2$

```
{function f(x1,x2:real):real;                    //№ 2
begin f:=sqr(x1-2)+sqr(x2-1)+x1+2*x2;end;}

{function f(x1,x2:real):real;                    // №3; max
begin f:=-3*sqr(x1-2)-4*sqr(x2-3);end;}

{function f(x1,x2:real):real;                    // №4; max
begin f:=-2*sqr(x1-2)-8*sqr(x2-1)+x1+2*x2;end;}
```



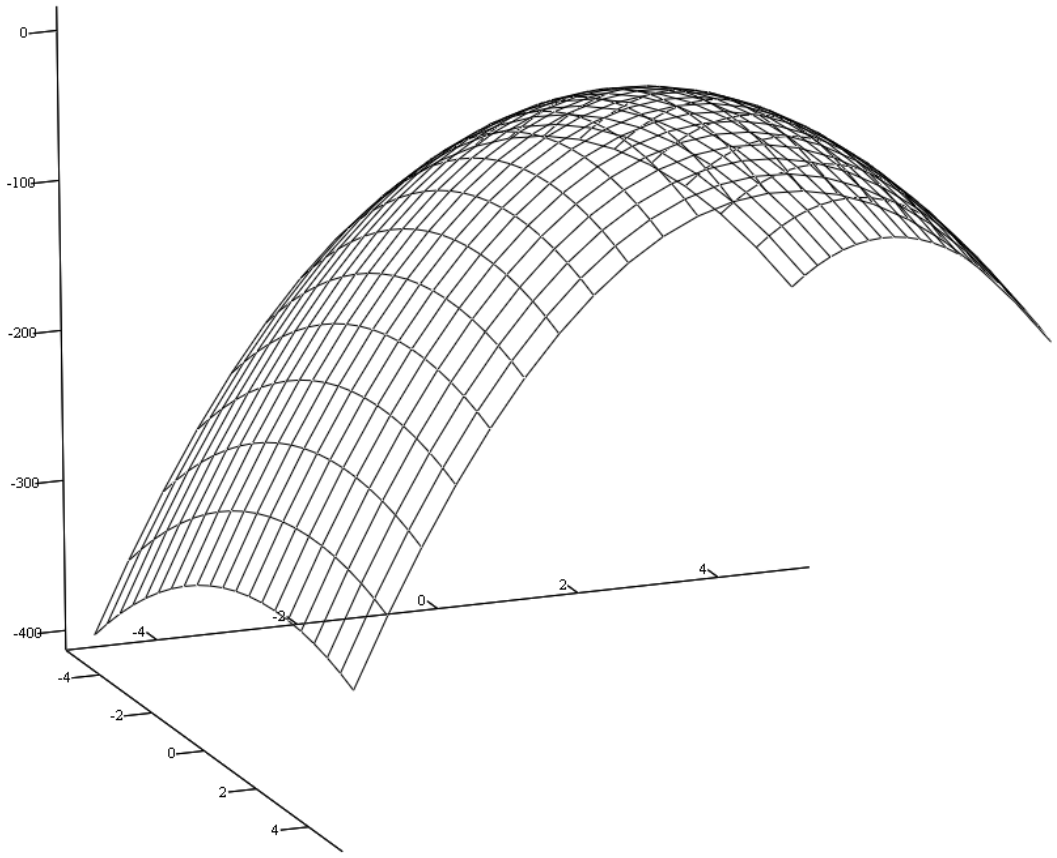
f

Рис. 60. Функция $f(x_1, x_2) = -3\left(x_1 - 2\right)^2 - 4\left(x_2 - 3\right)^2$

```
{function f(x1,x2:real):real;                    // №5
begin f:=2*sqr(3-x1)+8*sqr(sqr(x1)-2*x2);end;}

{function f(x1,x2:real):real;                    // №6
begin f:=2*x1-3.5*x2+exp(sqr(x1)+sqr(x2));end;}

function f(x1,x2:real):real;                     // №7
begin f:=x1-1.4*x2+exp(0.01*sqr(x1)+0.11*sqr(x2));end;
```



f

Рис. 61. Функция $f(x1, x2) = -2\left(x_1 - 2\right)^2 - 8\left(x_2 - 1\right)^2 + x_1 + 2x_2$

```
// Начало основной программы
begin
// определение пользователем вида экстремума
// (минимум или максимум):
write('Minimisation? (yes - 1, no - 0):');
readln(k);
// ввод метки экстремума
write('x1=');readln(x1);
write('x2=');readln(x2);
write('h1=');readln(h1);
write('h2=');readln(h2);
write('eps1=');readln(eps1);
```

```
write('eps2=');readln(eps2);
// ввод координат начального приближения, шагов и точности
```

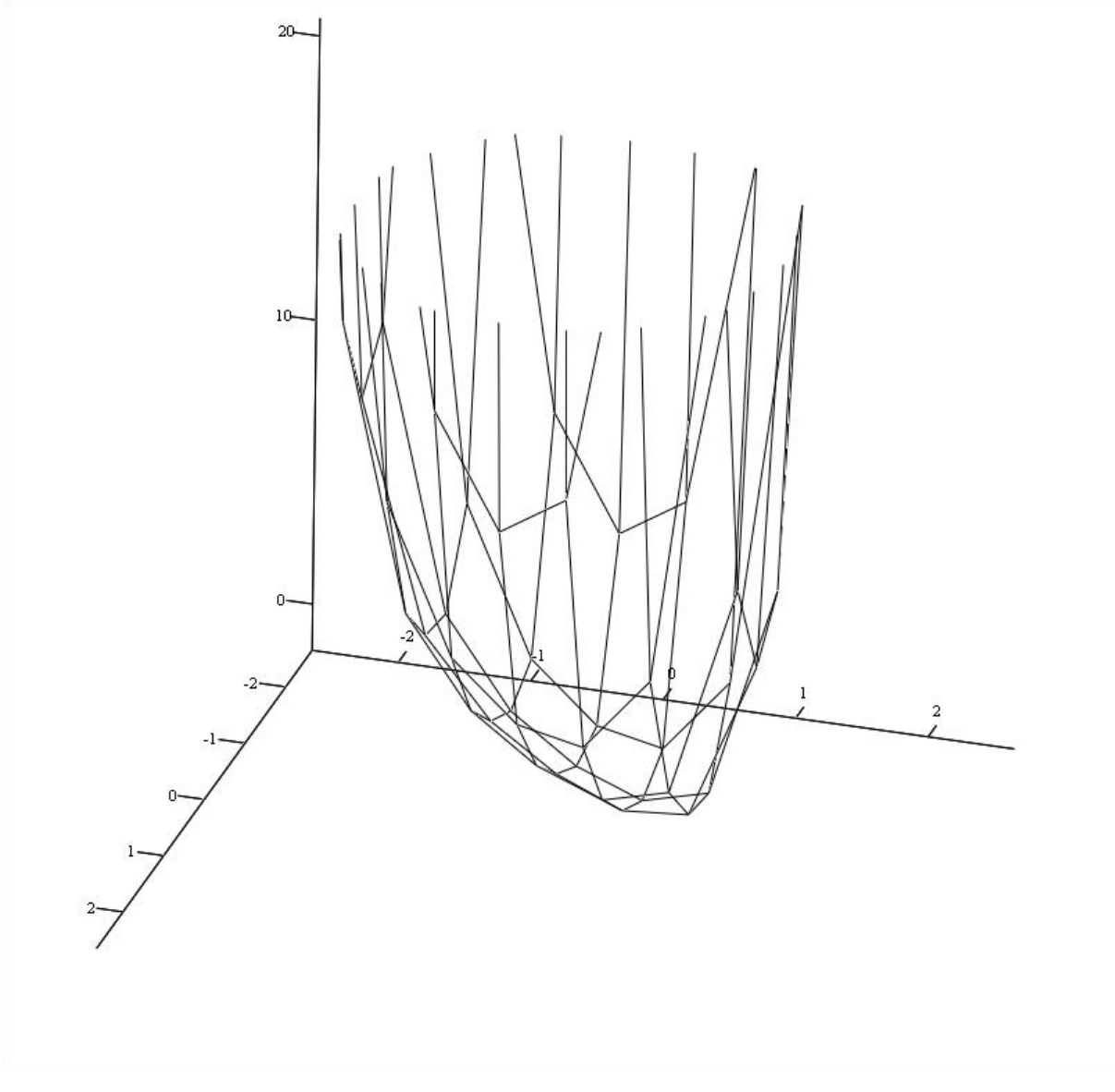
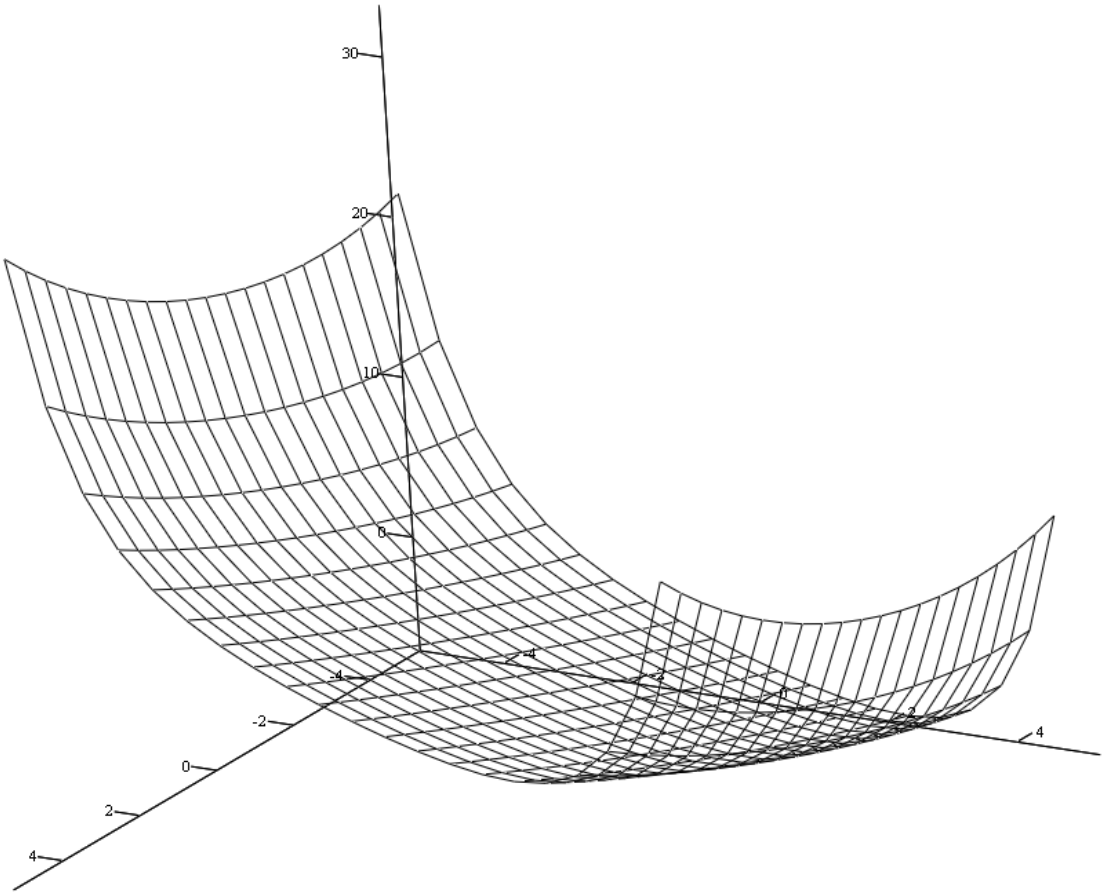


Рис. 62. Функция $f(x_1, x_2) = 2x_1 - 3.5x_2 + \exp\left(x_1^2 + x_2^2\right)$

```
// начало оптимизации; при выполнении этого условия
// имеет место минимум:
if k=1 then
begin // начало минимизации;
```



f

Рис. 63. Функция $f(x_1, x_2) = x_1 - 1.4x_2 + \exp\left(0.01x_1^2 + 0.11x_2^2\right)$

```
repeat // открытие итерационного цикла
// в цикле от исходной точки последовательно проводятся шаги
// по четырём направлениям вдоль осей и ищется наименьшее или
// наибольшее значение функции в полученных точках, затем
// центральная точка перемещается в соответствии с алгоритмом
min:=f(x1,x2+h2);
m:=1;
if f(x1+h1,x2)<min then
begin
min:=f(x1+h1,x2);
m:=2;
```

```
end;
if f(x1,x2-h2)<min then
begin
min:=f(x1,x2-h2);
m:=3;
end;
if f(x1-h1,x2)<min then
begin
min:=f(x1-h1,x2);
m:=4;
end;
if min<f(x1,x2) then
case m of // оператор выбора действий по результатам анализа
1:x2:=x2+h2;
2:x1:=x1+h1;
3:x2:=x2-h2;
4:x1:=x1-h1;
end
else // если нарушается монотонность. шаги уменьшаются
begin
h1:=h1/2;
h2:=h2/2;
end;
until (h1<eps1) and (h2<eps2);
// критерий выхода из итерационного цикла
writeln('Minimum f(x1,x2)=',f(x1,x2),' for x1=',x1,' and x2=',x2);
// вывод результатов – значения функции и аргумента

end // конец минимизации
else
// при выполнении обратного условия имеет место максимум
begin// начало поиска максимума

repeat // открытие итерационного цикла
// в цикле от исходной точки последовательно проводятся шаги
// по четырём направлениям вдоль осей и ищется наименьшее или
// наибольшее значение функции в полученных точках, затем
```

```
// центральная точка перемещается в соответствии с алгоритмом
min:=f(x1,x2+h2);
m:=1;
if f(x1+h1,x2)>min then
begin
min:=f(x1+h1,x2);
m:=2;
end;
if f(x1,x2-h2)>min then
begin
min:=f(x1,x2-h2);
m:=3;
end;
if f(x1-h1,x2)>min then
begin
min:=f(x1-h1,x2);
m:=4;
end;
if min>f(x1,x2) then
case m of // оператор выбора действий по результатам анализа
1:x2:=x2+h2;
2:x1:=x1+h1;
3:x2:=x2-h2;
4:x1:=x1-h1;
end
else // если нарушается монотонность. шаги уменьшаются
begin
h1:=h1/2;
h2:=h2/2;
end;
until (h1<eps1) and (h2<eps2);
// критерий выхода из итерационного цикла
writeln('Maximum f(x1,x2)=',f(x1,x2),' for x1=',x1,' and x2=',x2);
// вывод результатов – значения функции и аргумента

end; // конец поиска максимума
```

```
readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

1. Функция $f(x1, x2) = 3x_1^2 + 2x_2^2 + x_1 + 2x_2$ (рис. 59)

```
Minimisation? (yes - 1, no - 0):1
x1=5
x2=3
h1=0.2
h2=0.2
eps1=0.000001
eps2=0.000001
Minimum f(x1,x2)=-5.8333333333255721E-001
for x1=-1.6666717529297093E-001
and x2=-5.0000000000000044E-001
```

2. Функция $f(x1, x2) = (x_1 - 2)^2 + (x_2 - 1)^2 + x_1 + 2x_2$ (рис. 14),

```
Minimisation? (yes - 1, no - 0):1
x1=5
x2=3
h1=0.2
h2=0.2
eps1=0.000001
eps2=0.000001
Minimum f(x1,x2)= 2.7500000000000000E+000
for x1= 1.4999999999999976E+000
and x2=-3.8857805861880479E-016
```

3. Функция $f(x1, x2) = -3(x_1 - 2)^2 - 4(x_2 - 3)^2$ (max) (рис. 60),

```
Minimisation? (yes - 1, no - 0):0
```

x1=0.1
x2=0.1
h1=0.2
h2=0.2
eps1=0.000001
eps2=0.000001
Maximum f(x1,x2)=-1.3805065841367707E-030
for x1= 1.9999999999999996E+000
and x2= 3.0000000000000004E+000

4. Функция $f(x1, x2) = -2(x_1 - 2)^2 - 8(x_2 - 1)^2 + x_1 + 2x_2$ (max) (рис. 61),

Minimisation? (yes - 1, no - 0):0
x1=0.1
x2=0.1
h1=0.2
h2=0.2
eps1=0.000001
eps2=0.000001
Maximum f(x1,x2)= 4.2500000000000000E+000
for x1= 2.2500000000000000E+000
and x2= 1.1249999999999998E+000

5. Функция $f(x1, x2) = 2(3 - x_1)^2 + 8(x_1^2 - 2x_2)^2$ (рис. 37)

Minimisation? (yes - 1, no - 0):1
x1=-4.5
x2=-4.5
h1=0.2
h2=0.2
eps1=0.000001
eps2=0.000001
Minimum f(x1,x2)= 2.4139882292983987E-008
for x1= 2.9998901367187649E+000
and x2= 4.4996704101561900E+000

6. Функция $f(x1, x2) = 2x_1 - 3.5x_2 + \exp\left(x_1^2 + x_2^2\right)$ (рис. 62),

Minimisation? (yes - 1, no - 0):1
x1=-2
x2=-2
h1=0.1
h2=0.1
eps1=0.000001
eps2=0.000001
Minimum f(x1,x2)=-1.3801176989107438E+000
for x1=-4.4588775634765559E-001
and x2= 7.8030242919921922E-001

7. Функция $f(x1, x2) = x_1 - 1.4x_2 + \exp\left(0.01x_1^2 + 0.11x_2^2\right)$ (рис. 63).

Minimisation? (yes - 1, no - 0):1
x1=-1
x2=-1
h1=0.2
h2=0.2
eps1=0.000001
eps2=0.000001
Minimum f(x1,x2)=-8.8124053629380938E+000
for x1=-1.1251463317871085E+001
and x2= 1.4320053100585937E+000

Метод покоординатного спуска с дроблением шага – один из самых простых методов многомерной оптимизации. Он обладает очень медленной сходимостью, но имеет все преимущества методов нулевого порядка. Алгоритм легко распространяется на случай функций более чем двух аргументов.

v22 § 3. Метод покоординатного спуска с использованием золотого сечения

В этом методе начальным приближением является не точка, а прямоугольник, точно содержащий минимум, причём на этом прямоугольнике выполняется условие унимодальности (рис. 64). При этом задаются четыре координаты: a_1 и a_2 – по оси x_1, b_1 и b_2 – по оси x_2. На рис. 64 показаны линии уровня целевой функции, искомая точка минимума – 1, и прямоугольник, построенный по заданным координатам. Ось функции направлена из начала координат перпендикулярно плоскости рисунка.

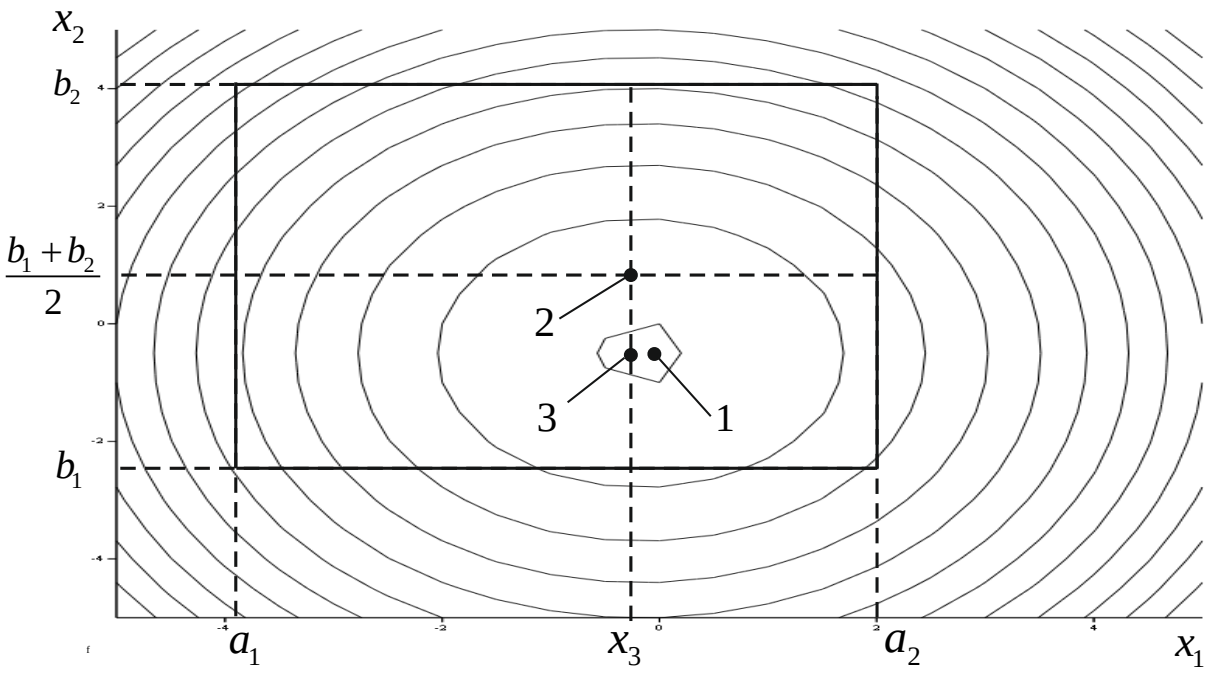


Рис. 64. Принцип алгоритма метода покоординатного спуска с использованием золотого сечения. 1 – искомая точка минимума, координаты заданного начального прямоугольника: a_1, a_2 – по оси x_1 b_1, b_2 – по оси x_2. Показана первая итерация; первый этап: $\frac{b_1+b_2}{2}$ – середина отрезка b_1b_2, 2 – приближение, полученное после первого

этапа, x_3 – координата точки 2; 3 – приближение, полученное после второго этапа

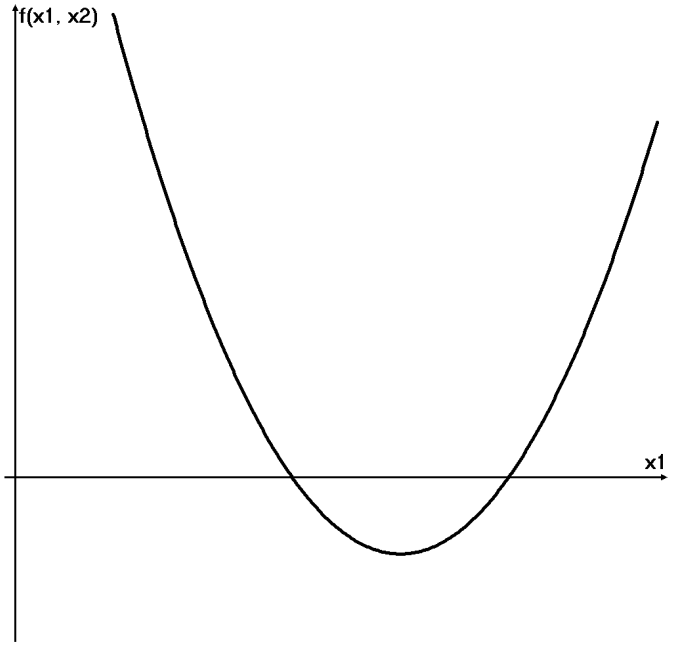


Рис. 65. Сечение плоскостью, перпендикулярной плоскости рис. 64. Проекция показанного сечения на рис. 64 представляет собой прямую, проходящую через $\frac{b_1+b_2}{2}$ точку 2.

Одна итерация состоит из двух этапов. На первом этапе перпендикулярно плоскости рисунка 64 и параллельно оси x_1 (например) проводится секущая плоскость, в которой затем проводится одномерная оптимизация. В первой итерации она проводится через середину отрезка на противоположной оси $\frac{b_1+b_2}{2}$, так как нет другого выбора. Полученное при этом сечение показано на рис. 65. После одномерной оптимизации по методу золотого сечения (например) будет найдена точка 2 (рис. 64), через которую на втором этапе проводится секущая плоскость, параллельная оси x_2. В этой секущей плоскости вновь проводится одномерная оптимизация и находится точка 3, как результат первой итерации. Затем алгоритм повторяется (в последующих итерациях, разумеется, на первом этапе секущая плоскость проводится не через середину отрезка

противоположной оси, а через предыдущее приближение). Метод имеет значительно более высокую сходимость по отношению в простому методу покоординатного спуска.

Программа, реализующая метод покоординатного спуска с золотым сечением, представлена на языке Lazarus (консольное приложение) Для апробации программы использовались функции (87).

Программа метода покоординатного спуска с использованием золотого сечения

```
program PocoordSpuskZolotSech2;
var
a1,a2,b1,b2,
// описание прямоугольника начального приближения:
// a1 и a2 – координаты по оси x1, b1 и b2 – координаты по оси x2
x1,x2,x01,x02,
// описание координат приближения экстремума;
// x01 и x02 – предыдущее приближение;
// x1 и x2 – последующее приближение
c,eps: real;
// описание точности eps и вспомогательной переменной c,
// которая представляет собой координату на оси,
// противоположной той оси, на которой проводится золотое
// сечение отрезка; через точку c проводится плоскость,
// перпендикулярная координатной плоскости аргументов
// и параллельная оси, содержащей отрезок
k:integer;
// описание метки вида экстремума: k=1 – минимум,
// k=0 – максимум;

//Далее описываются 7 функций, по которым проводилась
// апробация программы, 6 из которых закомментированы
// скобками { и }

{function f(x1,x2:real):real;                    // №1
begin f:=3*sqr(x1)+2*sqr(x2)+x1+ 2*x2;end;}
```

```
{function f(x1,x2:real):real;                    //№ 2
begin f:=sqr(x1-2)+sqr(x2-1)+x1+2*x2;end;}

{function f(x1,x2:real):real;                    // №3; max
begin f:=-3*sqr(x1-2)-4*sqr(x2-3);end;}

{function f(x1,x2:real):real;                    // №4; max
begin f:=-2*sqr(x1-2)-8*sqr(x2-1)+x1+2*x2;end;}

{function f(x1,x2:real):real;                    // №5
begin f:=2*sqr(3-x1)+8*sqr(sqr(x1)-2*x2);end;}

{function f(x1,x2:real):real;                    // №6
begin f:=2*x1-3.5*x2+exp(sqr(x1)+sqr(x2));end;}

function f(x1,x2:real):real;                     // №7
begin f:=x1-1.4*x2+exp(0.01*sqr(x1)+0.11*sqr(x2));end;

// Далее описывается универсальная функция оптимизации по
// методу золотого сечения; в неё передаются концы отрезка
// a и b, координата c на противоположной оси, через которую
// проводится плоскость, в которой производится золотое
// сечение; и метка k1: при минимизации, если k1=1, то отрезок
// находится на оси x1, если k1=2, то – на x2, при поиске
// максимума, если k1=3 то отрезок находится на оси x1,
// если k1=4, то – на x2
function f2(a,b,c:real;k1:integer):real;
// Описание внутренних для функции переменных:
var tau,x1,x2: real;
// tau – параметр золотого сечения, x1 и x2 – внутренние точки
// золотого сечения
p:boolean; // логическая переменная для анализа
begin
tau:=(sqrt(5)-1)/2;
x1:=a+(1-tau)*(b-a);
x2:=a+tau*(b-a);
repeat
```

```
if k1=2 then p:=f(c,x1)>f(c,x2);
if k1=1 then p:=f(x1,c)>f(x2,c);
if k1=4 then p:=f(c,x1)<f(c,x2);
if k1=3 then p:=f(x1,c)<f(x2,c);
if p then
begin
a:=x1;
x1:=x2;
x2:=a+tau*(b-a);
end
else
begin
b:=x2;
x2:=x1;
x1:=a+(1-tau)*(b-a);
end;
until abs(b-a)<eps;
f2:=(a+b)/2;
end;

// Начало основной программы
begin
// определение пользователем вида экстремума
// (минимум или максимум):
write('Minimisation? (yes - 1, no - 0):');
readln(k); // ввод метки экстремума
write('a1='); readln(a1);
write('a2='); readln(a2);
write('b1='); readln(b1);
write('b2='); readln(b2);
// ввод координат прямоугольника начального приближения:
// a1 и a2 – координаты по оси x1, b1 и b2 – координаты по оси x2
write('eps='); readln(eps);
// ввод точности

// начало оптимизации; при выполнении этого условия
// имеет место минимум:
```

```
if k=1 then
begin // начало минимизации; за одну итерацию золотое сечение
// делается последовательно по обеим осям

c:=(b1+b2)/2;
// при минимизации по оси x1 точка c берётся как середина
// отрезка на противоположной оси x2
x1:=f2(a1,a2,c,1);
c:=x1;
// при минимизации по оси x2 (в этой же самой итерации) точка c
// берётся как найденная при минимизации на противоположной
// оси x1
x2:=f2(b1,b2,c,2);
repeat // открытие итерационного цикла, в котором проводятся по
// две минимизации (то есть на каждой оси) за одну итерацию
x01:=x1;
x02:=x2;
c:=x2;
x1:=f2(a1,a2,c,1);
c:=x1;
x2:=f2(b1,b2,c,2);
until (abs(x1-x01)<eps) and (abs(x2-x02)<eps);
// критерий выхода из итерационного цикла
writeln('Minimum f(x1,x2)=',f(x1,x2),' for x1=',x1,' and x2=',x2);
// вывод результатов – значения функции и аргумента

end // конец минимизации
else
// при выполнении обратного условия имеет место максимум
begin// начало поиска максимума; за одну итерацию золотое
// сечение делается последовательно по обеим осям

c:=(b1+b2)/2;
// при поиске максимума по оси x1 точка c берётся как середина
// отрезка на противоположной оси x2
x1:=f2(a1,a2,c,3);
c:=x1;
```

```
// при поиске максимума по оси x2 (в этой же самой итерации)
// точка с берётся как найденная при минимизации на
// противоположной оси x1
x2:=f2(b1,b2,c,4);

repeat // открытие итерационного цикла, в котором проводятся по
// две минимизации (то есть на каждой оси) за одну итерацию
x01:=x1;
x02:=x2;
c:=x2;
x1:=f2(a1,a2,c,3);
c:=x1;
x2:=f2(b1,b2,c,4);
until (abs(x1-x01)<eps) and (abs(x2-x02)<eps);
// критерий выхода из итерационного цикла
writeln('Maximum f(x1,x2)=',f(x1,x2),' for x1=',x1,' and x2=',x2);
// вывод результатов – значения функции и аргумента

end; // конец поиска максимума

readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

1. Функция $f(x1, x2) = 3x_1^2 + 2x_2^2 + x_1 + 2x_2$ (рис. 59),

```
Minimisation? (yes - 1, no - 0):1
a1=-2
a2=2
b1=-2
b2=2
eps=0.000001
Minimum f(x1,x2)=-5.8333333333324844E-001
```

for x1=-1.6666654477459131E-001
and x2=-5.0000014186095476E-001

2. Функция $f(x1, x2) = (x_1 - 2)^2 + (x_2 - 1)^2 + x_1 + 2x_2$ (рис. 14),

Minimisation? (yes - 1, no - 0):1
a1=-5
a2=5
b1=-5
b2=5
eps=0.000001
Minimum f(x1,x2)= 2.7500000000000635E+000
for x1= 1.4999999294868336E+000
and x2=-2.4232744086967981E-007

3. Функция $f(x1, x2) = -3(x_1 - 2)^2 - 4(x_2 - 3)^2$ (max) (рис. 60),

Minimisation? (yes - 1, no - 0):0
a1=-5
a2=5
b1=-5
b2=5
eps=0.000001
Maximum f(x1,x2)=-4.8094368254774806E-014
for x1= 2.0000000484654885E+000
and x2= 3.0000001013011075E+000

4. Функция $f(x1, x2) = -2(x_1 - 2)^2 - 8(x_2 - 1)^2 + x_1 + 2x_2$ (max) (рис. 61),

Minimisation? (yes - 1, no - 0):0
a1=-5
a2=5
b1=-5
b2=5

eps=0.000001
Maximum f(x1,x2)= 4.2499999999998970E+000
for x1= 2.2499997730665293E+000
and x2= 1.1250000076969853E+000

5. Функция $f(x1, x2) = 2\left(3 - x_1\right)^2 + 8\left(x_1^2 - 2x_2\right)^2$ (рис. 37)

Minimisation? (yes - 1, no - 0):1
a1=-10
a2=10
b1=-10
b2=10
eps=0.000001
Minimum f(x1,x2)= 1.8517400340515578E-008
for x1= 2.9999037804602660E+000
and x2= 4.4997111691724676E+000

6. Функция $f(x1, x2) = 2x_1 - 3.5x_2 + \exp\left(x_1^2 + x_2^2\right)$ (рис. 62).

Minimisation? (yes - 1, no - 0):1
a1=-5
a2=5
b1=-5
b2=5
eps=0.000001
Minimum f(x1,x2)=-1.3801176989123518E+000
for x1=-4.4588783291137346E-001
and x2= 7.8030287375029861E-001

7. Функция $f(x1, x2) = x_1 - 1.4x_2 + \exp\left(0.01x_1^2 + 0.11x_2^2\right)$ (рис. 63).

Minimisation? (yes - 1, no - 0):1
a1=-15
a2=15
b1=-15

b2=15
eps=0.000001
Minimum f(x1,x2)=-8.8124053629384029E+000
for x1=-1.1251464871656140E+001
and x2= 1.4320043266373876E+000

v23 § 4. Метод Нелдера – Мида (подвижных многогранников)

Другие названия метода – подвижных многогранников или деформируемых многогранников. Метод предложен Нелдером и Мидом (Nelder J.A. Mead R.). Метод в общем случае применѝм и широко используется для оптимизации функции многих переменных (больше двух). В основе алгоритма лежит построение выпуклого многогранника, количество вершин которого совпадает с количеством аргументов целевой функции (точнее – на одну больше). В настоящем пособии рассматривается вариант функции двух переменных; в этом случае многогранник представляет собой треугольник на плоскости.

На первой итерации, исходя из заданной начальной точки, строится равносторонний (например) треугольник, то есть находятся остальные его две вершины. Далее производится шаг по определённым критериям (операция отражения). полученная точка может корректироваться: шаг может вытягиваться в сторону наиболее быстрого понижения функции или сжиматься при противоположной ситуации (операции растяжения и сжатия). При нарушении монотонности производится операция редукции – уменьшения сторон треугольника. При всех этих операциях первоначальный вид треугольника меняется.

Параметры отражения α, сжатия β, растяжения γ задаются пользователем, исходя из предварительной информации о целевой функции. Также задаётся точность и параметр построения начального треугольника.

В начале каждой итерации среди вершин треугольника выбирается «наилучшая» по значению функции точка xb (при минимизации – минимальная), «наихудшая» xw и точка xs, стоящая на «втором месте» после xw (в случае функции двух переменных это просто третья точка треугольника). Далее рассчитывается «центр

тяжести» всех вершин многогранника (для общего случая), за исключением наихудшей:

$$\text{xc} = \frac{1}{n}\left(\sum_{j=1}^{n+1} \left(x_j \right) - \text{xw} \right) \tag{88}$$

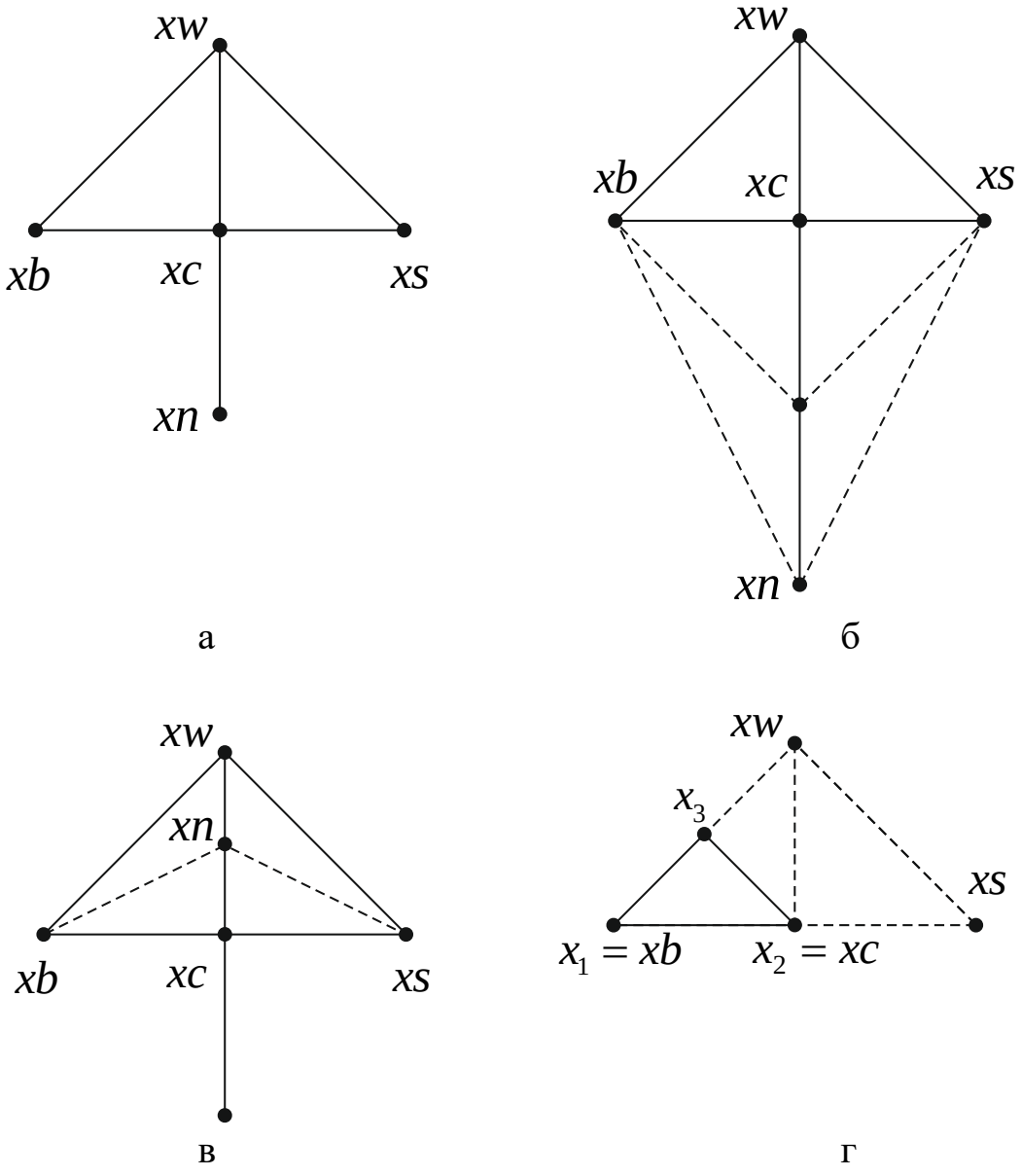


Рис. 66. Операции метода Нелдера-Мида: а) – отражение, б) – растяжение, в) – сжатие, г) – редукция (см. текст)

После этого проводится операция **отражения** наихудшей вершины и находится новая точка:

$$\mathrm{xn}_{mirror} = \mathrm{xc} + \alpha\left(\mathrm{xc} - \mathrm{xw}\right) \quad (89)$$

Если полученная точка меньше наилучшей, проводится операция **растяжения**:

$$\mathrm{xn}_{stretch} = \mathrm{xc} + \gamma\left(\mathrm{xn}_{mirror} - \mathrm{xc}\right) \quad (90)$$

Если же полученная перед этим точка лежит между xw и xs, проводится операция **сжатия**:

$$\mathrm{xn}_{compr} = \mathrm{xc} + \beta\left(\mathrm{xw} - \mathrm{xc}\right) \quad (91)$$

Если же полученная после отражения точка лежит между xb и xs наихудшая вершина просто заменяется на найденную новую точку.

При значении функции в новой точке худшем, чем в xw, производится **редукция**:

$$x_j = \mathrm{xb} + 0.5\left(x_j - \mathrm{xn}\right) \quad (92)$$

Критерием выхода из итерационного цикла служит величина:

$$\sigma = \left(\frac{1}{n+1} \sum_{j=1}^{n+1} \left(f\left(x_j\right) - f\left(xc\right)\right)^2 \right)^{\frac{1}{2}} \quad (93)$$

Значения коэффициентов. рекомендованных разными авторами [2, с. 141]:

Нелдер и Мид (Nelder Mead) – $\alpha = 1; \beta = 0.5; \gamma = 2$

Павиани (Paviani) – $\alpha = 1; 0.4 \le \beta \le 0.6; 2.8 \le \gamma \le 3$

Паркинсон и Хатчинсон (Parkinson Hutchinson) – $\alpha = 2; \beta = 0.25; \gamma = 2.5$

Программа, реализующая метод Нелдера-Мида для функции двух переменных, представлена на языке Lazarus (консольное приложение) Для апробации программы использовались функции (87).

Программа метода Нелдера-Мида

```
program NelMid;
var
x1,x2:array [1..3] of real;
// описание двух массивов координат по осям x1 x2 для трёх точек
// Далее описываются по две координаты для указанных точек:
xb1,xb2, // наилучшая (best) по значению функции
xw1,xw2, // наихудшая (worst) по значению функции
```

```
xs1,xs2, // вторая по наихудшему значению (second) функции
xc1,xc2, // центр тяжести (center) всех точек, кроме наихудшей
xn1,xn2, // новая точка, как результат отражения,
// растяжения или сжатия
h, // шаг построения исходного многогранника (треугольника)
al, // коэффициент отражения
bt, // коэффициент сжатия
gm, // коэффициент растяжения
min, // вспомогательная переменная минимум
// (для поиска минимума или максимума)
eps, // точность
sgm // переменная сигма, используется для сравнения
// с точностью эпсилон
:real; // тип указанных переменных
i,k,imin,kf:integer;
// описание индекса i, служащего для организации цикла;
// метки вида экстремума: k=1 – минимум,
// k=0 – максимум;
// номера экстремального значения imin (временного максимума
// или минимума) при поиске наименьшего или
// наибольшего значения;
// номера выбора коэффициентов по Нелдеру-Миду (kf=1),
// Павиани (kf=2) или Паркинсону и Хатчинсону (kf=3)

//Далее описываются 7 функций, по которым проводилась
// апробация программы, 6 из которых закомментированы
// скобками { и }

{function f(x1,x2:real):real;                    // №1
begin f:=3*sqr(x1)+2*sqr(x2)+x1+ 2*x2;end;}

{function f(x1,x2:real):real;                    //№ 2
begin f:=sqr(x1-2)+sqr(x2-1)+x1+2*x2;end;}

{function f(x1,x2:real):real;                    // №3; max
begin f:=-3*sqr(x1-2)-4*sqr(x2-3);end;}
```

```
{function f(x1,x2:real):real;                    // №4; max
begin f:=-2*sqr(x1-2)-8*sqr(x2-1)+x1+2*x2;end;}

{function f(x1,x2:real):real;                    // №5
begin f:=2*sqr(3-x1)+8*sqr(sqr(x1)-2*x2);end;}

{function f(x1,x2:real):real;                    // №6
begin f:=2*x1-3.5*x2+exp(sqr(x1)+sqr(x2));end;}

function f(x1,x2:real):real;                     // №7
begin f:=x1-1.4*x2+exp(0.01*sqr(x1)+0.11*sqr(x2));end;

// Далее описывается процедура ранжирования, то есть
// расстановки координат точек в порядке возрастания
// значений функции в них
procedure rang();
var i,j:integer;
// описание внутренних для процедуры индексов i и j
// (в основной программе они не используются)
begin // начало тела процедуры
for j:=1 to 2 do
// задание внешнего цикла последовательных поисков минимума
// среди каждый раз уменьшающегося количества
// элементов массива
begin
min:=f(x1[j],x2[j]);
imin:=j;
// первоначальное присваивание переменной min значения
// функции для j-тых координат и местоположению
// imin – соответствующего индекса
for i:=j+1 to 3 do
// поиск минимума среди всё уменьшающегося количества
// (это обеспечивает переменное значение i:=j+1)
// элементов массива
if f(x1[i],x2[i])<min
then
// если найдено меньшее значение функции, то минимум и
```

```
// его местоположение получают новые значения:
begin
min:=f(x1[i],x2[i]);
imin:=i;
end;
// Далее идёт перестановка координат найденного минимума
// функции  и координат первого элемента оставшегося массива;
// min используется в новом качестве вспомогательной
// переменной при перестановках:
min:=x1[j];
x1[j]:=x1[imin];
x1[imin]:=min;
min:=x2[j];
x2[j]:=x2[imin];
x2[imin]:=min;
end; // окончание ранжирования

// Далее следует переприсваивание наилучшим (xb), вторым (xs)
// и наихудшим (xw) точкам соответствующих значений в
// соответствии с видом искомого экстремума:
if k=1 then // для минимума
begin
xb1:=x1[1];
xb2:=x2[1];
xs1:=x1[2];
xs2:=x2[2];
xw1:=x1[3];
xw2:=x2[3];
end
else // для максимума
begin
xb1:=x1[3];
xb2:=x2[3];
xs1:=x1[2];
xs2:=x2[2];
xw1:=x1[1];
xw2:=x2[1];
```

```
end;
end; // конец процедуры

// Далее описывается процедура определения координат
// центра тяжести (center) (см. (88))
procedure cntr();
var i:integer;
// описание внутреннего для процедуры индекса i
// (в основной программе он не используется)
begin
xc1:=0;
xc2:=0;
for i:=1 to 3 do
begin
xc1:=xc1+x1[i];
xc2:=xc2+x2[i];
end;
xc1:=xc1-xw1;
xc2:=xc2-xw2;
xc1:=xc1/2;
xc2:=xc2/2;
end;

// Далее описывается процедура отражения (mirror), при этом
// определяются координаты новой точки (см. (89))
procedure mirr();
begin
xn1:=xc1+al*(xc1-xw1);
xn2:=xc2+al*(xc2-xw2);
end;

// Далее описывается процедура растяжения (stretch), при этом
// определяются координаты новой точки (см. (90))
procedure stch();
var tp1,tp2:real;
// описание внутренних для процедуры переменных
// (в основной программе они не используется)
```

```
begin
tp1:=xc1+gm*(xn1-xc1);
tp2:=xc2+gm*(xn2-xc2);
if k=1 then // при минимизации
begin
if f(tp1,tp2)<f(xb1,xb2)
then
begin
xn1:=tp1;
xn2:=tp2;
end;
end
else // при поиске максимума
begin
if f(tp1,tp2)>f(xb1,xb2)
then
begin
xn1:=tp1;
xn2:=tp2;
end;
end;
end;

// Далее описывается процедура сжатия (compression), при этом
// определяются координаты новой точки (см. (91))
procedure cmpr();
begin
xn1:=xc1+bt*(xw1-xc1);
xn2:=xc2+bt*(xw2-xc2);
end;

// Далее описывается процедура редукции (reduction), при этом
// определяются координаты нового многогранника (см. (92))
procedure rdcn();
begin
x1[1]:=xb1+0.5*(x1[1]-xb1);
x1[2]:=xb1+0.5*(x1[2]-xb1);
```

```
x1[3]:=xb1+0.5*(x1[3]-xb1);
x2[1]:=xb2+0.5*(x2[1]-xb2);
x2[2]:=xb2+0.5*(x2[2]-xb2);
x2[3]:=xb2+0.5*(x2[3]-xb2);
end;

// Начало основной программы
begin
// определение пользователем вида экстремума
// (минимум или максимум):
write('Minimisation? (yes - 1, no - 0):');
readln(k); // ввод метки экстремума
write('x1=');readln(x1[1]);
write('x2=');readln(x2[1]);
// ввод координат начальной точки; при этом значения
// записываются в ячейки памяти первой точки многогранника
// (треугольника) x1[1] и x2[1]
write('h=');readln(h);
// ввод шага построения многогранника (треугольника)
write('eps=');readln(eps);
// ввод точности

writeln('Coefficients from Nelder & Mead - 1,');
write('from Paviani - 2, from Parkinson Hutchinson - 3:');
readln(kf);
// ввод ключа назначения коэффициентов, рекомендованных
// разными авторами
// Нелдер и Мид (Nelder Mead)
if kf=1 then
begin
al:=1;
bt:=0.5;
gm:=2;
end;

// Павиани (Paviani)
if kf=2 then
```

```
begin
al:=1;
bt:=0.5; // 0.4<=bt<=0.6
gm:=2.9; // 2.8<=gm<=3
end;

// Паркинсон и Хатчинсон (Parkinson Hutchinson)
if kf=3 then
begin
al:=2;
bt:=0.25;
gm:=2.5;
end;

// Расчёт второй и третьей точки первоначального
// равностороннего треугольника (в общем случае
// это многогранник):
x1[2]:=x1[1]+h;
x2[2]:=x2[1];
x1[3]:=x1[1]+h/2;
x2[3]:=x2[1]+sqrt(5)*h/2;

// начало оптимизации; при выполнении этого условия
// имеет место минимум:
if k=1 then
begin // начало минимизации

repeat // открытие итерационного цикла
rang(); // ранжирование вершин многогранника (треугольника)
// Далее следует расчёт величины sgm (93), которая служит
// критерием окончания итерационного процесса:
sgm:=0;
for i:=1 to 3 do
sgm:=sgm+sqr(f(x1[i],x2[i])-f(xb1,xb2));
sgm:=sqrt(sgm/3);
cntr(); // определение центра тяжести
mirr(); // операция отражения
```

```
if f(xn1,xn2)<f(xb1,xb2)
then
stch() // операция растяжения
else
if (f(xn1,xn2)>f(xs1,xs2))and(f(xn1,xn2)<f(xw1,xw2))
then
cmpr() // операция сжатия
else
begin
rdcn(); // операция редукции
continue; // досрочный возврат в цикл; часть алгоритма,
// написанная до конца цикла при этом не выполняется
end;
// Далее – переназначение точек согласно алгоритму:
xw1:=xn1;
xw2:=xn2;
x1[3]:=xb1;
x2[3]:=xb2;
x1[2]:=xs1;
x2[2]:=xs2;
x1[1]:=xw1;
x2[1]:=xw2;
until sgm<eps; // критерий выхода из итерационного цикла
writeln('Minimum f(x1,x2)=',f(xn1,xn2),' for x1=',xn1,' and x2=',xn2);
// вывод результатов – значения функции и аргумента

end // конец минимизации
else
// при выполнении обратного условия имеет место максимум
begin// начало поиска максимума

repeat // открытие итерационного цикла
rang(); // ранжирование вершин многогранника (треугольника)
// Далее следует расчёт величины sgm (93), которая служит
// критерием окончания итерационного процесса:
sgm:=0;
for i:=1 to 3 do
```

```
sgm:=sgm+sqr(f(x1[i],x2[i])-f(xb1,xb2));
sgm:=sqrt(sgm/3);
cntr(); // определение центра тяжести
mirr(); // операция отражения
if f(xn1,xn2)>f(xb1,xb2)
then
stch() // операция растяжения
else
if (f(xn1,xn2)<f(xs1,xs2))and(f(xn1,xn2)>f(xw1,xw2))
then
cmpr()// операция сжатия
else
begin
rdcn(); // операция редукции
continue; // досрочный возврат в цикл; часть алгоритма,
// написанная до конца цикла при этом не выполняется
end;
// Далее – переназначение точек согласно алгоритму:
xw1:=xn1;
xw2:=xn2;
x1[3]:=xb1;
x2[3]:=xb2;
x1[2]:=xs1;
x2[2]:=xs2;
x1[1]:=xw1;
x2[1]:=xw2;
until sgm<eps; // критерий выхода из итерационного цикла
writeln
('Maximum f(x1,x2)=',f(xn1,xn2),' for x1=',xn1,' and x2=',xn2);
// вывод результатов – значения функции и аргумента

end; // конец поиска максимума

readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

1. Функция $f(x1, x2) = 3x_1^2 + 2x_2^2 + x_1 + 2x_2$ (рис. 59),

Minimisation? (yes - 1, no - 0):1
x1=5
x2=3
h=0.2
eps=0.000001
Minimum f(x1,x2)=-5.8333297864661726E-001
for x1=-1.6632286105082794E-001
and x2=-4.9999368289653134E-001

2. Функция $f(x1, x2) = (x_1 - 2)^2 + (x_2 - 1)^2 + x_1 + 2x_2$ (рис. 14),

Minimisation? (yes - 1, no - 0):1
x1=5
x2=3
h=0.2
eps=0.000001
Minimum f(x1,x2)= 2.7500038634769171E+000
for x1= 1.4999045582148183E+000
and x2=-1.9632543856111712E-003

3. Функция $f(x1, x2) = -3(x_1 - 2)^2 - 4(x_2 - 3)^2$ (max) (рис. 60),

Minimisation? (yes - 1, no - 0):0
x1=0.1
x2=0.1
h=0.2
eps=0.000001
Maximum f(x1,x2)=-3.2310453165753202E-006
for x1= 2.0010375976562518E+000
and x2= 2.9999825455750755E+000

4. Функция $f(x1, x2) = -2\left(x_1 - 2\right)^2 - 8\left(x_2 - 1\right)^2 + x_1 + 2x_2$ (max) (рис. 61),

Minimisation? (yes - 1, no - 0):0
x1=0.1
x2=0.1
h=0.2
eps=0.000001
Maximum f(x1,x2)= 4.2499998309270950E+000
for x1= 2.2502877756953232E+000
and x2= 1.1250207460926971E+000

5. Функция $f(x1, x2) = 2\left(3 - x_1\right)^2 + 8\left(x_1^2 - 2x_2\right)^2$ (рис. 37)

Minimisation? (yes - 1, no - 0):1
x1=-4.5
x2=-4.5
h=0.2
eps=0.000001
Minimum f(x1,x2)= 1.2938630934642150E-006
for x1= 3.0002104844306992E+000
and x2= 4.5004374026722900E+000

6. Функция $f(x1, x2) = 2x_1 - 3.5x_2 + \exp\left(x_1^2 + x_2^2\right)$ (рис. 62),

Minimisation? (yes - 1, no - 0):1
x1=-2
x2=-2
h=0.1
eps=0.000001
Minimum f(x1,x2)=-1.3801175723162387E+000
for x1=-4.4567553489660872E-001
and x2= 7.8033005158678104E-001

7. Функция $f(x1, x2) = x_1 - 1.4x_2 + \exp\left(0.01x_1^2 + 0.11x_2^2\right)$ (рис. 63).

Minimisation? (yes - 1, no - 0):1
x1=-1
x2=-1
h=0.2
eps=0.000001
Minimum f(x1,x2)=-8.8124048812397078E+000
for x1=-1.1251915113103248E+001
and x2= 1.4327071236715598E+000

v24 § 5. Метод градиентного спуска

Как известно, вектор градиента направлен в сторону наибольшего возрастания функции, это используется для определения направления в общей стратегии спуска. Если шаг в этом направлении t_k задаётся пользователем, то формула поиска минимума выглядит так:

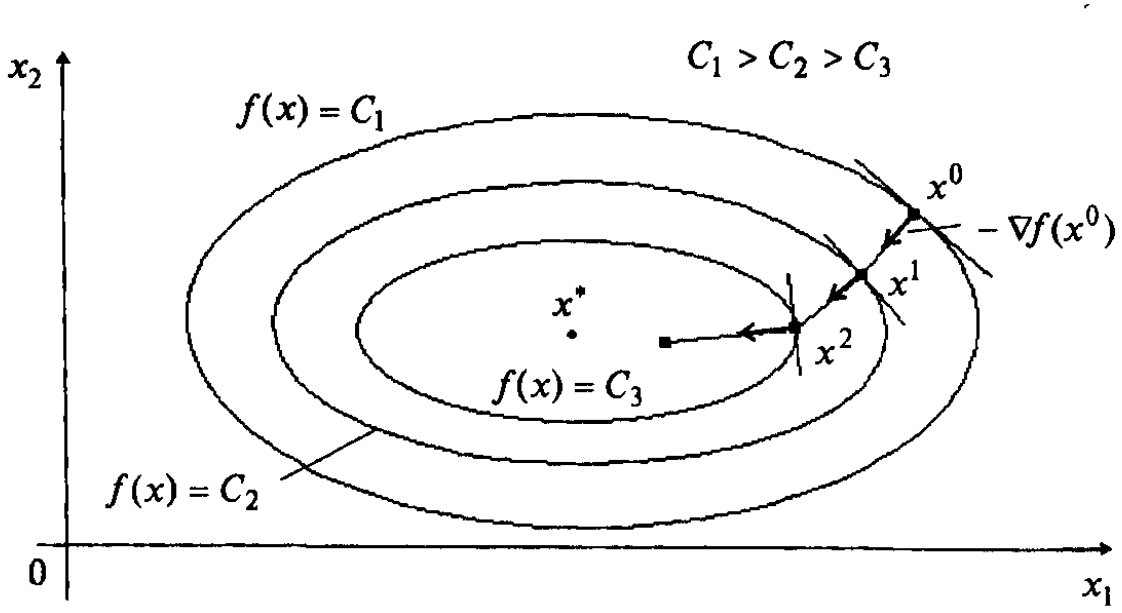


Рис. 67. Геометрическая интерпретация метода градиентного спуска с постоянным шагом [2, с. 179]

$$x^{(k+1)} = x^{(k)} - t_k \nabla f\left(x^{(k)}\right), \tag{94}$$

а максимума – так:

$$x^{(k+1)} = x^{(k)} + t_k \nabla f\left(x^{(k)}\right) \tag{95}$$

(здесь k – номер итерации). Таким образом, поиск минимума осуществляется по антиградиенту.

Формула для расчёта проекций градиента по осям:

$$\nabla f\left(x_j^{(k)}\right) = \frac{\left(\frac{\partial f}{\partial x_j^{(k)}}\right)}{d} \tag{96}$$

где

$$d = \sqrt{\sum_{j=1}^{n}\left(\frac{\partial f}{\partial x_j^{(k)}}\right)^2} \tag{97}$$

Геометрическая интерпретация метода показана на рис. 67.

Программа, реализующая метод градиентного спуска с постоянным шагом, представлена на языке Lazarus (консольное приложение). Для апробации программы использовались функции (87).

Программа метода градиентного спуска

```
program GradientSpusk;
var
x10,x1,x20,x2,
// описание координат приближения экстремума по
// двум осям x1 и x2:
// x10 и  x20 – предыдущее приближение,
// x1 и x2 – последующее приближение;
eps,al,d:real;
// описание точности eps;
// описание шага al;
// описание проекции градиента или антиградиента d
// на координатную плоскость аргументов;
k:integer;
```

```
// описание метки вида экстремума: k=1 – минимум,
// k=0 – максимум

//Далее описываются 7 функций, по которым проводилась
// апробация программы, 6 из которых закомментированы
// скобками { и }; f – функции, f1 и f2 – частные
//производные по соответствующим аргументам

{function f(x1,x2:real):real;                    // №1
begin f:=3*sqr(x1)+2*sqr(x2)+x1+ 2*x2;end;
function f1(x1,x2:real):real;
begin f1:=6*x1+1;end;
function f2(x1,x2:real):real;
begin f2:=4*x2+2;end;}

{function f(x1,x2:real):real;                    //№ 2
begin f:=sqr(x1-2)+sqr(x2-1)+x1+2*x2;end;
function f1(x1,x2:real):real;
begin f1:=2*(x1-2)+1;end;
function f2(x1,x2:real):real;
begin f2:=2*(x2-1)+2;end;}

{function f(x1,x2:real):real;                    // №3; max
begin f:=-3*sqr(x1-2)-4*sqr(x2-3);end;
function f1(x1,x2:real):real;
begin f1:=-6*(x1-2);end;
function f2(x1,x2:real):real;
begin f2:=-8*(x2-3);end;}

{function f(x1,x2:real):real;                    // №4; max
begin f:=-2*sqr(x1-2)-8*sqr(x2-1)+x1+2*x2;end;
function f1(x1,x2:real):real;
begin f1:=-4*(x1-2)+1;end;
function f2(x1,x2:real):real;
begin f2:=-16*(x2-1)+2;end;}

{function f(x1,x2:real):real;                    // №5
```

```
begin f:=2*sqr(3-x1)+8*sqr(sqr(x1)-2*x2);end;
function f1(x1,x2:real):real;
begin f1:=-4*(3-x1)+32*(sqr(x1)-2*x2)*x1;end;
function f2(x1,x2:real):real;
begin f2:=-32*(sqr(x1)-2*x2);end;}

{function f(x1,x2:real):real;                                    // №6
begin f:=2*x1-3.5*x2+exp(sqr(x1)+sqr(x2));end;
function f1(x1,x2:real):real;
begin f1:=2+2*x1*exp(sqr(x1)+sqr(x2));end;
function f2(x1,x2:real):real;
begin f2:=-3.5+2*x2*exp(sqr(x1)+sqr(x2));end;}

function f(x1,x2:real):real;                                     // №7
begin f:=x1-1.4*x2+exp(0.01*sqr(x1)+0.11*sqr(x2));end;
function f1(x1,x2:real):real;
begin f1:=1+0.02*x1*exp(0.01*sqr(x1)+0.11*sqr(x2));end;
function f2(x1,x2:real):real;
begin f2:=-1.4+0.22*x2*exp(0.01*sqr(x1)+0.11*sqr(x2));end;

// Начало основной программы
begin
// определение пользователем вида экстремума
// (минимум или максимум):
write('Minimisation? (yes - 1, no - 0):');
readln(k); // ввод метки экстремума

write('x10=');readln(x10);
write('x20=');readln(x20);
// ввод координат экстремума начального (предыдущего)
// приближения;
write('al=');readln(al);
// ввод шага
write('eps=');readln(eps);
// ввод точности

// начало оптимизации; при выполнении этого условия
```

```
// имеет место минимум:
if k=1 then
begin // начало минимизации

repeat // открытие итерационного цикла
d:=sqrt(sqr(f1(x10,x20))+sqr(f2(x10,x20)));
// расчёт проекции антиградиента
x1:=x10-al*f1(x10,x20)/d;
x2:=x20-al*f2(x10,x20)/d;
// расчёт новых точек спуска по антиградиенту
// Проверка условия дробления шага:
if f(x1,x2)>f(x10,x20)
then
al:=al/2 // дробления шага и заход на следующий цикл
// без переприсваивания
else // иначе – переприсваивание и заход на следующий цикл
begin
x10:=x1;
x20:=x2;
end;
until al<eps;
// критерий выхода из итерационного цикла
writeln('Minimum f(x1,x2)=',f(x1,x2),' for x1=',x1,' and x2=',x2);
// вывод результатов – значения функции и аргумента

end // конец минимизации
else
// при выполнении обратного условия имеет место максимум
begin// начало поиска максимума

repeat // открытие итерационного цикла
d:=sqrt(sqr(f1(x10,x20))+sqr(f2(x10,x20)));
// расчёт проекции градиента
x1:=x10+al*f1(x10,x20)/d;
x2:=x20+al*f2(x10,x20)/d;
// расчёт новых точек спуска по градиенту
// Проверка условия дробления шага:
```

```
if f(x1,x2)<f(x10,x20)
then
al:=al/2// дробления шага и заход на следующий цикл
// без переприсваивания
else // иначе – переприсваивание и заход на следующий цикл
begin
x10:=x1;
x20:=x2;
end;
until al<eps;
// критерий выхода из итерационного цикла
writeln('Maximum f(x1,x2)=',f(x1,x2),' for x1=',x1,' and x2=',x2);
// вывод результатов – значения функции и аргумента

end; // конец поиска максимума

readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

1. Функция $f(x1, x2) = 3x_1^2 + 2x_2^2 + x_1 + 2x_2$ (рис. 59),

$$\frac{\partial f}{\partial x_1} = 6x_1 + 1; \quad \frac{\partial f}{\partial x_2} = 4x_2 + 2;$$

```
Minimisation? (yes - 1, no - 0):1
x10=5
x20=3
al=0.1
eps=0.000001
Minimum f(x1,x2)=-5.8333333333151727E-001
for x1=-1.6666744471972481E-001
and x2=-5.0000000108576781E-001
```

2. Функция $f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2 + x_1 + 2x_2$ (рис. 14),

$$\frac{\partial f}{\partial x_1} = 2(x_1 - 2) + 1; \quad \frac{\partial f}{\partial x_2} = 2(x_2 - 1) + 2;$$

Minimisation? (yes - 1, no - 0):1
x10=5
x20=3
al=0.1
eps=0.000001
Minimum f(x1,x2)= 2.7500000000010520E+000
for x1= 1.4999992211939910E+000
and x2=-6.6754800779508075E-007

3. Функция $f(x_1, x_2) = -3(x_1 - 2)^2 - 4(x_2 - 3)^2$ (max) (рис. 60),

$$\frac{\partial f}{\partial x_1} = -6(x_1 - 2); \quad \frac{\partial f}{\partial x_2} = -8(x_2 - 3)$$

Minimisation? (yes - 1, no - 0):0
x10=0.1
x20=0.1
al=0.1
eps=0.000001
Maximum f(x1,x2)=-4.5314363950917557E-012
for x1= 1.9999999964335995E+000
and x2= 3.0000010643540573E+000

4. Функция . $f(x_1, x_2) = -2(x_1 - 2)^2 - 8(x_2 - 1)^2 + x_1 + 2x_2$ (max)
(рис. 61),

$$\frac{\partial f}{\partial x_1} = -4(x_1 - 2) + 1; \quad \frac{\partial f}{\partial x_1} = -16(x_2 - 1) + 2$$

Minimisation? (yes - 1, no - 0):0
x10=0.1
x20=0.1

al=0.1
eps=0.000001
Maximum f(x1,x2)= 4.2499999999974065E+000
for x1= 2.2499992126692732E+000
and x2= 1.1250004112897602E+000

5. Функция $f(x1, x2) = 2\left(3 - x_1\right)^2 + 8\left(x_1^2 - 2x_2\right)^2$ (рис. 37)

$$\frac{\partial f}{\partial x_1} = -4\left(3 - x_1\right) + 32\left(x_1^2 - 2x_2\right)x_1;$$

$$\frac{\partial f}{\partial x_2} = -32\left(x_1^2 - 2x_2\right)$$

Minimisation? (yes - 1, no - 0):1
x10=-4.5
x20=-4.5
al=0.1
eps=0.000001
Minimum f(x1,x2)= 7.1008032012561376E-010
for x1= 2.9999982118843724E+000
and x2= 4.4999993250230270E+000

6. Функция $f(x1, x2) = 2x_1 - 3.5x_2 + \exp\left(x_1^2 + x_2^2\right)$ (рис. 62).

$$\frac{\partial f}{\partial x_1} = 2 + 2x_1 \exp\left(x_1^2 + x_2^2\right);$$

$$\frac{\partial f}{\partial x_2} = -3.5 + 2x_2 \exp\left(x_1^2 + x_2^2\right)$$

Minimisation? (yes - 1, no - 0):1
x10=-2
x20=-2
al=0.1
eps=0.000001
Minimum f(x1,x2)=-1.3801176989086614E+000
for x1=-4.4588788601327312E-001

and x2= 7.8030385411552872E-001

7. Функция $f(x1, x2) = x_1 - 1.4x_2 + \exp\left(0.01x_1^2 + 0.11x_2^2\right)$ (рис. 63)

$$\frac{\partial f}{\partial x_1} = 1 + 0.02x_1 \exp\left(0.01x_1^2 + 0.11x_2^2\right);$$

$$\frac{\partial f}{\partial x_2} = -1.4 + 0.22x_2 \exp\left(0.01x_1^2 + 0.11x_2^2\right)$$

Minimisation? (yes - 1, no - 0):1
x10=-1
x20=-1
al=0.1
eps=0.000001
Minimum f(x1,x2)=-8.8124053629376959E+000
for x1=-1.1251465016180044E+001
and x2= 1.4320035222442191E+000

v25 § 6. Метод наискорейшего спуска

Этот метод представляет собой метод градиентного спуска с использованием золотого сечения в плоскостях, перпендикулярных координатной плоскости аргументов. При этом направления последовательных итераций будут ортогональны (рис. 68).

Метод во многом схож с предыдущим, но здесь величина шага фактически определяется результатом одномерной оптимизации. Метод значительно быстрее сходится, так как требует меньше итераций при прочих равных условиях.

Программа, реализующая метод градиентного спуска с использованием золотого сечения, представлена на языке Lazarus (консольное приложение) Для апробации программы использовались функции (87):

Программа метода наискорейшего спуска

```
program GradientSpuskZolSech3;
var
x10a,x20a,x1a,x2a,
```

// описание координат приближения экстремума по
// двум осям x1 и x2:
// x10a и x20a – предыдущее приближение,
// x1a и x2a – последующее приближение;
x10b,x20b, x1b,x2b,
// описание глобальных переменных x1b и x2b, используемых
// при одномерной оптимизации методом золотого сечения в
// соответствующей секущей плоскости; они представляют
// собой правые концы отрезков по отношению к
// левым концам x1a и x2a;
// x10b и x20b – предыдущее приближение,
// x1b и x2b – последующее приближение;

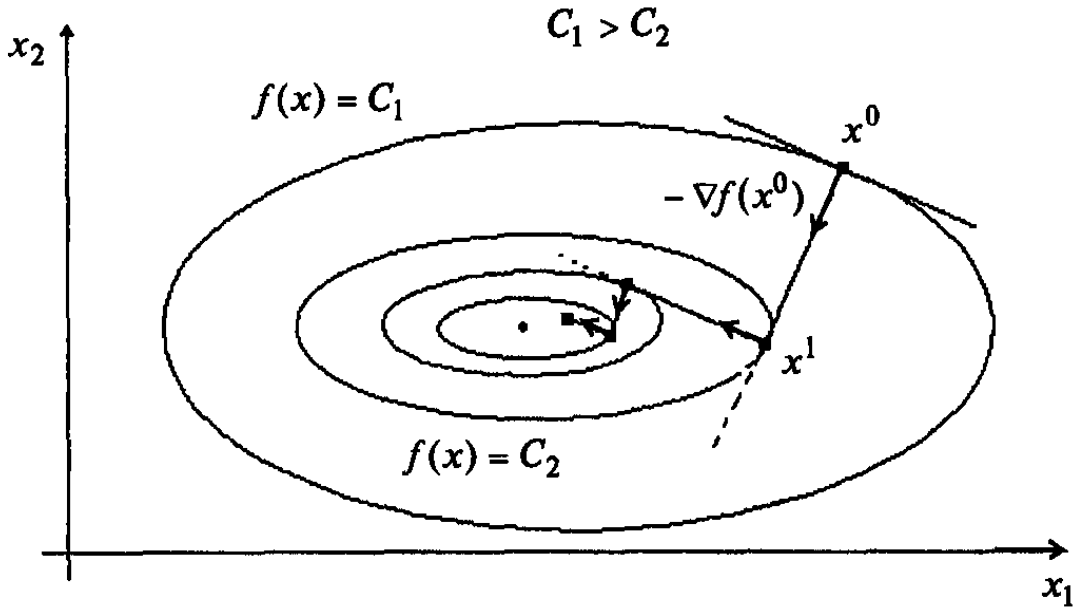


Рис. 68. Геометрическая интерпретация метода наискорейшего градиентного спуска. Направления последовательных итераций ортогональны [2, с. 186]

x11,x21,x12,x22,
// описание глобальных переменных, используемых
// при одномерной оптимизации методом золотого сечения в
// соответствующей секущей плоскости; они представляют
// собой внутренние точки двустороннего золотого сечения
eps,al,d:real;

```
// описание точности eps;
// описание шага al;
// описание проекции градиента или антиградиента d
// на координатную плоскость аргументов;
k:integer;
// описание метки вида экстремума: k=1 – минимум,
// k=0 – максимум

//Далее описываются 7 функций, по которым проводилась
// апробация программы, 6 из которых закомментированы
// скобками { и }; f – функции, f1 и f2 – частные
//производные по соответствующим аргументам

{function f(x1,x2:real):real;                    // №1
begin f:=3*sqr(x1)+2*sqr(x2)+x1+ 2*x2;end;
function f1(x1,x2:real):real;
begin f1:=6*x1+1;end;
function f2(x1,x2:real):real;
begin f2:=4*x2+2;end;}

{function f(x1,x2:real):real;                    //№ 2
begin f:=sqr(x1-2)+sqr(x2-1)+x1+2*x2;end;
function f1(x1,x2:real):real;
begin f1:=2*(x1-2)+1;end;
function f2(x1,x2:real):real;
begin f2:=2*(x2-1)+2;end;}

{function f(x1,x2:real):real;                    // №3; max
begin f:=-3*sqr(x1-2)-4*sqr(x2-3);end;
function f1(x1,x2:real):real;
begin f1:=-6*(x1-2);end;
function f2(x1,x2:real):real;
begin f2:=-8*(x2-3);end;}

{function f(x1,x2:real):real;                    // №4; max
begin f:=-2*sqr(x1-2)-8*sqr(x2-1)+x1+2*x2;end;
function f1(x1,x2:real):real;
```

```
begin f1:=-4*(x1-2)+1;end;
function f2(x1,x2:real):real;
begin f2:=-16*(x2-1)+2;end;}

{function f(x1,x2:real):real;                    // №5
begin f:=2*sqr(3-x1)+8*sqr(sqr(x1)-2*x2);end;
function f1(x1,x2:real):real;
begin f1:=-4*(3-x1)+32*(sqr(x1)-2*x2)*x1;end;
function f2(x1,x2:real):real;
begin f2:=-32*(sqr(x1)-2*x2);end;}

{function f(x1,x2:real):real;                    // №6
begin f:=2*x1-3.5*x2+exp(sqr(x1)+sqr(x2));end;
function f1(x1,x2:real):real;
begin f1:=2+2*x1*exp(sqr(x1)+sqr(x2));end;
function f2(x1,x2:real):real;
begin f2:=-3.5+2*x2*exp(sqr(x1)+sqr(x2));end;}

function f(x1,x2:real):real;                     // №7
begin f:=x1-1.4*x2+exp(0.01*sqr(x1)+0.11*sqr(x2));end;
function f1(x1,x2:real):real;
begin f1:=1+0.02*x1*exp(0.01*sqr(x1)+0.11*sqr(x2));end;
function f2(x1,x2:real):real;
begin f2:=-1.4+0.22*x2*exp(0.01*sqr(x1)+0.11*sqr(x2));end;

// Далее описываются вспомогательные функции оптимизации
// методом золотого сечения; zs1 – левая внутренняя точка,
// zs2 – правая внутренняя точка; a –  левый конец отрезка,
// b – правый конец отрезка
function zs1(a,b:real):real;
begin zs1:=a+(1-(sqrt(5)-1)/2)*(b-a);end;
function zs2(a,b:real):real;
begin zs2:=a+(sqrt(5)-1)/2*(b-a);end;

// Начало основной программы
begin
// определение пользователем вида экстремума
```

```
// (минимум или максимум):
write('Minimisation? (yes - 1, no - 0):');
readln(k); // ввод метки экстремума

write('x10=');readln(x1a);
write('x20=');readln(x2a);
// ввод координат экстремума начального (предыдущего)
// приближения; при этом значения записываются в ячейки
// памяти последующего приближения x1a и x2a, так как в начале
// итерационного цикла происходит переприсваивание

write('al=');readln(al);
// ввод шага
write('eps=');readln(eps);
// ввод точности

// начало оптимизации; при выполнении этого условия
// имеет место минимум:
if k=1 then
begin // начало минимизации

repeat // открытие итерационного цикла
x10a:=x1a;
x20a:=x2a;
// переприсваивание последующих и предыдущих координат
d:=sqrt(sqr(f1(x1a,x2a))+sqr(f2(x1a,x2a)));
// расчёт проекции антиградиента
x1b:=x1a-al/d*f1(x1a,x2a);
x2b:=x2a-al/d*f2(x1a,x2a);
// расчёт новых точек спуска по антиградиенту
// Далее идёт цикл для расчёта тех же точек с дроблением шага:
while (f(x1b,x2b)-f(x1a,x2a))/abs(f(x1a,x2a))>10 do
begin
al:=al/2;
x1b:=x1a-al/d*f1(x1a,x2a);
x2b:=x2a-al/d*f2(x1a,x2a);
end;
```

```
x10b:=x1b;
x20b:=x2b;
// полученные значения x1b и x2b запоминаем в x10b и x20b
x11:=zs1(x1a,x1b);
x21:=zs1(x2a,x2b);
x12:=zs2(x1a,x1b);
x22:=zs2(x2a,x2b);
// внутренние точки двустороннего золотого сечения
repeat
// цикл минимизации по методу золотого сечения в плоскости,
// перпендикулярной координатной плоскости x1 и x2
if f(x11,x21)<f(x12,x22) then
begin
x1b:=x12;
x2b:=x22;
x12:=x11;
x22:=x21;
x11:=zs1(x1a,x1b);
x21:=zs1(x2a,x2b);
end
else
begin
x1a:=x11;
x2a:=x21;
x11:=x12;
x21:=x22;x12:=zs2(x1a,x1b);
x22:=zs2(x2a,x2b);
end;
until abs(f(x1a,x2a)-f(x1b,x2b))<eps;
// конец минимизации по золотому сечению
x1a:=x11;
x2a:=x21;
// присваивание последующим координатам x1a и x2a
// найденных минимумов
// Далее происходит увеличение шага, если это целесообразно:
if abs(f(x1a,x2a)-f(x10b,x20b))<eps then
```

```
begin
al:=al*2;
end;

until abs(f(x1a,x2a)-f(x10a,x20a))<eps;
// критерий выхода из итерационного цикла
writeln('Minimum f(x1,x2)=',f(x1a,x2a),' for x1=',x1a,' and x2=',x2a);
// вывод результатов – значения функции и аргумента

end // конец минимизации
else
// при выполнении обратного условия имеет место максимум
begin// начало поиска максимума

repeat // открытие итерационного цикла
x10a:=x1a;
x20a:=x2a;
// переприсваивание последующих и предыдущих координат
d:=sqrt(sqr(f1(x1a,x2a))+sqr(f2(x1a,x2a)));
// расчёт проекции градиента
x1b:=x1a+al/d*f1(x1a,x2a);
x2b:=x2a+al/d*f2(x1a,x2a);
// расчёт новых точек спуска по антиградиенту
// Далее идёт цикл для расчёта тех же точек с дроблением шага:
while (f(x1b,x2b)-f(x1a,x2a))/abs(f(x1a,x2a))>10 do
begin
al:=al/2;
x1b:=x1a+al/d*f1(x1a,x2a);
x2b:=x2a+al/d*f2(x1a,x2a);
end;

x10b:=x1b;
x20b:=x2b;
// полученные значения x1b и x2b запоминаем в x10b и x20b
x11:=zs1(x1a,x1b);
x21:=zs1(x2a,x2b);
x12:=zs2(x1a,x1b);
```

```
x22:=zs2(x2a,x2b);
// внутренние точки двустороннего золотого сечения
repeat
// цикл поиска максимума по методу золотого сечения в
// плоскости, перпендикулярной координатной плоскости x1 и x2
if f(x11,x21)>f(x12,x22) then
begin
x1b:=x12;x2b:=x22;
x12:=x11;x22:=x21;
x11:=zs1(x1a,x1b);
x21:=zs1(x2a,x2b);
end
else
begin
x1a:=x11;
x2a:=x21;
x11:=x12;
x21:=x22;
x12:=zs2(x1a,x1b);
x22:=zs2(x2a,x2b);
end;
until abs(f(x1a,x2a)-f(x1b,x2b))<eps;
// конец поиска максимума по золотому сечению
x1a:=x11;
x2a:=x21;
// присваивание последующим координатам x1a и x2a
// найденных максимумов
// Далее происходит увеличение шага, если это целесообразно:
if abs(f(x1a,x2a)-f(x10b,x20b))<eps then
begin
al:=al*2;
end;

until abs(f(x1a,x2a)-f(x10a,x20a))<eps;
// критерий выхода из итерационного цикла
writeln('Maximum f(x1,x2)=',f(x1a,x2a),' for x1=',x1a,' and x2=',x2a);
// вывод результатов – значения функции и аргумента\
```

```
end; // конец поиска максимума

readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

1. Функция $f(x1, x2) = 3x_1^2 + 2x_2^2 + x_1 + 2x_2$ (рис. 59),

$$\frac{\partial f}{\partial x_1} = 6x_1 + 1; \quad \frac{\partial f}{\partial x_2} = 4x_2 + 2;$$

```
Minimisation? (yes - 1, no - 0):1
x10=5
x20=3
al=0.1
eps=0.000001
Minimum f(x1,x2)=-5.8333331833946822E-001
for x1=-1.6669372690828027E-001
and x2=-5.0007999092202449E-001
```

2. Функция $f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2 + x_1 + 2x_2$ (рис. 14),

$$\frac{\partial f}{\partial x_1} = 2(x_1 - 2) + 1; \quad \frac{\partial f}{\partial x_2} = 2(x_2 - 1) + 2;$$

```
Minimisation? (yes - 1, no - 0):1
x10=5
x20=3
al=0.1
eps=0.000001
Minimum f(x1,x2)= 2.7500000276501022E+000
for x1= 1.4998737483829836E+000
and x2=-1.0821567172835430E-004
```

3. Функция $f(x_1, x_2) = -3(x_1 - 2)^2 - 4(x_2 - 3)^2$ (max) (рис. 60),

$$\frac{\partial f}{\partial x_1} = -6(x_1 - 2); \quad \frac{\partial f}{\partial x_2} = -8(x_2 - 3)$$

Minimisation? (yes - 1, no - 0):0
x10=0.1
x20=0.1
al=0.1
eps=0.000001
Maximum f(x1,x2)=-2.9997156654710097E-008
for x1= 2.0000339678317123E+000
and x2= 3.0000814489347398E+000

4. Функция . $f(x_1, x_2) = -2(x_1 - 2)^2 - 8(x_2 - 1)^2 + x_1 + 2x_2$ (max) (рис. 61),

$$\frac{\partial f}{\partial x_1} = -4(x_1 - 2) + 1; \quad \frac{\partial f}{\partial x_1} = -16(x_2 - 1) + 2$$

Minimisation? (yes - 1, no - 0):0
x10=0.1
x20=0.1
al=0.1
eps=0.000001
Maximum f(x1,x2)= 4.2499998244024697E+000
for x1= 2.2497465037542850E+000
and x2= 1.1250767111769058E+000

5. Функция $f(x1, x2) = 2(3 - x_1)^2 + 8(x_1^2 - 2x_2)^2$ (рис. 37)

$$\frac{\partial f}{\partial x_1} = -4(3 - x_1) + 32(x_1^2 - 2x_2)x_1;$$

$$\frac{\partial f}{\partial x_2} = -32(x_1^2 - 2x_2)$$

Minimisation? (yes - 1, no - 0):1

x10=-4.5
x20=-4.5
al=0.1
eps=0.000001
Minimum f(x1,x2)= 3.9754188344704659E-004
for x1= 2.9859454547821858E+000
and x2= 4.4576566626330028E+000

6. Функция $f(x1, x2) = 2x_1 - 3.5x_2 + \exp\left(x_1^2 + x_2^2\right)$ (рис. 62)

$$\frac{\partial f}{\partial x_1} = 2 + 2x_1 \exp\left(x_1^2 + x_2^2\right);$$

$$\frac{\partial f}{\partial x_2} = -3.5 + 2x_2 \exp\left(x_1^2 + x_2^2\right)$$

Minimisation? (yes - 1, no - 0):1
x10=-2
x20=-2
al=0.1
eps=0.000001
Minimum f(x1,x2)=-1.3801176532389374E+000
for x1=-4.4580404949910635E-001
and x2= 7.8025526450351512E-001

7. Функция $f(x1, x2) = x_1 - 1.4x_2 + \exp\left(0.01x_1^2 + 0.11x_2^2\right)$ (рис. 63)

$$\frac{\partial f}{\partial x_1} = 1 + 0.02x_1 \exp\left(0.01x_1^2 + 0.11x_2^2\right);$$

$$\frac{\partial f}{\partial x_2} = -1.4 + 0.22x_2 \exp\left(0.01x_1^2 + 0.11x_2^2\right)$$

Minimisation? (yes - 1, no - 0):1
x10=-1
x20=-1
al=0.1
eps=0.000001

Minimum f(x1,x2)=-8.8124052158670985E+000
for x1=-1.1250488572560538E+001
and x2= 1.4320128260607567E+000

v26 § 7. Метод сопряжённых градиентов

Метод сочетает в себе принципы **градиентных методов** и общих **методов сопряжённых направлений**. Алгоритм разработан Флетчером и Ривсом (Fletcher R., Reeves C.M.), вариант метода предложен Полаком и Рибьером (Polak E. Ribiere G.).

В основе лежит общая стратегия спуска:

$$\mathrm{x}^{(k+1)} = \mathrm{x}^{(k)} + t_k d^{(k)} \tag{98}$$

где x – вектор аргументов, $k = 0, 1, 2, \ldots$ – номер итерации, t_k – шаг.

Направление спуска в зависимости от номера итерации вычисляется по правилу:

$$\begin{cases} d^{(0)} = -\nabla f\left(x^{(0)}\right) \\ d^{(k)} = -\nabla f\left(x^{(k)}\right) + \beta_{k-1} d^{(k-1)} \end{cases} \tag{99}$$

при минимизации, и

$$\begin{cases} d^{(0)} = \nabla f\left(x^{(0)}\right) \\ d^{(k)} = \nabla f\left(x^{(k)}\right) + \beta_{k-1} d^{(k-1)} \end{cases} \tag{100}$$

при поиске максимума.

Шаг t_k определяется по одномерной оптимизации в поперечном к плоскости координат аргументов сечении, как и в предыдущем методе.

Сопряжённые направления строятся с использованием формулы

$$\beta_{k-1} = \frac{\left\|\nabla f\left(\mathrm{x}^{(k)}\right)\right\|^2}{\left\|\nabla f\left(\mathrm{x}^{(k-1)}\right)\right\|^2} = \frac{\sum_{i=1}^{n}\left(\frac{\partial f\left(\mathrm{x}^{(k)}\right)}{\partial x_i}\right)^2}{\sum_{i=1}^{n}\left(\frac{\partial f\left(\mathrm{x}^{(k-1)}\right)}{\partial x_i}\right)^2} \tag{101}$$

Геометрическая интерпретация метода сопряжённых градиентов представлена на рис. 69.

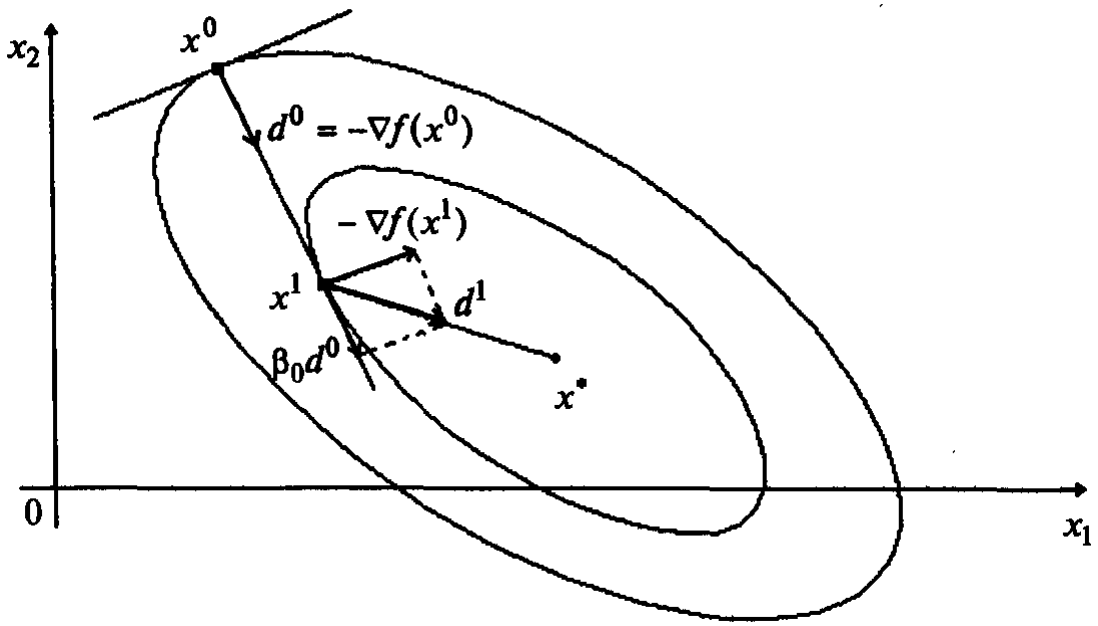


Рис. 69. Геометрическая интерпретация метода сопряжённых градиентов [2, с. 203]

Для квадратичных целевых функций с положительной матрицей Гессе метод Флетчера-Ривса является конечным и сходится за число шагов, не превышающих число аргументов целевой функции. Если же целевая функция неквадратична, метод не является конечным и большое влияние на результат оказывают погрешности вычислений. В этом случае можно использовать алгоритм Полака и Рибьера, в котором:

$$\beta_{k-1} = \begin{cases} \dfrac{\left(\nabla f\left(\mathrm{x}^{(k)}\right), \left(\nabla f\left(\mathrm{x}^{(k)}\right) - \nabla f\left(\mathrm{x}^{(k-1)}\right)\right)\right)}{\left\|\nabla f\left(\mathrm{x}^{(k-1)}\right)\right\|^2}, & k \notin J \\ 0, \quad k \in J \end{cases} \qquad (102)$$

где $J = \{0, n, 2n, \ldots\}$.

Таким образом, алгоритм Полака и Рибьера предусматривает использование итераций наискорейшего градиентного спуска через каждые n шагов [2, с. 201 – 202].

На практике, всё же во многих случаях алгоритм Флетчера-Ривса более устойчив.

Метод сопряжённых градиентов весьма полезен для оптимизации так называемых **овражных функций**, которые характеризуются

крутыми склонами «оврага» и большой вытянутостью дна «оврага». Дно, как правило, – очень пологое, и целевая функция здесь меняется медленно. Обычные градиентные методы позволяют быстро спуститься на дно «оврага», но движение по нему в точку минимума – очень медленное, зигзагообразное. Метод сопряжённых градиентов позволяет преодолеть эту трудность.

Программа, реализующая метод сопряжённых градиентов, представлена на языке Lazarus (консольное приложение) Для апробации программы использовались функции (87):

Программа метода сопряжённых градиентов (алгоритм Флетчера-Ривса)

SoprGradient_Fl_R_0 – алгоритм Флетчера-Ривса с раздельными частями определения минимума и максимума (различия в тексте выделены жирным шрифтом; в следующая программа в этом плане оптимизирована

```
program SoprGradient_Fl_R_0;
var x10a,x20a,x1a,x2a,x1b,x2b,eps,al,d1,d2,bt,fold:real;
// описание координат приближения экстремума по
// двум осям x1 и x2:
// x10a и  x20a – предыдущее приближение,
// x1a и x2a – последующее приближение;
// описание глобальных переменных x1b и x2b, используемых
// при одномерной оптимизации методом золотого сечения в
// соответствующей секущей плоскости; они представляют
// собой правые концы отрезков по отношению к
// левым концам x1a и x2a;
// описание точности eps;
// описание шага al для определения правых концов отрезков
// при одномерной оптимизации;
// описание проекций градиента или антиградиента d1 и d2
// на координатную плоскость аргументов;
// описание параметра сопряжения градиентов bt;
// описание старого (предыдущего) значения функции fold
k:integer;
```

```
// описание метки вида экстремума: k=1 – минимум,
// k=0 – максимум
//Далее описываются 7 функций, по которым проводилась
// апробация программы, 6 из которых закомментированы
// скобками { и }; f – функции, f1 и f2 – частные
//производные по соответствующим аргументам

{function f(x1,x2:real):real;                    // №1
begin f:=3*sqr(x1)+2*sqr(x2)+x1+ 2*x2;end;
function f1(x1,x2:real):real;
begin f1:=6*x1+1;end;
function f2(x1,x2:real):real;
begin f2:=4*x2+2;end;}

{function f(x1,x2:real):real;                    //№ 2
begin f:=sqr(x1-2)+sqr(x2-1)+x1+2*x2;end;
function f1(x1,x2:real):real;
begin f1:=2*(x1-2)+1;end;
function f2(x1,x2:real):real;
begin f2:=2*(x2-1)+2;end;}

{function f(x1,x2:real):real;                    // №3; max
begin f:=-3*sqr(x1-2)-4*sqr(x2-3);end;
function f1(x1,x2:real):real;
begin f1:=-6*(x1-2);end;
function f2(x1,x2:real):real;
begin f2:=-8*(x2-3);end;}

{function f(x1,x2:real):real;                    // №4; max
begin f:=-2*sqr(x1-2)-8*sqr(x2-1)+x1+2*x2;end;
function f1(x1,x2:real):real;
begin f1:=-4*(x1-2)+1;end;
function f2(x1,x2:real):real;
begin f2:=-16*(x2-1)+2;end;}

{function f(x1,x2:real):real;                    // №5
begin f:=2*sqr(3-x1)+8*sqr(sqr(x1)-2*x2);end;
```

```
function f1(x1,x2:real):real;
begin f1:=-4*(3-x1)+32*(sqr(x1)-2*x2)*x1;end;
function f2(x1,x2:real):real;
begin f2:=-32*(sqr(x1)-2*x2);end;}

{function f(x1,x2:real):real;                    // №6
begin f:=2*x1-3.5*x2+exp(sqr(x1)+sqr(x2));end;
function f1(x1,x2:real):real;
begin f1:=2+2*x1*exp(sqr(x1)+sqr(x2));end;
function f2(x1,x2:real):real;
begin f2:=-3.5+2*x2*exp(sqr(x1)+sqr(x2));end;}

function f(x1,x2:real):real;                     // №7
begin f:=x1-1.4*x2+exp(0.01*sqr(x1)+0.11*sqr(x2));end;
function f1(x1,x2:real):real;
begin f1:=1+0.02*x1*exp(0.01*sqr(x1)+0.11*sqr(x2));end;
function f2(x1,x2:real):real;
begin f2:=-1.4+0.22*x2*exp(0.01*sqr(x1)+0.11*sqr(x2));end;

// Далее описываются вспомогательные функции оптимизации
// методом золотого сечения; zs1 – левая внутренняя точка,
// zs2 – правая внутренняя точка; a –  левый конец отрезка,
// b – правый конец отрезка
function zs1(a,b:real):real;
begin zs1:=a+(1-(sqrt(5)-1)/2)*(b-a);end;
function zs2(a,b:real):real;
begin zs2:=a+(sqrt(5)-1)/2*(b-a);end;

// Далее описывается вспомогательная процедура stgr,
// которая определяет правые концы отрезков x1b и x2b
// при минимизации по методу золотого сечения;
// работа процедуры: правый конец отрезка сдвигается на
// шаг al вправо, пока функция не начнёт возрастать
procedure stgr();
begin
x1b:=x1a;
x2b:=x2a;
```

```
repeat
x1a:=x1b;
x2a:=x2b;
x1b:=x1a+al*d1;
x2b:=x2a+al*d2;
until f(x1b,x2b)>f(x1a,x2a);end;

// Далее описывается процедура минимизации по
// методу золотого сечения zs; рузультатом является
// перенос последующих координат x1a и x2a в точки
// x11 и x21, найденные с точностью eps по
// методу золотого сечения;
procedure zs();
var x11,x21,x12,x22:real;
// описание внутренних для процедуры точек x11,x21,x12 и x22
// (в основной программе они не используются)
begin
x11:=zs1(x1a,x1b);
x21:=zs1(x2a,x2b);
x12:=zs2(x1a,x1b);
x22:=zs2(x2a,x2b);
repeat
if f(x11,x21)<f(x12,x22) then
begin
x1b:=x12;
x2b:=x22;
x12:=x11;
x22:=x21;
x11:=zs1(x1a,x1b);
x21:=zs1(x2a,x2b);
end
else
begin
x1a:=x11;
x2a:=x21;
x11:=x12;
x21:=x22;
```

```
x12:=zs2(x1a,x1b);
x22:=zs2(x2a,x2b);
end;
until abs(f(x1a,x2a)-f(x1b,x2b))<eps;
x1a:=x11;
x2a:=x21;
end;

// Далее описывается вспомогательная процедура stgrMX,
// которая определяет правые концы отрезков x1b и x2b
// при поиске максимума по методу золотого сечения;
// работа процедуры: правый конец отрезка сдвигается на
// шаг al вправо, пока функция не начнёт убывать
procedure stgrMX();
begin
x1b:=x1a;
x2b:=x2a;
repeat
x1a:=x1b;
x2a:=x2b;
x1b:=x1a+al*d1;
x2b:=x2a+al*d2;
until f(x1b,x2b)<f(x1a,x2a);end;

// Далее описывается процедура поиска максимума по
// методу золотого сечения zsMX; результатом является
// перенос последующих координат x1a и x2a в точки
// x11 и x21, найденные с точностью eps по
// методу золотого сечения;
procedure zsMX();// max
var x11,x21,x12,x22:real;
// описание внутренних для процедуры точек x11,x21,x12 и x22
// (в основной программе они не используются)
begin
x11:=zs1(x1a,x1b);
x21:=zs1(x2a,x2b);
x12:=zs2(x1a,x1b);
```

```
x22:=zs2(x2a,x2b);
repeat
if f(x11,x21)>f(x12,x22) then
begin
x1b:=x12;
x2b:=x22;
x12:=x11;
x22:=x21;
x11:=zs1(x1a,x1b);
x21:=zs1(x2a,x2b);
end
else
begin
x1a:=x11;
x2a:=x21;
x11:=x12;
x21:=x22;
x12:=zs2(x1a,x1b);
x22:=zs2(x2a,x2b);
end;
until abs(f(x1a,x2a)-f(x1b,x2b))<eps;
x1a:=x11;
x2a:=x21;
end;

// Начало основной программы
begin
// определение пользователем вида экстремума
// (минимум или максимум):
write('Minimisation? (yes - 1, no - 0):');
readln(k); // ввод метки экстремума
write('x10=');readln(x1a);
write('x20=');readln(x2a);
// ввод координат экстремума начального (предыдущего)
// приближения; при этом значения записываются в ячейки
// памяти последующего приближения x1a и x2a, так как в начале
// итерационного цикла происходит переприсваивание
```

```
write('al=');readln(al);
// ввод шага для оптимизации по методу золотого сечения
write('eps=');readln(eps);
// ввод точности

// начало оптимизации; при выполнении этого условия
// имеет место минимум:
if k=1 then
begin // начало минимизации

repeat // открытие итерационного цикла
fold:=f(x1a,x2a);
// расчёт и сохранение предыдущего значения функции
// (то есть значения функции предыдущей итерации) для
//сравнения с точностью eps
x10a:=x1a;
x20a:=x2a;
// переприсваивание последующих и предыдущих координат
d1:=-f1(x10a,x20a);
d2:=-f2(x10a,x20a);
// расчёт проекций антиградиента
stgr();
// подготовка минимизации по методу золотого сечения;
// расчёт правых концов отрезков
x1a:=x10a;
x2a:=x20a;
// подготовка минимизации по методу золотого сечения;
// временное переприсваивание последующих и
// предыдущих координат
zs();
// минимизация методом золотого сечения
bt:=(sqr(f1(x1a,x2a))+sqr(f2(x1a,x2a)))/(sqr(f1(x10a,x20a))+sqr(f2(x
10a,x20a))); // расчёт параметра сопряжения градиентов
d1:=-f1(x1a,x2a)+bt*d1;
d2:=-f2(x1a,x2a)+bt*d2;
// расчёт новых значений проекций антиградиента
// (в правой части присваивания – старые значения)
```

```
x10a:=x1a;
x20a:=x2a;
// переприсваивание последующих и предыдущих координат
stgr();
// подготовка минимизации по методу золотого сечения;
// расчёт правых концов отрезков
x1a:=x10a;
x2a:=x20a;
// подготовка минимизации по методу золотого сечения;
// временное переприсваивание последующих и
// предыдущих координат
zs();
// минимизация методом золотого сечения
until abs(fold-f(x1a,x2a))<eps;
// критерий выхода из итерационного цикла
writeln('Minimum f(x1,x2)=',f(x1a,x2a),' for x1=',x1a,' and x2=',x2a);
// вывод результатов – значения функции и аргумента

end // конец минимизации
else
// при выполнении обратного условия имеет место максимум
begin// начало поиска максимума

repeat // открытие итерационного цикла
fold:=f(x1a,x2a);
// расчёт и сохранение предыдущего значения функции
// (то есть значения функции предыдущей итерации) для
//сравнения с точностью eps
x10a:=x1a;
x20a:=x2a;
// переприсваивание последующих и предыдущих координат
d1:=f1(x10a,x20a);
d2:=f2(x10a,x20a);
// расчёт проекций градиента
stgrMX();
// подготовка поиска максимума по методу золотого сечения;
// расчёт правых концов отрезков
```

```
x1a:=x10a;
x2a:=x20a;
// подготовка поиска максимума по методу золотого сечения;
// временное переприсваивание последующих и
// предыдущих координат
zsMX();
// поиск максимума методом золотого сечения
bt:=(sqr(f1(x1a,x2a))+sqr(f2(x1a,x2a)))/(sqr(f1(x10a,x20a))+sqr(f2(x
10a,x20a))); // расчёт параметра сопряжения градиентов
d1:=f1(x1a,x2a)+bt*d1;
d2:=f2(x1a,x2a)+bt*d2;
// расчёт новых значений проекций градиента
// (в правой части присваивания – старые згачения)
x10a:=x1a;
x20a:=x2a;
// переприсваивание последующих и предыдущих координат
stgrMX();
// подготовка поиска максимума по методу золотого сечения;
// расчёт правых концов отрезков
x1a:=x10a;
x2a:=x20a;
// подготовка минимизации по методу золотого сечения;
// временное переприсваивание последующих и
// предыдущих координат
zsMX();
// поиск максимума методом золотого сечения
until abs(fold-f(x1a,x2a))<eps;
// критерий выхода из итерационного цикла
writeln('Maximum f(x1,x2)=',f(x1a,x2a),' for x1=',x1a,' and x2=',x2a);
// вывод результатов – значения функции и аргумента
end; // конец поиска максимума

readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

SoprGradient_Fl_R_2.lpr – **алгоритм Флетчера-Ривса, оптимизированный** в плане универсальности по минимуму и максимуму и с рестартом:

```
program SoprGradient_Fl_R_2;
var x10a,x20a,x1a,x2a,x1b,x2b,eps,al,d1,d2,bt,fold:real;
// описание координат приближения экстремума по
// двум осям x1 и x2:
// x10a и  x20a – предыдущее приближение,
// x1a и x2a – последующее приближение;
// описание глобальных переменных x1b и x2b, используемых
// при одномерной оптимизации методом золотого сечения в
// соответствующей секущей плоскости; они представляют
// собой правые концы отрезков по отношению к
// левым концам x1a и x2a;
// описание точности eps;
// описание шага al для определения правых концов отрезков
// при одномерной оптимизации;
// описание проекций градиента или антиградиента d1 и d2
// на координатную плоскость аргументов;
// описание параметра сопряжения градиентов bt;
// описание старого (предыдущего) значения функции fold
k,n,j:integer;
// описание метки вида экстремума: k=1 – минимум,
// k=0 – максимум;
// n – количество итераций, через которое происходит обновление
// метода (рестарт), то есть расчёт и использование обычного
// градиента или антиградиента
// j – счётчик числа итераций для работы рестарта

//Далее описываются 7 функций, по которым проводилась
// апробация программы, 6 из которых закомментированы
// скобками { и }; f – функции, f1 и f2 – частные
//производные по соответствующим аргументам

{function f(x1,x2:real):real;                    // №1
```

```
begin f:=3*sqr(x1)+2*sqr(x2)+x1+ 2*x2;end;
function f1(x1,x2:real):real;
begin f1:=6*x1+1;end;
function f2(x1,x2:real):real;
begin f2:=4*x2+2;end;}

{function f(x1,x2:real):real;                    //№ 2
begin f:=sqr(x1-2)+sqr(x2-1)+x1+2*x2;end;
function f1(x1,x2:real):real;
begin f1:=2*(x1-2)+1;end;
function f2(x1,x2:real):real;
begin f2:=2*(x2-1)+2;end;}

{function f(x1,x2:real):real;                    // №3; max
begin f:=-3*sqr(x1-2)-4*sqr(x2-3);end;
function f1(x1,x2:real):real;
begin f1:=-6*(x1-2);end;
function f2(x1,x2:real):real;
begin f2:=-8*(x2-3);end;}

{function f(x1,x2:real):real;                    // №4; max
begin f:=-2*sqr(x1-2)-8*sqr(x2-1)+x1+2*x2;end;
function f1(x1,x2:real):real;
begin f1:=-4*(x1-2)+1;end;
function f2(x1,x2:real):real;
begin f2:=-16*(x2-1)+2;end;}

{function f(x1,x2:real):real;                    // №5
begin f:=2*sqr(3-x1)+8*sqr(sqr(x1)-2*x2);end;
function f1(x1,x2:real):real;
begin f1:=-4*(3-x1)+32*(sqr(x1)-2*x2)*x1;end;
function f2(x1,x2:real):real;
begin f2:=-32*(sqr(x1)-2*x2);end;}

{function f(x1,x2:real):real;                    // №6
begin f:=2*x1-3.5*x2+exp(sqr(x1)+sqr(x2));end;
```

```
function f1(x1,x2:real):real;
begin f1:=2+2*x1*exp(sqr(x1)+sqr(x2));end;
function f2(x1,x2:real):real;
begin f2:=-3.5+2*x2*exp(sqr(x1)+sqr(x2));end;}

function f(x1,x2:real):real;                                    // №7
begin f:=x1-1.4*x2+exp(0.01*sqr(x1)+0.11*sqr(x2));end;
function f1(x1,x2:real):real;
begin f1:=1+0.02*x1*exp(0.01*sqr(x1)+0.11*sqr(x2));end;
function f2(x1,x2:real):real;
begin f2:=-1.4+0.22*x2*exp(0.01*sqr(x1)+0.11*sqr(x2));end;

// Далее описываются вспомогательные функции оптимизации
// методом золотого сечения; zs1 – левая внутренняя точка,
// zs2 – правая внутренняя точка; a –  левый конец отрезка,
// b – правый конец отрезка
function zs1(a,b:real):real;
begin zs1:=a+(1-(sqrt(5)-1)/2)*(b-a);end;
function zs2(a,b:real):real;
begin zs2:=a+(sqrt(5)-1)/2*(b-a);end;

// Далее описывается вспомогательная процедура stgr,
// которая определяет правые концы отрезков x1b и x2b
// при минимизации или поиску максимума по методу
// золотого сечения;
// работа процедуры: правый конец отрезка сдвигается на
// шаг al вправо, пока функция не начнёт возрастать
// (при минимизации) или убывать (при поиске максимума)
procedure stgr();
var usl:boolean;
// описание внутренней для процедуры булевской переменной
// usl, которая формирует своё значение в соответствии
// с видом экстремума k
begin
x1b:=x1a;
x2b:=x2a;
```

```
repeat
x1a:=x1b;
x2a:=x2b;
x1b:=x1a+al*d1;
x2b:=x2a+al*d2;
if k=1 then
usl:=f(x1b,x2b)>f(x1a,x2a)
else
usl:=f(x1b,x2b)<f(x1a,x2a);
until usl;
// выход из цикла в соответствии со значением
// булевской переменной usl
end;

// Далее описывается процедура минимизации или
// поиска максимума по методу золотого сечения zs;
// результатом является перенос последующих координат
// x1a и x2a в точки x11 и x21, найденные с точностью eps по
// методу золотого сечения;
procedure zs();
var x11,x21,x12,x22:real;
// описание внутренних для процедуры точек x11,x21,x12 и x22
// (в основной программе они не используются)
usl:boolean;
// описание внутренней для процедуры булевской переменной
// usl, которая формирует своё значение в соответствии
// с видом экстремума k
begin
x11:=zs1(x1a,x1b);
x21:=zs1(x2a,x2b);
x12:=zs2(x1a,x1b);
x22:=zs2(x2a,x2b);
repeat
if k=1
then usl:=f(x11,x21)<f(x12,x22)
else usl:=f(x11,x21)>f(x12,x22);
```

```
// формирование значения булевской переменной usl
// в соответствии с видом экстремума k
if usl
then
begin
x1b:=x12;
x2b:=x22;
x12:=x11;
x22:=x21;
x11:=zs1(x1a,x1b);
x21:=zs1(x2a,x2b);
end
else
begin
x1a:=x11;
x2a:=x21;
x11:=x12;
x21:=x22;
x12:=zs2(x1a,x1b);
x22:=zs2(x2a,x2b);
end;
until abs(f(x1a,x2a)-f(x1b,x2b))<eps;
x1a:=x11;
x2a:=x21;
end;

// Начало основной программы
begin
// определение пользователем вида экстремума
// (минимум или максимум):
write('Minimisation? (yes - 1, no - 0):');
readln(k); // ввод метки экстремума
// определение пользователем количества итераций n до рестарта
write('Restart after: ');
read(n); // ввод количества итераций n до рестарта
writeln(' iterations');
```

```
write('x10=');readln(x1a);
write('x20=');readln(x2a);
// ввод координат экстремума начального (предыдущего)
// приближения; при этом значения записываются в ячейки
// памяти последующего приближения x1a и x2a, так как в начале
// итерационного цикла происходит переприсваивание
write('al=');readln(al);
// ввод шага для оптимизации по методу золотого сечения
write('eps=');readln(eps);
// ввод точности
// Начало оптимизации
j:=0;
// обнуление счётчика итераций
repeat // открытие итерационного цикла
j:=j+1; // увеличение счётчика итераций
fold:=f(x1a,x2a);
// расчёт и сохранение предыдущего значения функции
// (то есть значения функции предыдущей итерации) для
//сравнения с точностью eps
x10a:=x1a;
x20a:=x2a;
// переприсваивание последующих и предыдущих координат
if k=1 // проверка вида экстремума
then // если минимум
begin
d1:=-f1(x10a,x20a);
d2:=-f2(x10a,x20a);
// расчёт проекций антиградиента
end
else // если максимум
begin
d1:=f1(x10a,x20a);
d2:=f2(x10a,x20a);
// расчёт проекций градиента
end; //конец расчёта проекций
stgr();
```

```
// подготовка оптимизации по методу золотого сечения;
// расчёт правых концов отрезков
x1a:=x10a;
x2a:=x20a;
// подготовка оптимизации по методу золотого сечения;
// временное переприсваивание последующих и
// предыдущих координат
zs();
// оптимизация методом золотого сечения
if j=n then
begin j:=0;continue; end;
// если производится рестарт, то обнуляется счётчик итераций
// и производится заход на следующий итерационный цикл repeat,
// минуя расчёт bt и сопряжённых градиентов (continue;)
bt:=(sqr(f1(x1a,x2a))+sqr(f2(x1a,x2a)))/(sqr(f1(x10a,x20a))+sqr(f2(x
10a,x20a))); // расчёт параметра сопряжения градиентов
if k=1 // проверка вида экстремума
then // если минимум
begin
d1:=-f1(x1a,x2a)+bt*d1;
d2:=-f2(x1a,x2a)+bt*d2;
// расчёт новых значений проекций антиградиента
// (в правой части присваивания – старые значения)
end
else // если максимум
begin
d1:=f1(x1a,x2a)+bt*d1;
d2:=f2(x1a,x2a)+bt*d2;
// расчёт новых значений проекций градиента
// (в правой части присваивания – старые значения)
end;
x10a:=x1a;
x20a:=x2a;
// переприсваивание последующих и предыдущих координат
stgr();
// подготовка оптимизации по методу золотого сечения;
```

```
// расчёт правых концов отрезков
x1a:=x10a;
x2a:=x20a;
// подготовка оптимизации по методу золотого сечения;
// временное переприсваивание последующих и
// предыдущих координат
zs();
// оптимизация методом золотого сечения
until abs(fold-f(x1a,x2a))<eps;
// критерий выхода из итерационного цикла

if k=1 then
writeln('Minimum f(x1,x2)=',f(x1a,x2a),' for x1=',x1a,' and x2=',x2a)
else
writeln('Maximum f(x1,x2)=',f(x1a,x2a),' for x1=',x1a,' and x2=',x2a);
// вывод результатов – значения функции и аргумента
// в зависимости от вида экстремума
readln; // задержка закрытия программы консольного
// приложения для анализа результатов; «нажмите
// любую клавишу», лучше нажимать Enter
end.// конец программы
```

Результаты работы программы:

1. Функция $f(x1, x2) = 3x_1^2 + 2x_2^2 + x_1 + 2x_2$ (рис. 59),

$$\frac{\partial f}{\partial x_1} = 6x_1 + 1; \quad \frac{\partial f}{\partial x_2} = 4x_2 + 2;$$

```
Minimisation? (yes - 1, no - 0):1
Restart after: 4
 iterations
x10=5
x20=3
al=0.2
```

eps=0.000001
Minimum f(x1,x2)=-5.8333333266971943E-001
for x1=-1.6665194946887932E-001
and x2=-5.0000262927776185E-001

2. Функция $f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 1)^2 + x_1 + 2x_2$ (рис. 14),

$$\frac{\partial f}{\partial x_1} = 2(x_1 - 2) + 1; \quad \frac{\partial f}{\partial x_2} = 2(x_2 - 1) + 2;$$

Minimisation? (yes - 1, no - 0):1
Restart after: 4
iterations
x10=5
x20=3
al=0.2
eps=0.000001
Minimum f(x1,x2)= 2.7500000000000657E+000
for x1= 1.5000001950468551E+000
and x2= 1.6718301852488904E-007

3. Функция $f(x_1, x_2) = -3(x_1 - 2)^2 - 4(x_2 - 3)^2$ (max) (рис. 60)

$$\frac{\partial f}{\partial x_1} = -6(x_1 - 2); \quad \frac{\partial f}{\partial x_2} = -8(x_2 - 3)$$

Minimisation? (yes - 1, no - 0):0
Restart after: 4
iterations
x10=0.1
x20=0.1
al=0.2
eps=0.000001
Maximum f(x1,x2)=-6.4078969063486751E-010
for x1= 1.9999856145278987E+000
and x2= 2.9999977659304005E+000

4. Функция . $f(x_1, x_2) = -2(x_1 - 2)^2 - 8(x_2 - 1)^2 + x_1 + 2x_2$ (max)

(рис. 61)

$$\frac{\partial f}{\partial x_1} = -4(x_1 - 2) + 1; \quad \frac{\partial f}{\partial x_1} = -16(x_2 - 1) + 2$$

Minimisation? (yes - 1, no - 0):0
Restart after: 4
iterations
x10=0.1
x20=0.1
al=0.2
eps=0.000001
Maximum f(x1,x2)= 4.2499999610012020E+000
for x1= 2.2498603701182862E+000
and x2= 1.1249991492468550E+000

5. Функция $f(x1, x2) = 2(3 - x_1)^2 + 8(x_1^2 - 2x_2)^2$ (рис. 37)

$$\frac{\partial f}{\partial x_1} = -4(3 - x_1) + 32(x_1^2 - 2x_2)x_1;$$

$$\frac{\partial f}{\partial x_2} = -32(x_1^2 - 2x_2)$$

Minimisation? (yes - 1, no - 0):1
Restart after: 4
iterations
x10=-4.5
x20=-4.5
al=0.2
eps=0.000001
Minimum f(x1,x2)= 1.9229229880270118E-007
for x1= 2.9996930810209768E+000
and x2= 4.4990682592692464E+000

6. Функция $f(x1, x2) = 2x_1 - 3.5x_2 + \exp\left(x_1^2 + x_2^2\right)$ (рис. 62)

$$\frac{\partial f}{\partial x_1} = 2 + 2x_1 \exp\left(x_1^2 + x_2^2\right);$$

$$\frac{\partial f}{\partial x_2} = -3.5 + 2x_2 \exp\left(x_1^2 + x_2^2\right)$$

Minimisation? (yes - 1, no - 0):1
Restart after: 4
iterations
x10=-2
x20=-2
al=0.001
eps=0.000001
Minimum f(x1,x2)=-1.3801176488013223E+000
for x1=-4.4601885421630760E-001
and x2= 7.8023190815323895E-001
при шаге 0,01 или 0,1
программа работает некорректно

7. Функция $f(x1, x2) = x_1 - 1.4x_2 + \exp\left(0.01x_1^2 + 0.11x_2^2\right)$ (рис. 62)

$$\frac{\partial f}{\partial x_1} = 1 + 0.02x_1 \exp\left(0.01x_1^2 + 0.11x_2^2\right);$$

$$\frac{\partial f}{\partial x_2} = -1.4 + 0.22x_2 \exp\left(0.01x_1^2 + 0.11x_2^2\right)$$

Minimisation? (yes - 1, no - 0):1
Restart after: 4
iterations
x10=-1
x20=-1
al=0.2
eps=0.000001
Minimum f(x1,x2)=-8.8124053492158048E+000

for x1=-1.1251147144605216E+001
and x2= 1.4321198501570369E+000

В заключение можно перечислить некоторые методы многомерной оптимизации, не вошедшие в настоящее пособие. Методы нулевого порядка; конфигураций, Розенброка, сопряжённых направлений, случайного поиска. Методы первого порядка: Гаусса-Зейделя, Дэвидона-Флетчера-Пауэлла, кубической интерполяции. К методам первого порядка также можно отнести сведéние задачи оптимизации целевой функции к задаче решения нелинейного уравнения для производной этой функции (некоторые соответствующие методы рассмотрены в учебном пособии [12]). Методы второго порядка: Ньютона, Ньютона-Рафсона, Марквардта. Для ознакомления со всеми перечисленными методами можно рекомендовать монографию [2].

v27 Глава 4. НЕКОТОРЫЕ ВОПРОСЫ УСЛОВНОЙ ОПТИМИЗАЦИИ

v28 § 1. Введение

Разработка аналитических и численных методов оптимизации с ограничениями является более трудной задачей по сравнению с методами безусловной оптимизации. В настоящее время эффективные алгоритмы построены только для ограниченного количества классов задач, таких, как задачи линейного, квадратичного и выпуклого программирования. Среди них наиболее простой и, следовательно, более разработанной, является задача линейного программирования (ЗЛП). На практике такая задача встречается очень часто, например – в экономике, кроме того, некоторые задачи нелинейного программирования, при их некотором упрощении, могут быть сведены к ЗЛП.

В линейном программировании (ЛП) целевая функция и накладываемые на решение ограничения представляют собой линейные выражения от нескольких независимых переменных. Термин «программирование» не связан с языками программирования, появился задолго до них, и имеет в виду понятие оптимальности программы решения рассматриваемых задач.

Если ограничения, накладываемые на решение, линейны, то область допустимых решений (ОДР) всегда будет представлять собой выпуклый (см. Глава 1, §4) многогранник (для функции двух переменных – многоугольник) с известным фиксированным числом вершин. Если целевая функция также линейна, то ЗЛП можно решить даже графически (см. §2 данной Главы). В простейших случаях возможно и аналитическое решение.

Графическое решение и даже графическая интерпретация практически невозможны, если число аргументов целевой функции – три и больше. На практике во многих случаях реальное количество переменных может составлять сотни и даже тысячи. В плане этого исключительная роль в решении таких задач оптимизации принадлежит численным методам.

В случае функции двух переменных, когда ОДР представляет собой плоский многоугольник, а решением часто является одна из его

вершин, можно просто перебрать все эти вершины и найти минимальное значение целевой функции в них. В более сложных случаях (а при большом числе ограничений количество вершин может составлять десятки и сотни тысяч) простой перебор не рационален и необходимо использовать специальные численные методы.

В общем случае ЗЛП формулируется так: найти координаты $x_1, x_2, ..., x_n$ вектора x, доставляющие минимум линейной функции

$$f(\mathrm{x}) = c_1 x_1 + c_2 x_2 + ... + c_n x_n, \qquad (103)$$

и удовлетворяющие ограничениям, которые могут быть только равенствами и неравенствами вида:

$$\begin{aligned} &a_{i1} x_1 + a_{i2} x_2 + ... + a_{in} x_n \le b_i; \\ &a_{k1} x_1 + a_{k2} x_2 + ... + a_{kn} x_n = b_k; \\ &a_{l1} x_1 + a_{l2} x_2 + ... + a_{l\,n} x_n \ge b_l \end{aligned} \qquad (104)$$

Коэффициенты c_i, a_{ij}, b_i – известные величины.

Среди ограничений в подавляющем большинстве случаев встречаются требования неотрицательности всех или части аргументов:

$$x_j \ge 0, \quad j = 1, 2, ..., s \qquad (105)$$

Таким образом, (103) и (104) представляют собой **общую форму ЗЛП**. Ограничения (105) выделяют в особую группу.

Алгоритмы условной оптимизации разрабатываются для **канонической формы ЗЛП**. Здесь функция (103) остаётся той же самой, а ограничения формулируются в виде:

$$\begin{cases} a_{11} x_1 + a_{i2} x_2 + ... + a_{1n} x_n = b_1 \\ a_{21} x_1 + a_{22} x_2 + ... + a_{2n} x_n = b_2 \\ ..... \\ a_{m1} x_1 + a_{l2} x_2 + ... + a_{mn} x_n = b_m \\ x_i \ge 0, \quad i = 1, 2, ..., n \end{cases} \qquad (106)$$

Общая форма ЗЛП всегда может быть сведена к канонической. Например, условие $a_{i1} x_1 + a_{i2} x_2 + ... + a_{in} x_n \le b_i$ можно представить в виде двух условий: $a_{i1} x_1 + a_{i2} x_2 + ... + a_{in} x_n + x_{n+1} = b_i$ и $x_{n+1} \ge 0$. введя дополнительную переменную x_{n+1}. То есть неравенства в (104) можно

всегда записать в виде равенств. При этом дополнительные переменные войдут в целевую функцию с нулевыми коэффициентами

$$f(\mathrm{x}) = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n + 0 \cdot x_{n+1} + 0 \cdot x_{n+2} + \ldots + 0 \cdot x_{n+r} \qquad (107)$$

Можно сформулировать следующие правила приведения ЗЛП к каноническому виду [5, с. 29]:

1. Если в прямой задаче необходимо найти максимум целевой функции, то её надо умножить на -1, изменив тем самым знак.

2. Если среди ограничений-неравенств есть такие, в которых правая часть меньше 0, то следует также изменить их знак умножением на -1.

3. Ввести в неравенства дополнительные (фиктивные) переменные, преобразовав их в равенства.

4. Переменные, для которых не установлено, что они неотрицательные, должны быть представлены разностью двух неотрицательных дополнительных переменных.

v29 § 2. Графическая интерпретация

В качестве примера рассмотрим сравнительно простую задачу линейного программирования для целевой функции двух переменных

$$f(x_1, x_2) = 6x_1 + 10x_2 \qquad (108)$$

График этой функции показан на рис. 70 и представляет собой наклонную плоскость в трёхмерном пространстве.

В задаче требуется найти значения x_1 и x_2, удовлетворяющие условиям:

$$\begin{cases} 2x_1 + x_2 \le 180 \\ x_1 + 3x_2 \le 240 \\ 24x_1 + 20x_2 \le 2400 \end{cases} \qquad (109)$$

$$x_1 \ge 0, \quad x_2 \ge 0 \qquad (110)$$

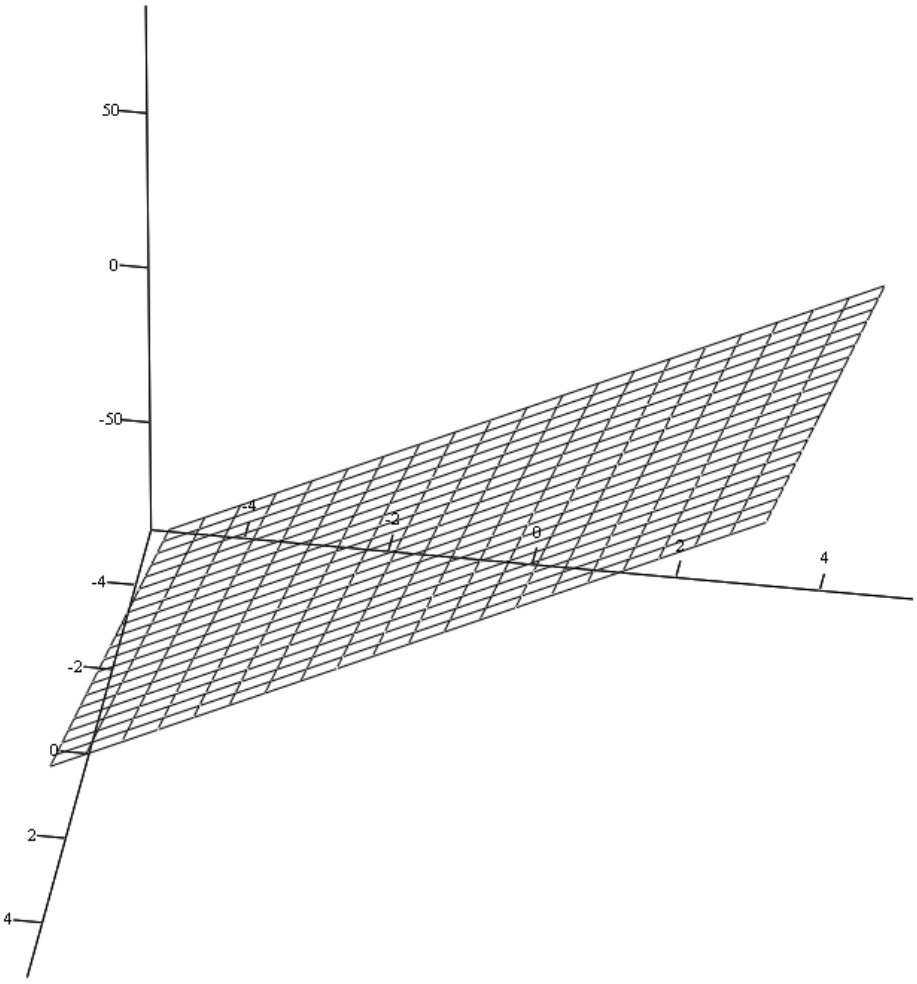
и доставляющими максимум целевой функции.

Геометрическую интерпретацию задачи можно представить на двумерной плоскости координат x_1 и x_2. Первое условие $2x_1 + x_2 \le 180$ предполагает область ниже прямой, проходящей через две точки:

при $x_1 = 0$ $x_2 = 180$ (из $2x_1 + x_2 = 180$);

при $x_2 = 0$ $x_1 = 90$ (из $2x_1 + x_2 = 180$).

Это показано на рис. 71.



fl

Рис. 70. Функция $f(x_1, x_2) = 6x_1 + 10x_2$

Аналогично, из второго условия $x_1 + 3x_2 \le 240$ имеем:

при $x_1 = 0$ $x_2 = 80$ (из $x_1 + 3x_2 = 240$);

при $x_2 = 0$ $x_1 = 240$.

(Рис. 72).

Из третьего условия $24x_1 + 20x_2 \le 2400$:

при $x_1 = 0$ $x_2 = 120$ (из $24x_1 + 20x_2 = 2400$);

при $x_2 = 0$ $x_1 = 100$.

(Рис. 73).

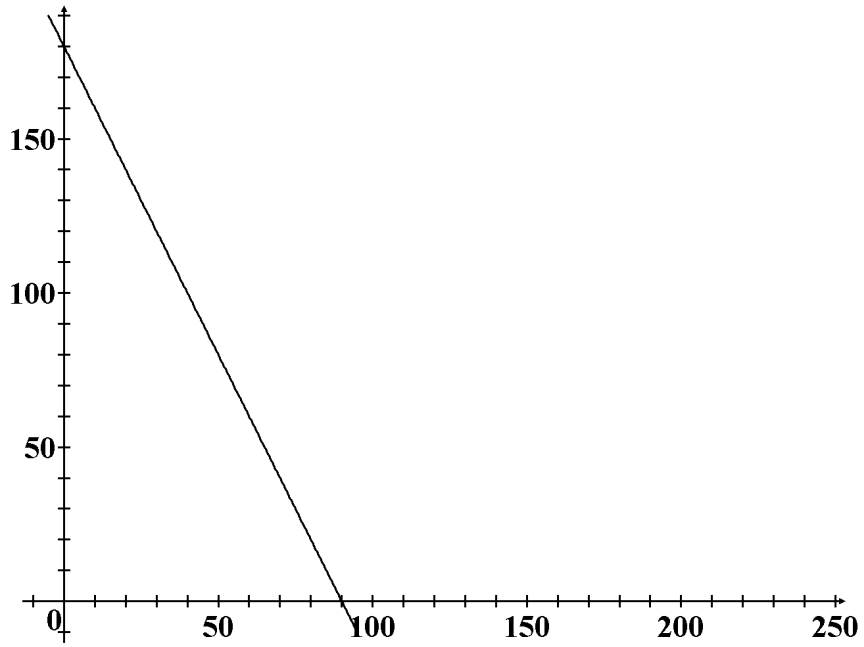


Рис. 71. Геометрическая интерпретация неравенства $2x_1 + x_2 \leq 180$

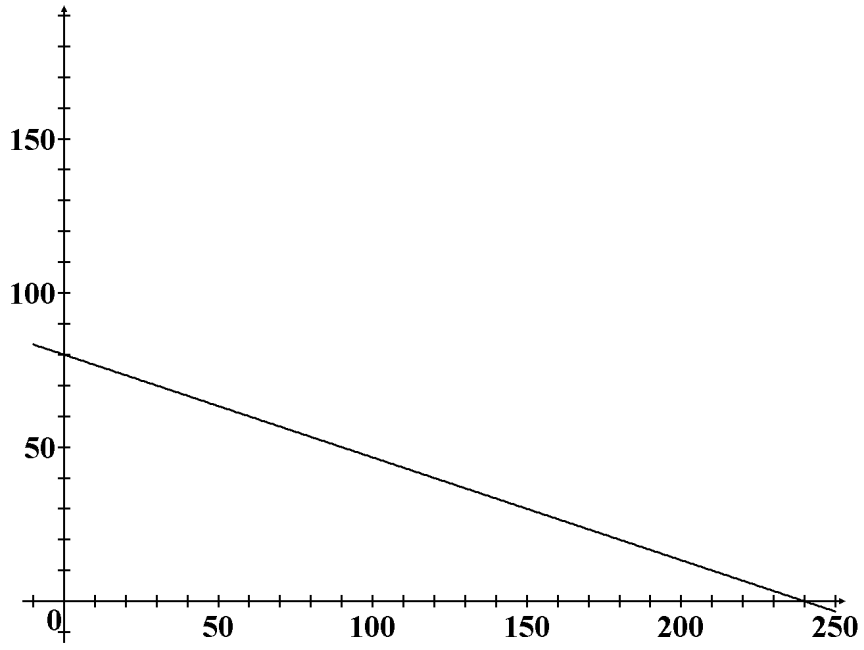


Рис. 72. Геометрическая интерпретация неравенства $x_1 + 3x_2 \leq 240$

В случае второго и третьего условий области также лежат ниже прямых. С учётом (110) можно получить замкнутую область, удовлетворяющую всем условиям (109) и (110), которая показана на

рис. (74) как заштрихованная (одноцветная). Условия (109) дают две точки пересечения 8 и 9, принадлежащие этой области. Таким образом, искомые x_1 и x_2 должны входить в полученную область.

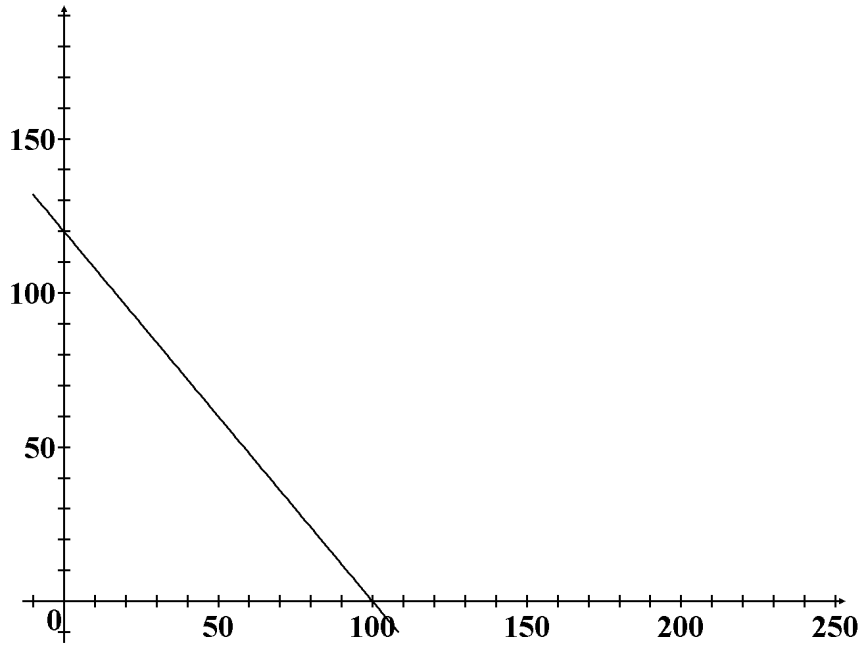


Рис. 73. Геометрическая интерпретация неравенства

$$24x_1 + 20x_2 \le 2400$$

Поскольку целевая функция – наклонная плоскость (рис. 70), то речь идёт скорее о нахождении её максимального значения на границах области, а не о локальном максимуме внутри этой области. Линии уровня (линии на графике с одинаковым значением функции) представляют собой параллельные прямые (см. рис. 70). Вычислим градиент целевой функции:

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 6; \quad \frac{\partial f(x_1, x_2)}{\partial x_2} = 10. \qquad (111)$$

Проекция градиента 6 (рис. 74) направлена из начала координат в точку – (6, 10) (на рис. 74 показана точка 7 – (60, 100), лежащая в том же направлении). Линии уровня должны быть перпендикулярны проекции градиента и параллельны между собой. В нашем случае, чем дальше лежит линия уровня от начала координат, тем бóльшее значение функции она имеет.

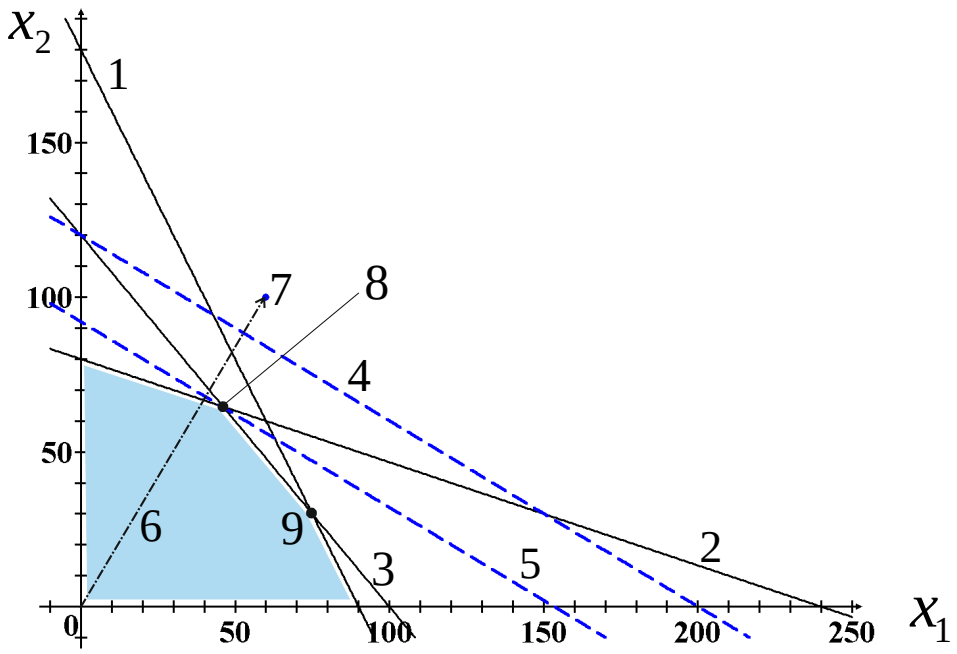


Рис. 74. Геометрическая интерпретация условий (2) и (3). Условие $2x_1 + x_2 \le 180$ – прямая 1, условие $x_1 + 3x_2 \le 240$ – прямая 2, условие $24x_1 + 20x_2 \le 2400$ – прямая 3. Заштрихована область, отвечающая всем условиям. Остальное – см. текст

Найдём линию уровня (точнее – её проекцию). проходящую, например, через точку (0, 120). При $x_1 = 0$ и $x_2 = 120$ значение функции равно $f(x_1, x_2) = 1200$ (см. (108)). Тогда другая точка будет иметь координаты (200, 0), так как при $x_2 = 0$ и $f(x_1, x_2) = 1200$ $x_1 = 200$. Проекция этой линии уровня показана на рис 74 штриховой линией 4. Эта линия лежит вне найденной области ограничений, поэтому, чтобы найти нужную линию уровня, необходимо перемещать линию 4 параллельно самой себе и перпендикулярно проекции градиента в направлении начала координат до касания с заштрихованной областью. При этом значение функции на линиях уровня будет понижаться. Таким образом, получаем линию уровня 5, которая имеет с областью ограничений общую точку 8. То есть координаты этой точки и будут решением задачи.

Чтобы найти эти координаты, необходимо решить линейную систему

$$\begin{cases} x_1 + 3x_2 = 240 \\ 24x_1 + 20x_2 = 2400 \end{cases}. \qquad (112)$$

Решением будет $x_1 = 46.2$ и $x_2 = 64.6$.

(Аналогично, можно найти координаты точки 9:

$$\begin{cases} 2x_1 + x_2 = 180 \\ 24x_1 + 20x_2 = 2400 \end{cases}, \qquad (113)$$

решение $x_1 = 75$ и $x_2 = 30$).

Таким образом, решением задачи является точка (46.2, 64.6), в которой значение целевой функции будет равно:

$$f(x_1, x_2) = 6x_1 + 10x_2 = 6 \cdot 46.2 + 10 \cdot 64.6 = 923.2 \qquad (114)$$

v30 § 3. Аналитическое решение

Для решения задачи (108), (109), (110) аналитически, приведём её к каноническому виду (103), (106). Для этого введём три (по количеству неравенств в (109)) дополнительные переменные x_3, x_4 и x_5, которые войдут в целевую функцию с нулевыми коэффициентами. При замене целевой функции её правая часть умножается на -1, так как в условиях (109) фигурирует знак «меньше». Тогда задача будет выглядеть так:

целевая функция:

$$f(x_1, x_2, x_3, x_4, x_5) = -6x_1 - 10x_2 - 0 \cdot x_3 - 0 \cdot x_4 - 0 \cdot x_5; \qquad (115)$$

ограничения:

$$\begin{cases} 2x_1 + x_2 + x_3 = 180 \\ x_1 + 3x_2 + x_4 = 240 \\ 24x_1 + 20x_2 + x_5 = 2400 \end{cases} \qquad (116)$$

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0, \quad x_5 \geq 0 \qquad (117)$$

Необходимо найти x_1, x_2, x_3, x_4 и x_5, которые удовлетворяют (116) и (117) и доставляют минимум (sic!) целевой функции (115).

Переменные x_3, x_4 и x_5, входящие в (116), – ненулевые. В качестве начального приближения положим $x_1 = 0$ и $x_2 = 0$ (эти переменные называются **свободными**) и выразим через них

переменные x_3, x_4 и x_5 (они называются **базисными**), используя (116), тогда начальное приближение будет выглядеть так:

$$x_1^{(0)} = 0,\ x_2^{(0)} = 0,\ x_3^{(0)} = 180,\ x_4^{(0)} = 240,\ x_5^{(0)} = 2400, \qquad (118)$$

Этому приближению соответствует значение функции $f^{(0)} = 0$. Далее выразим из (116) базисные переменные через свободные:

$$\begin{aligned} x_3 &= 180 - 2x_1 - x_2, \\ x_4 &= 240 - x_1 - 3x_2, \\ x_5 &= 2400 - 24x_1 - 20x_2 \end{aligned} \qquad (119)$$

Целевая функция в переменных x_1 и x_2 имеет вид: $f(x_1, x_2) = -6x_1 - 10x_2$, то есть, если увеличивать x_1 и x_2, то значение функции будет уменьшаться. Положим $x_2 = 0$. Из (117) и (119) получаем:

$$\begin{aligned} x_3 &= 180 - 2x_1 \geq 0, & x_1 &\leq 90; \\ x_4 &= 240 - x_1 \geq 0, & x_1 &\leq 240; \\ x_5 &= 2400 - 24x_1 \geq 0 & x_1 &\leq 100; \end{aligned} \qquad (120)$$

По (120) общим условием будет $x_1 = 90$, тогда с учётом (119) получаем новое опорное решение (следующей итерации):

$$x_1^{(1)} = 90,\ x_2^{(1)} = 0,\ x_3^{(1)} = 0,\ x_4^{(1)} = 150,\ x_5^{(1)} = 240, \qquad (121)$$

Целевая функция равна $f^{(1)} = -540$, что меньше предыдущего приближения, то есть новое опорное решение лучше.

Далее примем нулевые переменные x_2 и x_3 за свободные, а x_1, x_4 и x_5 – за базисные. Из (116) получаем:

$$\begin{aligned} x_1 &= 90 - x_2 / 2 - x_3 / 2, \\ x_4 &= 240 - x_1 - 3x_2, \qquad ; \\ x_5 &= 2400 - 24x_1 - 20x_2 \end{aligned}$$

в полученных формулах выражаем базисные переменные только через свободные, для чего x_1 из первой формулы подставляем в остальные:

$$\begin{aligned} x_1 &= 90 - x_2 / 2 - x_3 / 2, \\ x_4 &= 150 - 2.5x_2 + x_3 / 2, \\ x_5 &= 240 - 8x_2 + 12x_3 \end{aligned} \qquad (122)$$

Из (115) и (122) выражаем целевую функцию в новых свободных переменных:

$$f(x_2, x_3) = -6 \cdot (90 - x_2/2 - x_3/2) - 10x_2 = \\ = -540 + 3x_2 + 3x_3 - 10x_2 = -540 - 7x_2 + 3x_3 \quad (123)$$

Из (123) видно, что если увеличивать x_2 при $x_3 = 0$, то значение функции будет уменьшаться. Положим $x_3 = 0$. Из (117) и (122) получаем:

$$x_1 = 90 - x_2/2 - x_3/2 \geq 0, \quad x_1 = 90 - x_2/2 \geq 0, \quad x_2 \leq 45$$
$$x_4 = 150 - 2.5x_2 + x_3/2 \geq 0, \quad x_4 = 150 - 2.5x_2 \geq 0, \quad x_2 \leq 60$$
$$x_5 = 240 - 8x_2 + 12x_3 \geq 0 \quad x_5 = 240 - 8x_2 \geq 0 \quad x_2 \leq 30 \quad (124)$$

По (124) общим условием будет $x_2 = 30$, тогда с учётом (122) получаем новое опорное решение (второй итерации):

$$x_1^{(2)} = 75,\ x_2^{(2)} = 30,\ x_3^{(2)} = 0,\ x_4^{(2)} = 75,\ x_5^{(2)} = 0, \quad (125)$$

Целевая функция равна $f^{(2)} = -750$, что меньше предыдущего приближения, то есть новое опорное решение ещё лучше.

Итерации можно продолжать подобным образом; для свободных переменных x_3 и x_5 из (116) можно получить:

$$x_1 = 75 - 5x_3/4 - x_5/16,$$
$$x_2 = 30 - 3x_3/2 - x_5/8, \quad ,$$
$$x_4 = 75 - 13x_3/4 + 5x_5/16$$
$$f(x_3, x_5) = -750 - 15x_3/2 + 7x_5/8.$$

Если увеличивать x_3 при $x_5 = 0$, то функция будет уменьшаться.

$$x_1 = 75 - 5x_3/4 + x_5/16 \geq 0, \quad x_2 \leq 60$$
$$x_2 = 30 + 3x_3/2 - x_5/8 \geq 0, \quad x_2 \leq -20 \quad x_2 = 23.077$$
$$x_4 = 75 - 13x_4/4 + 5x_5/16 \geq 0 \quad x_2 \leq 23.077$$
$$x_1^{(3)} = 46.154,\ x_2^{(3)} = 64.616,\ x_3^{(3)} = 23.077,\ x_4^{(3)} = 0,\ x_5^{(3)} = 0,$$

Целевая функция: $f^{(3)} = -923.084 < f^{(2)}$. Далее можно показать, что для свободных переменных x_4 и x_5 целевая функция будет равна:

$$f(x_4, x_5) = -923.08 + 30x_4/13 + 8x_5/15,$$

и мы не можем их менять, уменьшая функцию. Тогда решением будет (в переводе на исходную задачу поиска максимума):

$$x_1 = 46.154, \quad x_2 = 64.616, \quad f(x_1, x_2) = 923.08,$$

что совпадает с графическим решением (114).

v31 § 4. Понятие о симплекс-методе

Самым распространённым численным методом решения ЗЛП является так называемый симплекс-метод, который в настоящее время хорошо проработан и широко используется на практике. Во многих математических пакетах программ имеются соответствующие стандартные алгоритмы (например, – MathCAD, MATLAB, Excel). Метод не только является универсальным для ЗЛП, но и применяется для решения смежных задач. Его достоинством является возможность использования матричного представления задачи,

ЗЛП должна быть сформулирована в канонической форме. В этом случае ОДР представляет собой выпуклый многогранник (латинское simplex означает «», то есть имеется в виду «простой выпуклый многогранник»). Число граней этого многогранника равно числу измерений: нульмерное пространство – точка, одномерное – отрезок прямой, двумерное – многоугольник, трёхмерное – трёхмерный многогранник и так далее. Решение ищется в вершинах многогранника.

Как уже отмечалось, метод полного перебора опорных точек (для двумерного случая – вершин многогранника) позволяет решить ЗЛП за конечное число шагов. На практике это число шагов чаще всего представляет собой очень большое число. вследствие чего метод практической ценности не имеет. В симплекс-методе полный перебор заменяется перебором оптимизированным, при котором неоптимальные опорные точки не исследуются и алгоритм существенно сокращается. Причём, начальная точка выбирается случайным образом. В методе на каждой итерации производится анализ параметров и делается вывод либо об уже достигнутом решении задачи; либо обосновывается. что задача решения не имеет; либо определяется «лучшая» опорная точка, на основе которой проводится следующая итерация. То есть последующие точки выбираются так, чтобы решение каждый раз было лучше предыдущего; это контролируется по значению целевой функции. Теоретически конечно возможно, что в методе произойдёт полный перебор всех точек, но практическое использование метода показало, что в подавляющем большинстве случаев количество итераций достаточно мало́.

Итак, симплекс-метод состоит в последовательном построении опорных решений, каждое следующее из которых уменьшает целевую

функцию. В некоторых случаях решение ЗЛП симплекс-методом даёт некорректные результаты, что выясняется в процессе самогó решения. Это может происходить, когда [5, с. 48]:

1) решение вырождено;

2) имеются альтернативные оптимальные решения;

3) имеют место неограниченные решения;

4) отсутствуют допустимые решения.

Симплекс-метод в некоторых случаях можно реализовать и с помощью ручного счёта; при этом составляются так называемые **симплекс-таблицы**. Программы численной реализации весьма сложны, так как включают в себя автоматический анализ поставленной задачи.

v32 ЗАКЛЮЧЕНИЕ

Представленный в учебном пособии материал является третьей частью курса «Математические методы в химии и химической технологии» и посвящён такой важнейшей для химии области, как оптимизация химических и технологических процессов. Оптимизация подразумевает поиск оптимальных параметров и условий, наиболее подходящих для решения конкретных научных и технологических задач. Это легко подвергается математической формализации и сводится к решению чисто математических задач. Особая роль в их решении принадлежит численным методам, которые позволяют составить соответствующие компьютерные программы.

Первая часть курса «Математические методы в химии и химической технологии» – «Основы программирования вычислительных задач» – была издана в 2018 году. В ней были рассмотрены методы программирования на языке Pascal, который является базовым языком таких современных средств программирования, как пакеты Lazarus и DELPHI. Это служит основой для представленных в настоящем пособии конкретных программ по оптимизации на языке Lazarus.

Вторая часть курса – «Численные методы решения алгебраических задач и обработки функций» (издана в 2019 году) посвящена численным методам самого общего для химии характера. Многие понятия этой области используются и при решении задач оптимизации, в частности в некоторых случаях последние можно свести к решению нелинейных уравнений, которые рассмотрены во второй части курса.

В настоящем пособии представлены теоретические основы решения задач оптимизации и классический метод решения этих задач, рассмотрены численные методы решения задач одномерной и многомерной оптимизации с их реализацией в виде компьютерных программ на языке Lazarus, дано понятие об условной оптимизации.

В будущем планируется издание четвёртой части курса «Математические методы в химии и химической технологии», в которой будет рассмотрено решение дифференциальных уравнений, обыкновенных и частных производных, которые описывают многие химические и химико-технологические процессы и состояния систем.

Учебное пособие закладывает у будущих специалистов базовые основы для успешного решения прикладных задач в области химии и химической технологии. Цель пособия – оказать помощь обучающимся в более продуктивном овладении знаниями и навыками для практической работы на производстве и в научных учреждениях.

v33 ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. В каких областях химии и химического производства могут встречаться задачи оптимизации? Приведите примеры.
2. В чём отличие безусловной и условной оптимизации?
3. Является ли окружность выпуклым множеством?
4. Что такое унимодальная функция?
5. Где в задачах оптимизации используется матрица Гессе?
6. Можно ли градиент как таковой использовать при решении задачи минимизации?
7. Перечислите необходимые и достаточные условия безусловной оптимизации.
8. Почему при решении задач оптимизации нельзя ограничиваться классическим методом?
9. Преимущества и недостатки метода сканирования в одномерной оптимизации.
10. Главный недостаток метода дихотомии в одномерной оптимизации.
11. Что такое золотое сечение? Основное свойство золотого сечения.
12. Почему среди методов одномерной оптимизации именно метод золотого сечения чаще всего используется как вспомогательный в оптимизации многомерной?
13. Для каких целевых функций лучше использовать метод квадратичной интерполяции?
14. Что такое целевая функция?
15. В чём преимущество методов покоординатного спуска и Нелдера-Мида перед градиентными методами?
16. Что такое принцип монотонности?
17. Можно ли назвать метод Нелдера-Мида «умным» методом покоординатного спуска?
18. В чём преимущество метода наискорейшего спуска перед методом градиентного спуска?
19. Преимущества и недостатки метода сопряжённых градиентов.

20. Какой метод наиболее оптимален для обработки «овражных» функций?
21. В чём состоит принципиальная сложность оптимизации с ограничениями?
22. В чём состоит отличие симплекс-метода от метода простого перебора?
23. Что такое локальный и глобальный экстремум?
24. Что такое поверхность уровня функции?
25. Что такое методы нулевого, первого и второго порядков?
26. По какому принципу строится ряд чисел Фибоначчи?
27. В чём отличие канонической формы задачи линейного программирования от общей формы?

v34 БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Использованная литература

1. Васильев Ф.П. Численные методы решения экстремальных задач. Изд. 2-е, перераб. и дополн. М.: Наука. 1988. 550 с.

2. Пантелеев А.В., Летова Т.А. Методы оптимизации в примерах и задачах. Изд. 2-е, исправл. М.: Высшая школа. 2005. 544 с.

3. Сухарев А.Г., Тимохов А.В., Фёдоров В.В. Курс методов оптимизации. Изд. 2-е. М.: Физматлит. 2005. 368 с.

4. Панов В.А. Математические основы теории систем. Методы оптимизации. Изд. 2-е, перераб. и дополн. Пермь: Изд-во Пермского национального исследовательского политехнического университета. 2011. 148 с.

5. Певнева А.Г., Калинкина М.Е. Методы оптимизации. Санкт-Петербург. Университет ИТМО. 2020. 65 с.

6. Гребенникова И.В. Методы оптимизации. Учебное пособие. Екатеринбург. Изд. Уральского университета. 2017. 150 с.

7. Бронов С.А. Методы оптимизации в САПР. Красноярск. 2011. 122 с.

8. Прокопенко Н.Ю. Методы оптимизации. Учебное пособие. Нижний Новгород. ННГАСУ. 2018. 120 с.

9. Плис А.И., Сливина Н.А. Лабораторный практикум по высшей математике. Изд. 2-е, перераб. и дополн. М.: Высшая школа. 1994. 416 с.

10. Банди Б. Методы оптимизации. Вводный курс. М.: Радио и связь. 1988. 128 с.

11. Лобко В.Н. Математические методы в химии и химической технологии. Основы программирования вычислительных задач. Учебное пособие / Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых; Владимир: Изд-во ВлГУ, 2018. 108 с.

12. Лобко В.Н. Математические методы в химии и химической технологии. Численные методы решения алгебраических задач и обработки функций. Учебное пособие / Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых; Владимир: Изд-во ВлГУ, 2019. 144 с.

13. Галеев Э.М. Оптимизация. Теория. Примеры. Задачи. М.: Едиториал УРСС. 2002. 302 с.

14. Смирнов И.А. Методы оптимизации. Базовый курс. Санкт-Петербург. СПбГТИ(ТУ). 2010. 102 с.

15. Бочкарёв В.В. Оптимизация химико-технологических процессов. Томск. Изд. Томского политехнического университета. 2014. 264 с.

16. Гилл Ф., Мюррей У. Райт М. Практическая оптимизация. М.: Мир. 1985. 510 с.

17. Поляк Б.Т. Введение в оптимизацию. М.: Наука. Главная редакция физико-математической литературы. 1983. 384 с.

18. Аоки М. Введение в методы оптимизации. М.: Наука. Главная редакция физико-математической литературы. 1977. 344 с.

19. Ларичев О.И. Горвиц Г.Г. Методы поиска локального экстремума овражных функций. М.: Наука 1990. 95 с.

Дополнительная литература

1. Есипов Б.А. Методы оптимизации и исследование операций. Конспект лекций. Самара. Изд. Самарского государственного аэрокосмического университета. 2007. 90 с.

2. Мицель А.А. Романенко В.В., Грибанова Е.Б. Методы оптимизации. Учебно-методическое пособие по выполнению контрольных и лабораторных работ. Томск. ФДО. ТУСУР. 2018. 451 с.

3. Аттетков А.В. Канатников А.Н. Тверская Е.С. Численные методы решения задач многомерной безусловной оптимизации. Часть 1. Методы первого и второго порядков. М.: Изд. МГТУ им. Баумана. 2009. 48 с.

4. Калиткин Н.Н. Численные методы. М.: Наука. Главная редакция физико-математической литературы. 1978. 512 с.

5. Пирумов У.Г. Численные методы. М.: Дрофа. 2003. 224 с.

6. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. М.-СПб. Физматлит. 2001. 632 с.

7. Каханер Д. Моулер К. Нэш С. Численные методы и программное обеспечение. М.: Мир. 1998. 576 с.

Учебное электронное издание

ЛОБКО Владимир Николаевич

МАТЕМАТИЧЕСКИЕ МЕТОДЫ
В ХИМИИ И ХИМИЧЕСКОЙ ТЕХНОЛОГИИ

ЗАДАЧИ ОПТИМИЗАЦИИ

Учебное пособие

Издается в авторской редакции

Системные требования: Intel от 1,3 ГГц; Windows XP/7/8/10; Adobe Reader; дисковод CD-ROM.

Тираж 9 экз.

Издательство Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.