МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

А.И. Данилин

Методы оптимизации

Электронное учебное пособие

САМАРА
2011

УДК 51.380.115
ББК 22.18
Д 182

Данилин А. И., Методы оптимизации [Электронный ресурс] : электрон. учеб. пособие / А. И. Данилин; Минобрнауки России, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т). - Электрон. текстовые и граф. дан. (1,4 МБайт). - Самара, 2011. - 1 эл. опт. диск (CD-ROM).

Рассмотрены методы поиска экстремума в самой неблагоприятной овражной ситуации. Основное внимание уделено практической реализации методов оптимизации, поэтому для каждого метода последовательно излагается теория, блок-схема и алгоритм поиска, проводится подробный разбор контрольного примера и прилагается текст программы на языке С++.

Учебное пособие предназначено для студентов факультета инженеров воздушного транспорта, обучающихся по направлению подготовки магистров специальностей 162300.68 "Техническая эксплуатация летательных аппаратов и двигателей" и 162500.68 «Техническая эксплуатация авиационных электросистем и пилотажно-навигационных комплексов», дисциплина «Методы оптимизации», семестр 9.

Разработано на кафедре эксплуатации авиационной техники.

1. Введение.

В пособии рассмотрены методы поиска оптимальных значений максимума или минимума функции одной и n действительных переменных $f(x_1, x_2, \ldots x_n)$. Если функция выражает производительность, получаемую при эксплуатации самолётов типа x_i в количестве P_i, то мы будем стремиться максимизировать функцию. С другой стороны, если она выражает себестоимость эксплуатации самолёта, то мы будем стремиться минимизировать функцию. С математической точки зрения не играет существенной роли, рассматривать максимизацию или минимизацию, поскольку максимизация функции f эквивалентна минимизации $(-f)$. Мы ограничимся рассмотрением минимизации.

Значения переменных могут подчиняться ограничениям или изменяться без ограничений. Если, например, они действительно выражают количество определенных эксплуатируемых самолётов, то при этом будет существовать ограничение на их производительность и на их количество, которое может позволить себе авиакомпания. Таким образом, любое решение оптимизационной задачи должно учитывать эти ограничения. В пособии рассмотрены задачи, в которых на переменные не наложены ограничения, однако в [1] показаны способы сведения задач с ограничениями к эквивалентным задачам без ограничений.

В любой практической оптимизационной задаче существует много совпадающих этапов. Наиболее важным этапом является моделирование рассматриваемой физической ситуации с целью получения математической функции, которую необходимо минимизировать, а также определения ограничений, если таковые существуют. Затем следует выбрать подходящую процедуру для осуществления минимизации. Эта процедура должна быть реализована на практике, что во многих реальных случаях вынуждает использовать компьютер для выполнения большого объема вычислений. И наконец, математический результат должен быть интерпретирован опять же в терминах физического содержания задачи. Некоторые методические рекомендации по выполнению этих этапов также рассмотрены в пособии [1].

Хотя ни одним из этих этапов нельзя пренебречь, основной упор в настоящем пособии сделан на изучение процедур, предназначенных для осуществления минимизации, и возможностей их преобразования в такие вычислительные процедуры, которые можно выполнить на компьютере.

Не случайно, что многие важные методы оптимизации были разработаны в течение последних пяти десятилетий, в период появления компьютеров, и эти методы являются машинными. Разрабатываемые алгоритмы оптимизации трудно считать сколько-нибудь практически значимыми без компьютеров, находящихся в нашем распоряжении. В настоящее время в коммерческом распространении имеются пакеты программ оптимизации, реализующие эти методы. Они могут оказаться весьма эффективными и позволят решить широкий круг задач. При этом они являются достаточно самостоятельными и могут использоваться без оценки физической сути решаемой задачи. Для большинства методов, рассматриваемых в этом пособии (причем не исчерпывающих список существующих методов), приведены программы на языке C++. Дано подробное объяснение процесса создания этих программ, что позволяет по-новому взглянуть на применимость методов оптимизации.

2. Классические методы

2.1. Функции одной переменной

Функция $f(x)$ имеет локальный минимум в точке x_0, если существует некоторая положительная величина δ, такая, что если $| x - x_0 | < \delta$, то $f(x) \geq f(x_0)$, то есть, если существует окрестность точки x_0, такая, что для всех значений x в этой окрестности $f(x)$ больше $f(x_0)$. Функция $f(x)$ имеет глобальный минимум в точке x^*, если для всех x справедливо неравенство $f(x) \geq f(x^*)$.

На рис. 1 дано графическое представление функции $f(x)$, которая имеет локальный минимум в точке x_0 и глобальный минимум в точке x^*.

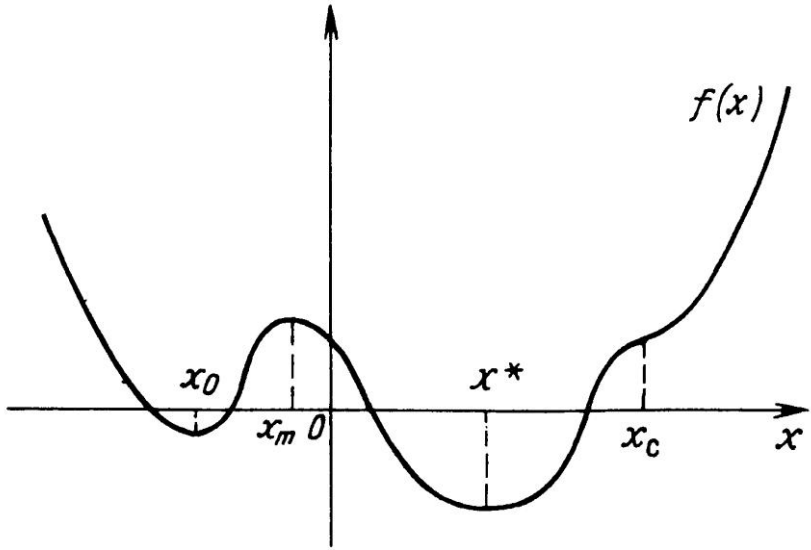

Рис. 1. Локальный и глобальный минимум

Классический подход к задаче нахождения значений x_0 и x^* состоит в поиске уравнений, которым они должны удовлетворять. Представленная на рис. 1 функция и ее производные непрерывны, и видно, что в точках x_0 и x^* производная $f'(x) = df(x) / dx$ (градиент функции) равна нулю. Следовательно, x_0 и x^* будут решениями уравнения

$$f'(x) = 0. \qquad (1)$$

Точка x_m, в которой достигается локальный максимум, и точка x_c, в которой имеется точка горизонтального перегиба функции, также удовлетворяют этому уравнению. Следовательно, уравнение (1) является только *необходимым* условием минимума, но не является *достаточным* условием минимума.

Заметим, однако, что в точках x_0 *и* x^* производная $f'(x)$ меняет знак с отрицательного на положительный. В точке x_m знак меняется с положительного на отрицательный, в то время как в точке x_c он не меняется. Следовательно, производная в точке минимума является возрастающей функцией, а поскольку степень

возрастания $f'(x)$ измеряется второй производной, можно ожидать, что $f''(x_0) > 0$, $f''(x^*) > 0$, тогда как $f''(x_m) < 0$.

Если, однако, вторая производная равна нулю, ситуация остается неопределённой.

Приведённые рассуждения могут найти надёжное обоснование, если рассмотреть разложение функции $f(x)$ в ряд Тейлора в окрестности точки x_0 (или x^*, или x_m), что, конечно, требует непрерывности функции $f(x)$ и ее производных:

$$f(x_0 + h) - f(x_0) = h\, f'(x_0) + \frac{h^2}{2!} f''(x_0) + \ldots \qquad (2)$$

Если в точке x_0 достигается минимум, то левая часть (2) будет неотрицательной для любого достаточно малого h ($| h | < \delta$). Следовательно, первая производная $f'(x)$ должна быть равна нулю, и это является достаточным условием, см. уравнение (1). Если бы она была положительной, то достаточно малое отрицательное значение h делало бы правую часть (2) отрицательной, а если бы она была отрицательной, то достаточно малое положительное значение h делало бы правую часть отрицательной.

Так как в следующем члене (2) всегда $h^2 > 0$, тогда, если

$$f''(x_0) > 0, \qquad (3)$$

то в точке x_0 достигается минимум. Если $f'(x) = 0$ и $f''(x_m) < 0$, то из аналогичных соображений в точке x_m достигается максимум. Для определения различия между локальным и глобальным минимумами необходимо сравнить значения функций $f(x_0)$ и $f(x^*)$.

Неоднозначность, возникающую при равенстве нулю второй производной: $f''(x_0) = 0$, можно разрешить, увеличив количество членов в формуле разложения в ряд Тейлора:

$$f(x_0 + h) - f(x_0) = h\, f'(x_0) + \frac{h^2}{2!} f''(x_0) + \frac{h^3}{3!} f'''(x_0) + \frac{h^4}{4!} f^{(4)}(x_0) \ldots$$

При этом можно сформулировать следующее правило: если функция $f(x)$ и ее производные непрерывны, то точка x_0 является

точкой экстремума (максимума или минимума) тогда, и только тогда, когда n - четное, где n - порядок *первой* необращающейся в нуль производной в точке x_0. Если $f^{(n)}(x_0) < 0$, то в точке x_0 достигается максимум, если $f^{(n)}(x_0) > 0$, то в точке x_0 достигается минимум.

Пример 1. Найти точку экстремума функции $f(x) = (x - 1)^6$.

$f'(x) = 6(x - 1)^5 = 0$ при $x = 1$.

Первой необращающейся в нуль производной в точке $x = 1$ будет $f^{(6)}(1) = (6!) > 0$. Следовательно, функция $f(x)$ имеет минимум в точке $x = 1$.

2.2. Функции n переменных.

Рассмотрим функцию n действительных переменных

$$f(X) = f(x_1, x_2, x_3 \ldots x_n).$$

Точка в n-мерном евклидовом пространстве с координатами x_1, x_2, x_3, ... x_n обозначается вектором-столбцом $\boldsymbol{X}$. Градиент функции, то есть вектор с компонентами $\partial f/\partial x_1$, $\partial f/dx_2$, … $\partial f/\partial x_n$, обозначается $\nabla f(X)$ или, иногда, $g(X)$. Матрица Гессе (гессиан) функции $f(X)$ обозначается как $G(X)$ и является симметричной матрицей (n x n) элементов вида

$$G_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

Функция $f(X)$ имеет локальный минимум в точке X_0, если существует окрестность точки X_0, такая, что $f(X)$ больше $f(X_0)$ во всех точках этой окрестности, то есть существует положительная величина δ, такая, что для $| X — X_0 | < \delta$ справедливо неравенство $f(X) > f(X_0)$.

В случае глобального минимума в точке X^* для всех X справедливо неравенство $f(X) > f(X^*)$.

При таких определениях и очевидных предположениях относительно дифференцируемости можно обобщить уравнение (2) и получить:

$$f(X+H)-f(X_0)=\sum_{i=1}^{n}h_i\frac{\partial f(x_1...x_n)}{\partial x_i}+\frac{1}{2!}\sum_{i=1}^{n}\sum_{j=1}^{n}h_ih_j\frac{\partial^2 f(x_1...x_n)}{\partial x_i\partial x_j}+...= \\ =H^T\nabla f(X_0)+\frac{1}{2}H^TG(X_0)H+... \qquad (4)$$

Тогда, если X_0 является точкой минимума функции f(X), то каждая первая частная производная $\partial f/\partial x_i$ (i = 1,...n) должна обращаться в нуль в точке X_0. Если это не так, то соответствующим выбором h_i можно добиться того, что разность $f(X_0+H) - f(X_0)$ будет отрицательна.

Следовательно, необходимым условием минимума в точке X_0 является уравнение

$$\nabla f(X_0)=0, \qquad (5)$$

то есть

$$\frac{\partial f(X_0)}{\partial x_i}=0, \quad i=1,2...n. \qquad (6)$$

Тогда знак разности $f(X_0 + H) - f(X_0)$ определяется членом $\frac{1}{2}H^TG(X_0)H$ из уравнения (4). Если матрица $G(X_0)$ положительно определена, то этот член положителен для всех H. Следовательно, необходимыми и достаточными условиями минимума являются:

$$\nabla f(X_0)=0, \ \ G(X_0) \text{ положительно определена.} \qquad (7)$$

Необходимыми и достаточными условиями максимума являются

$$\nabla f(X_0)=0, \ \ G(X_0) \text{ отрицательно определена} \qquad (8)$$

Пример 2. Исследовать экстремальную точку функции $f(X) = x_1^2 + x_2^2 + x_3^2 - 4x_1 - 8x_2 - 12x_3 + 100$.

$$\nabla f(X) = \begin{Bmatrix} 2x_1 - 4 \\ 2x_2 - 8 \\ 2x_3 - 12 \end{Bmatrix} = 0 \text{ при } x_1 = 2;\ x_2 = 4;\ x_3 = 6.$$

$$G(X) = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

в точке $x_1 = 2$; $x_2 = 4$; $x_3 = 6$ и положительно определена, поскольку определитель $G(X)$ больше нуля (все собственные значения положительны и равны 2).

Следовательно, в точке $X(2; 4; 6)$ функция $f(X)$ достигает минимума.

2.3. Метод Ньютона

Для функций одной переменной классический подход при поиске значений x в точках перегиба функции $f(x)$ состоит в решении уравнения

$$f'(x) = 0.$$

Решить такое уравнение не всегда просто. Поэтому кратко рассмотрим численный метод его решения. Приблизительный эскиз кривой $y = f'(x)$ позволит получить приближенное решение. Если можно найти два значения a и b таких, что $f'(a)$ и $f'(b)$ имеют противоположные знаки, то тогда, в силу очевидных предположений о непрерывности, будет существовать корень η настоящего уравнения, причем $a < \eta < b$, см. рис. 2.

Метод Ньютона позволяет улучшить относительно грубую аппроксимацию, чтобы получить корень уравнения $\varphi(x) = 0$. Здесь $\varphi(x) = f'(x)$. На рис. 3 точка x_0, являющаяся координатой x точки P, представляет собой аппроксимацию корня уравнения $\varphi(x) = 0$. Пусть PT — касательная к кривой $y = \varphi(x)$ в точке P, а T — точка, в которой касательная пересекает ось x.

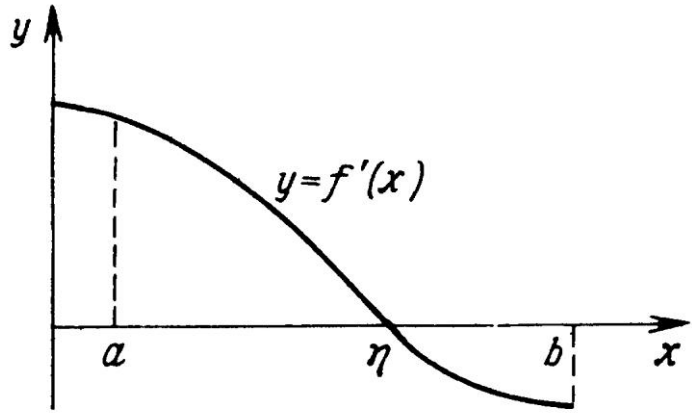

Рис. 2. Корень уравнения $y = f'(x)$.

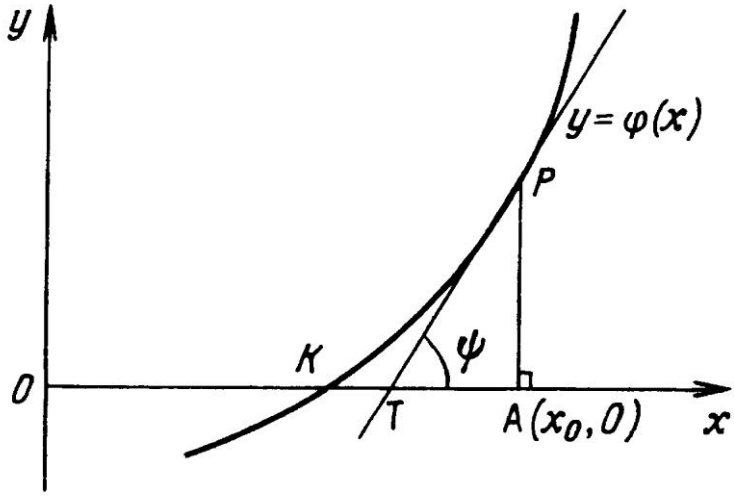

Рис. 3. Аппроксимация решения уравнения $y = f'(x)$.

Тогда в общем случае значение x в точке T является лучшей аппроксимацией корня, лежащего на самом деле в точке K.

Теперь $OT = OA - TA = x_0 - TA$. Кроме того, $\frac{PA}{TA} = tg\,\Psi = \varphi'(x_0)$, следовательно, $TA = \frac{PA}{\varphi'(x_0)} = \frac{\varphi(x_0)}{\varphi'(x_0)}$ и соответственно

$$x_1 = x_0 - \frac{\varphi(x_0)}{\varphi'(x_0)}. \qquad (9)$$

Аналогично можно получить улучшенное значение для $x_2 = x_1 - \frac{\varphi(x_1)}{\varphi'(x_1)}$ и в общем случае

$$x_{r+1} = x_r - \frac{\varphi(x_r)}{\varphi'(x_r)} \qquad (10)$$

Итерации могут быть продолжены до тех пор, пока для двух последующих аппроксимаций не будет достигнута требуемая точность. Приведенная далее подпрограмма реализует этот алгоритм.

```
#include <math.h>
double Newton(x0)
{
// Программа поиска корня функции φ(x) вблизи точки x0
// Значение функции φ(x) вычисляется в подпрограмме Fi(x)
// Значение производной – в подпрограмме Fi_der(x,dx)
extern double Fi(double);
extern double Fi_der(double, double);

double  fi, x_current, x, dx=0.001, derivative,
        tolerance=0.0001;

x_current = x0;
do {
      x = x_current;
      fi = Fi(x_current);
      derivative = Fi_der(x,dx);
      x_current = x - fi/derivative;
}
while (fabs(x_current - x) > tolerance);
return (x);
}
```

Универсальность программы достигается за счет того, что функция *fi = φ(x)* вычисляется в подпрограмме *Fi(x)*, а производная функции *derivative = φ'(x)* вычисляется в подпрограмме *Fi_der(x,dx)* для произвольного значения *x*. Точность решения задаётся переменной *tolerance*.

Если в подпрограммах *Fi(x)* и *Fi_der(x,dx)* вычислять соответственно первую и вторую производную функции *f(x)*, то мы найдем точку её экстремума. Если функции *f(x)* и *φ(x)* заданы аналитически, то параметр *dx* при вычислении их производных не нужен и в соответствующих подпрограммах его можно не использовать. В общем случае, когда аналитическое выражение для функции неизвестно, производная может быть вычислена с помощью конечных разностей, например:

```
double Fi_der(double x, double dx)
{// Вычисление производной с помощью
 // центральной разностной схемы
extern double Fi(double);
return((Fi(x+dx)- Fi(x-dx)/(2*dx));
}
```

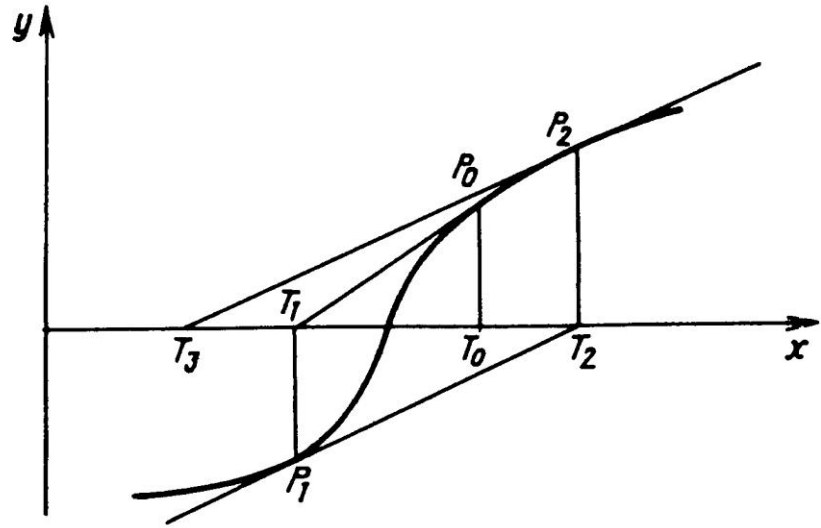

Рис. 4. «Раскачка» метода Ньютона

Применение метода Ньютона будет неудачным, если первая аппроксимация корня такова, что отношение $\varphi(x)/\varphi'(x)$ недостаточно мало, то есть мы «проскочим» корень уравнения $y=\varphi(x)$, см. рис. 4. Для того чтобы итерации сходились, в общем случае необходимо улучшить начальную аппроксимацию корня, - положение точки P_0.

2.4. Упражнения

1. Найти значения максимума и минимума функции $f(x) = x\,(x\text{-}1)^2$.

2. Найти значения максимума и минимума функции $f(x) = x\,/\,(x^2+1)$.

3. Показать, что минимальным значением функции $y(\theta) = (a\cos\theta + b\sin\theta)$ является $y = -\sqrt{a^2+b^2}$. Можно ли получить этот результат, не используя производных?

4. Равнобедренный треугольник с вертикальным углом 2θ вписан в окружность радиуса R. Найти выражение для площади треугольника как функции от θ и показать, что она максимальна, когда треугольник равносторонний, то есть $\theta = 30^0$.

5. Исследовать функцию $f(x) = x^{2/3} - 1$. Нарисовать её график. Показать, что $f(x)$ имеет минимум при $x = 0$. Чему равно значение $f'(x)$ при $x = 0$? Меняет ли знак $f'(x)$, если x возрастает при прохождении через 0 ?

6. Исследовать функцию $f(x) = |x|$. Найти её минимум. Что можно сказать относительно поведения $f'(x)$ в точке минимума ?

7. Исследовать точки перегиба функции $f(x) = x^4 - 14x^3 + 60x^2 - 70x$. Этот кажущийся простым пример иллюстрирует одну из классических задач. Требуется решить уравнение $f'(x) = 0$. В данном случае оно является кубическим уравнением, которое не так просто раскладывается на множители. Здесь надо применить метод Ньютона для нахождения точек перегиба функции $f(x)$, но предварительно надо найти хорошее приближение к ним.

8. Исследовать точки перегиба функции $f(x) = x_1^2 + 4x_1x_2 + 5x_2^2$.

9. Исследовать точки перегиба функции

$$f(x) = -x_1^2 - 6x_2^2 - 23x_3^2 - 4x_1x_2 + 6x_1x_3 + 20x_2x_3.$$

10. Пусть *f(X)* есть квадратичная функция вида

$$f(X) = a + b^T X + \frac{1}{2} X^T G X,$$

где *a* — константа; b — вектор, не зависящий от *X*, а *G* - положительно определенная симметричная матрица, не зависящая от *X*. Показать, что точка минимума квадратичной функции *f(X)* имеет координаты $X^* = -G^{-1}b$.

11. Показать, что функция $f(X) = (x_1 - a)^2 + (x_2 - b)^2 + (x_3 - c)^2$ имеет минимум в точке $X^*(a, b, c)$.

3. Методы поиска минимума функции одной переменной

Упражнение 7 раздела 2.4 иллюстрирует общую задачу, возникающую при классическом подходе. Уравнение $f'(x) = 0$ не решается простым способом, и поэтому мы вынуждены прибегать к численным методам. Рассмотрим несколько простых численных процедур, непосредственно локализирующих минимум функции *f(x)*.

С помощью численных методов мы непосредственно ищем минимум функции *f(x)* в некотором интервале $a \le x \le b$, в котором, как предполагается, лежит минимум, вычисляя значения функции в выбранных точках данного интервала. Несомненным достоинством поисковых методов такого рода является то, что они основаны лишь на вычислении значений функций. При этом не требуется, чтобы исследуемые функции были дифференцируемы; более того, допустимы случаи, когда функцию нельзя даже записать в аналитическом виде. Единственным требованием является возможность определения значений функции *f(x)* в заданных точках *x* с помощью прямых расчётов или имитационных экспериментов.

Вообще в процессе применения рассматриваемых методов поиска можно выделить два этапа:

этап отыскания границ интервала неопределённости, на котором реализуется процедура поиска границ достаточно широкого интервала, содержащего точку оптимума;

этап уменьшения интервала, на котором реализуется конечная последовательность преобразований исходного интервала с тем, чтобы уменьшить его длину до заранее установленной величины.

3.1.Отыскание границ интервала неопределённости

На этом этапе сначала выбирается исходная точка, а затем на основе правила исключения строится относительно широкий интервал, содержащий точку оптимума. Обычно поиск граничных точек такого интервала проводится с помощью эвристических методов поиска, хотя в ряде случаев можно также использовать методы экстраполяции. Обычно это шаговый процесс, при котором из некоторой начальной точки делается серия шагов по аргументу и вычисляется функция. Процесс продолжается до того момента, когда функция начнёт возрастать. Это признак того, что точка минимума пройдена и последние три точки задают интервал неопределённости, содержащий минимум функции, см. рис. 5.

В зависимости от способа выбора шага различают разные виды поиска. Если, например, шаг постоянный, то говорят о равномерном поиске. В соответствии с другим эвристическим методом, который был предложен Свенном [2,3], (k+1) пробная точка определяется по рекуррентной формуле

$$x_{k+1} = x_k + 2^k \cdot \Delta, \quad k = 0, 1, 2, \ldots \qquad (11)$$

где x_0 - произвольно выбранная начальная точка, Δ - подбираемая некоторым способом величина шага. Знак Δ определяется путём сравнения значений $f(x_0)$, $f(x_0+|\Delta|)$ и $f(x_0-|\Delta|)$. Если

$$f(x_0-|\Delta|) \geq f(x_0) \geq f(x_0+|\Delta|), \qquad (12)$$

то, согласно предположению об унимодальности функции, точка минимума должна располагаться правее точки x_0 и величина Δ выбирается положительной. Если изменить знаки неравенств на противоположные, то для Δ следует выбирать отрицательное значение. Если же

$$f(x_0-|\Delta|) \geq f(x_0) \leq f(x_0+|\Delta|), \qquad (13)$$

то точка минимума лежит между $(x_0-|\Delta|)$ и $(x_0+|\Delta|)$ и поиск граничных точек завершён. Случай, когда

$$f(x_0-|\Delta|) \leq f(x_0) \geq f(x_0+|\Delta|), \qquad (14)$$

противоречит предположению об унимодальности. Выполнение этого условия свидетельствует о том, что функция не является унимодальной.

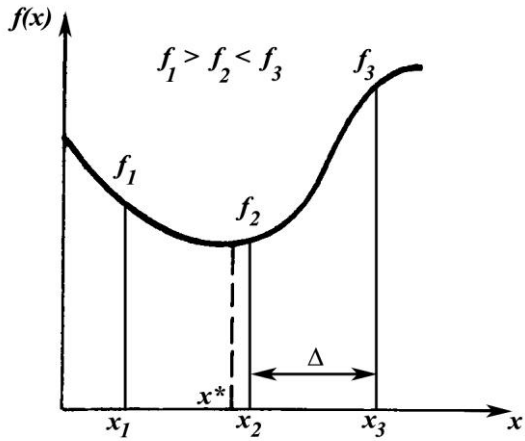

Рис. 5. Интервал неопределённости

Пример 3. Рассмотрим задачу минимизации функции $f(x)=(100 - x)^2$ при заданной начальной точке x_0 = 30 и величине шага $|\Delta|$ = 5. Знак Δ определяется на основе сравнения значений:

$f(x_0)$ = $f(30)$ = 4900;

$f(x_0+|\Delta|) = f(35) = 4225;$

$f(x_0 - |\Delta|) = f(25) = 5625.$

Так как $f(x_0-|\Delta|) > f(x_0) > f(x_0+|\Delta|)$,

то величина Δ должна быть положительной, а координата точки минимума x^* должна быть больше 30. Имеем $x_1 = x_0 + \Delta = 35$. Далее $x_2 = x_1 + 2\Delta = 45$, $f(45) = 3025 < f(x_1)$,

откуда $x^* > 35$.

$x_3=x_2 + 2^2\Delta = 65, \quad f(65) = 1225 < f(x_2)$,

откуда $x^* > 45$.

$x_4=x_3 + 2^3\Delta = 105, \quad f(105) = 25 < f(x_3)$,

откуда $x^* > 65$.

$x_5=x_4 + 2^4\Delta = 185, \quad f(185) = 7225 > f(x_4)$,

следовательно, $x^* < 185$. Таким образом, шесть шагов вычислений x^* позволили выявить интервал неопределённости $65 \le x^* \le 185$, в котором расположена точка минимума x^*. Заметим, что эффективность поиска граничных точек непосредственно зависит от величины шага Δ. Если Δ велико, то получаем грубые оценки координат граничных точек, и построенный интервал оказывается весьма широким. С другой стороны, если Δ мало, для определения граничных точек может потребоваться достаточно большой объем вычислений.

3.2. Уменьшение интервала неопределённости

После того как установлены границы интервала неопределённости, содержащего точку минимума, можно применить более сложную процедуру его уменьшения с целью получения уточнённых оценок координат минимума. Величина отрезка интервала, исключаемого на каждом шаге, зависит от расположения пробных точек x_1 и x_2 внутри интервала неопределённости. Поскольку местонахождение точки минимума

заранее неизвестно, целесообразно предположить, что размещение пробных точек должно обеспечивать уменьшение интервала в одном и том же отношении. Кроме того, в целях повышения эффективности алгоритма необходимо потребовать, чтобы указанное отношение было максимальным. Подобную стратегию иногда называют минимаксной стратегией поиска.

3.2.1. Метод деления интервала пополам (метод дихотомии)

Рассматриваемый метод позволяет исключать *в точности* половину интервала на каждой итерации. Иногда этот метод называют *трёхточечным поиском на равных интервалах,* поскольку его реализация основана на выборе трёх пробных точек, равномерно распределённых в интервале неопределённости. Приведём описание основных шагов поисковой процедуры, ориентированной на нахождение точки минимума функции $f(x)$ в интервале (a, b).

Шаг 1. Положить $x_m = (a+b)/2$ и $L = b - a$. Вычислить значение $f(x_m)$.

Шаг 2. Положить $x_1 = a + L/4$ и $x_2 = b - L/4$. Заметим, что точки x_1, x_m и x_2 делят интервал (a, b) на четыре равные части. Вычислить значения $f(x_1)$ и $f(x_2)$.

Шаг 3. Сравнить $f(x_1)$ и $f(x_m)$. Если $f(x_1) < f(x_m)$, то исключить интервал (x_m, b), положив $b = x_m$. Средней точкой нового интервала неопределённости становится точка x_1. Следовательно, необходимо положить $x_m = x_1$ и перейти к шагу 5.

Если $f(x_1) \geq f(x_m)$, то перейти к шагу 4.

Шаг 4. Сравнить $f(x_2)$ и $f(x_m)$. Если $f(x_2) < f(x_m)$, то исключить интервал (a, x_m), положив $a = x_m$. Так как средней точкой нового интервала становится точка x_2, то положить $x_m = x_2$ и перейти к шагу 5.

Если $f(x_2) \geq f(x_m)$, то исключить интервалы (a, x_1) и (x_2, b). Положить $a = x_1$ и $b = x_2$. Заметим, что x_m продолжает оставаться средней точкой нового интервала. Перейти к шагу 5.

Шаг 5. Вычислить $L=b - a$. Если величина $| L |$ мала, то закончить поиск. В противном случае вернуться к шагу 2.

Замечания

1. На каждой итерации алгоритма исключается в точности половина интервала поиска.

2. Средняя точка последовательно получаемых интервалов всегда совпадает с одной из пробных точек x_1, x_2 или x_m, найденных на предыдущей итерации. Следовательно, на каждой итерации требуется не более двух вычислений значения функции.

3. Если проведено n вычислений значения функции, то длина полученного интервала составляет $(1/2)^{n/2}$ величины исходного интервала.

4. В работе [3] показано, что из всех методов поиска на равных интервалах (двухточечный, трёхточечный, четырёхточечный и т. д.) трёхточечный поиск, или метод деления интервала пополам, отличается наибольшей эффективностью.

Пример 4. Метод деления интервала пополам

Минимизировать $f(x) = (100 - x)^2$ в интервале $60 \le x \le 150$. Здесь a=60, b=150 и L=150 – 60 = 90.

$x_m = (60+150)/2=105$.

И т е р а ц и я 1.

$x_1 = a + (L/4) = 60+(90/4) = 82.5$;

$x_2 = b - (L/4) = 150 - (90/4)=127.5$;

$f(82.5) = 306.25 > f(105) = 25$;

$f(127.5) = 756.25 > f(105) = 25$.

Таким образом, исключаются интервалы (60, 82.5) и (127.5, 150). Длина интервала поиска уменьшается с 90 до 45.

Итерация 2.

$a = 82.5; \quad b = 127.5; \quad x_m = 105.$

$L = 127.5 - 82.5 = 45;$

$x_1 = 82.5 + (45/4) = 93.5;$

$x_2 = 127.5 - (45/4) = 116.25;$

$f(93.75) = 39.06 > f(105) = 25;$

$f(116.25) = 264.06 > f(105) = 25.$

Таким образом, интервал неопределённости равен (93.75, 116.25).

Итерация 3.

$a = 93.75; \quad b = 116.25; \quad x_m = 105;$

$L = 116.25 - 93.75 = 22.5;$

$x_1 = 99.375; \quad x_2 = 110.625;$

$f(x_1) = 0.39 < f(105) = 25.$

Таким образом, исключается интервал (105, 116.25). Новый интервал неопределённости равен (93.75, 105), его средняя точка есть 99.375 (точка x_1 на итерации 3). Отметим, что за три итерации (шесть вычислений значения функции) исходный интервал поиска длины 90 уменьшился до величины $(90)\ (1/2)^3 = 11.25$.

Далее приведена программа на языке C++. Все вычисления проводятся с двойной точностью (16 десятичных цифр мантиссы чисел с плавающей точкой). Программа оформлена в виде подпрограммы и может вызываться из других программ.

```
#include <math.h>

double Devided_2(double *a, double *b)
{
// Программа поиска минимума на интервале [a, b]
// методом деления интервала пополам.
```

```
// Значение функции f(x) вычисляется в подпрограмме Fx(x)
// В результате интервал неопределённости уменьшается и
// его границы доступны в вызывающей программе.
// Также «наверх» передаётся минимальное значение функции.

extern double Fx(double);

double  f, f1, f2, fm, fa, fb, x1, x2, L,
        tolerance=0.00001;


// Значения функции на границах и в середине интервала
fa = Fx(*a);
fb = Fx(*b);
L = *b - *a;
xm = (*a + *b)/2.0;
fm = Fx(xm);

// Основной цикл поиска
do {
     x1 = *a + L/4.0;
     x2 = *b - L/4.0;
     f1 = Fx(x1);
     f2 = Fx(x2);

     if (f1 < fm) {//Исключаем интервал [xm,b]
         f = f1;
         *b = xm; fb = fm;
         xm = x2; fm = f2;
     } else {
         if (f2 < fm) {//Исключаем интервал [a,xm]
             f = f2;
             *a = xm; fa = fm;
             xm = x2; fm = f2;
         } else {//Исключаем интервалы [a,x1] и [x2, b]
             f = fm
             *a = x1; fa = f1;
             *b = x2; fb = f2;
         }
     }

     L = *b - *a;

} while (fabs(L) > tolerance);

return (f);
}
```

3.2.2. Метод золотого сечения.

Из проведённого обсуждения методов исключения отрезков интервала и минимаксных стратегий поиска можно сделать следующие выводы.

1. Если внутри интервала неопределённости количество пробных точек принимается равным двум, то их следует размещать на одинаковых расстояниях от середины интервала.

2. В соответствии с общей минимаксной стратегией пробные точки должны размещаться в интервале по симметричной схеме таким образом, чтобы отношение длины исключаемого отрезка интервала к величине интервала неопределённости оставалось постоянным.

3. На каждой итерации процедуры поиска должно вычисляться *только одно* значение функции в получаемой точке.

Руководствуясь этими выводами, рассмотрим симметричное расположение двух пробных точек на исходном интервале единичной длины, которое показано на рис. 6. (Выбор единичного интервала обусловлен соображениями удобства.) Пробные точки отстоят от граничных точек интервала на расстоянии τ.

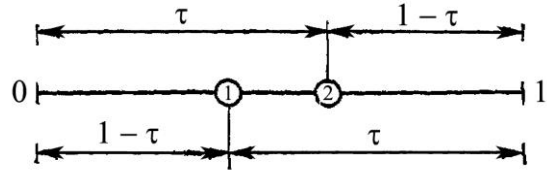

Рис. 6. Отрезки золотого сечения.

При таком симметричном расположении точек длина остающегося после исключения отрезка всегда равна τ независимо от того, какое из значений функции в пробных точках оказывается меньшим. Предположим, что исключается правый отрезок. На рис. 7 показано, что оставшийся отрезок длины τ содержит одну пробную точку, расположенную на расстоянии (1 - τ) от левой граничной точки.

Для того чтобы симметрия поискового образца сохранялась, расстояние (1 - τ) должно составлять τ -ю часть длины интервала (которая равна τ). При таком выборе τ следующая пробная точка размещается на расстоянии, равном τ -й части длины интервала, от правой граничной точки интервала, см. рис. 7.

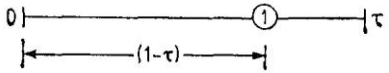

Рис. 7. Интервалы метода золотого сечения.

Отсюда следует, что при выборе τ в соответствии с условием $1 - \tau = \tau^2$ симметрия поискового образца, показанного на рис. 6, сохраняется при переходе к уменьшенному интервалу, который изображён на рис. 8. Решая это квадратное уравнение, получаем

$$\tau = (-1 \pm \sqrt{5}) / 2; \qquad (15)$$

откуда положительное решение τ = 0,61803... . Схема поиска, при которой пробные точки делят интервал в этом отношении, известна под названием *поиска с помощью метода золотого сечения.* Заметим, что после первых двух вычислений значений функции каждое последующее вычисление позволяет исключить отрезок, величина которого составляет (1 - τ)-ю долю от длины интервала поиска. Следовательно, если исходный интервал имеет единичную длину, то величина интервала, полученного в результате N вычислений значений функции, равна x^{N-1}. Можно показать, что поиск с помощью метода золотого сечения является асимптотически наиболее эффективным способом реализации минимаксной стратегии поиска.

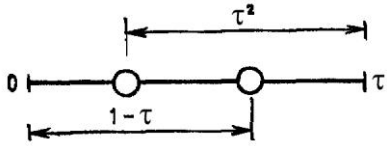

Рис. 8. Симметрия золотого сечения при уменьшении интервала

Алгоритм поиска минимума методов золотого сечения аналогичен методу деления интервала пополам, поэтому далее приведём только программу его реализации.

```
#include <math.h>

double Gold_Cut(double *a, double *b)
{
// Программа поиска минимума на интервале [a, b]
// методом золотого сечения.
// Значение функции f(x) вычисляется в подпрограмме Fx(x)
// В результате интервал неопределённости уменьшается и
// его границы доступны в вызывающей программе.
// Также «наверх» передаётся минимальное значение функции.

extern double Fx(double);

double  f, f1, f2, f3, x0, x1, x2, x3, L, t1, t2,
        tolerance=0.00001;

t1 = 0.3819660113; // (1 – tau) – левая точка внутри
t2 = (1.0 – t1);   // (tau)     – правая точка внутри
// Значения функции на границах и внутри интервала
x0 = *a;                f0 = Fx(x0);
x1 = *a + t1*(*b - *a); f1 = Fx(x1);
x2 = *a + t2*(*b - *a); f2 = Fx(x2);
x3 = *b;                f3 = Fx(x3);

// Основной цикл поиска
do {
     if (f1 <= f2) { //Исключаем отрезок [x2, x3]
         L = x2 – x0;
         x3 = x2; f3 = f2;
         x2 = x1; f2 = f1;
         x1 = x0 + t1*L; f = f1 = Fx(x1);
     } else {        // Исключаем отрезок [x0, x1]
             L = x3 – x1;
             x0 = x1; f1 = f0;
             x1 = x2; f1 = f2;
             x2 = x0 + t2*L; f = f2 = Fx(x2);
     }

} while (fabs(L) > tolerance);

*a = x0; *b = x3;
return (f);
}
```

Пример 5. Метод золотого сечения

Рассмотрим задачу из примера 4, в которой требуется минимизировать $f(x)=(100—x)^2$ в интервале $60 \le x \le 150$. Для того чтобы перейти к интервалу единичной длины, проведём замену переменной, положив $w = (x - 60) / 90$. Таким образом, задача принимает следующий вид: минимизировать $f(w)=(40 - 90w)^2$ при ограничении $0 \le w \le 1$.

И т е р а ц и я 1.

$I_1 = (0, 1)$; $L_1 = 1$. Проведём вычисления значений функции:

$w_1 = \tau = 0.618$; $f(w_1) = 244.0$.

$w_2 = 1— \tau = \tau^2 = 0.382$; $f(w_2) = 31.6$.

Так как $f(w_2) < f(w_1)$ и $w_2 < w_1$ интервал $w \ge w_1$ исключается.

И т е р а ц и я 2.

$I_2 = (0, 0.618)$; $L_2 = 0.618 = \tau$. Следующее вычисление значения функции проводится в точке

$w_3 = \tau - \tau^2 = \tau (1 - \tau) = \tau^3 = 0.236$; $f(w_3) = 352$.

Так как $f(w3) > f(w_2)$ и $w_3 < w_2$, интервал $w \le w_3$ исключается.

И т е р а ц и я 3.

$I_3 = (0.236, 0.618)$; $L_3 = 0.382 = \tau^2$.

Следующее вычисление значения функции проводится в точке, расположенной на расстоянии $\tau \times$ (длину полученного интервала) от левой граничной точки интервала, или на расстоянии $(1— \tau) \times$ (длину интервала) от правой граничной точки. Таким образом,

$w_4 = 0.618 - (1 - \tau) L_3 = 0.618 - \tau^2 L_3 = 0.618 - \tau^2(\tau^2) = 0{,}618 - \tau^4 = 0.472$;

$f(w_4) = 6.15$.

Так как $f(w_4) < f(w_2)$ и $w_4 > w_2$, интервал $w \le w_2$ исключается.

В результате получен следующий интервал неопределённости: $0.382 \le w \le 0.618$ для переменной w, или $94.4 < x \le 115.6$ для переменной x.

Если в процессе поиска будет проведено шесть вычислений значений функции, то длина результирующего интервала для переменной w станет равной

$$\tau^{N-1} = \tau^5 = 0.09;$$

что соответствует интервалу длиной 8.1 для переменной x. Для сравнения напомним, что в аналогичной ситуации метод деления интервала пополам привёл к получению интервала длиной 11.5.

Поиск с помощью метода золотого сечения может быть окончен либо исходя из заданного количества вычислений значений функции и, следовательно, величины интервала неопределённости, либо по достижении относительной точности искомого значения функции. Наиболее предпочтительным является использование обоих критериев одновременно. Однако в приведённой программе используется текущая величина интервала в качестве критерия окончания поиска. Для использования в качестве критерия относительного изменения функции модификация программы очевидна.

3.2.3. Квадратичная аппроксимация. Метод Пауэлла.

Применение ранее рассмотренных методов уменьшения интервалов неопределённости накладывает единственное требование на исследуемую функцию: она должна быть унимодальной. Следовательно, рассмотренные методы можно использовать для анализа как непрерывных, так и разрывных функций, а также в случаях, когда переменные принимают значения из дискретного множества. Логическая структура поиска с помощью методов исключения отрезков интервалов основана на простом сравнении значений функции в двух пробных точках. Кроме того, при таком сравнении в расчёт принимается только

отношение порядка на множестве значений функции и не учитывается величина разности между значениями функции. Теперь рассмотрим методы поиска, которые позволяют учесть относительные изменения значений функции и как следствие в ряде случаев оказываются более эффективными, чем методы исключения отрезков интервалов. Однако выигрыш в эффективности достигается ценой введения дополнительного требования, согласно которому исследуемые функции должны быть достаточно гладкими.

Основная идея рассматриваемых методов связана с возможностью аппроксимации гладкой функции полиномом и последующего использования аппроксимирующего полинома для оценивания координаты точки оптимума.

Необходимыми условиями эффективной реализации такого подхода являются унимодальность и непрерывность исследуемой функции. Согласно теореме Вейерштрасса об аппроксимации, если функция непрерывна в некотором интервале, то её с любой степенью точности можно аппроксимировать полиномом достаточно высокого порядка. Следовательно, если функция унимодальна и найден *полином*, который достаточно точно её аппроксимирует, то координату точки оптимума функции можно оценить путём вычисления координаты точки *оптимума полинома*. Согласно теореме Вейерштрасса, качество оценок координаты точки оптимума, получаемых с помощью аппроксимирующего полинома, можно повысить двумя способами: использованием полинома более высокого порядка и уменьшением интервала аппроксимации. Второй способ, вообще говоря, является более предпочтительным, поскольку построение аппроксимирующего полинома порядка выше третьего становится весьма сложной процедурой, тогда как уменьшение интервала в условиях, когда выполняется предположение об унимодальности функции, особой сложности не представляет.

Простейшим вариантом полиномиальной интерполяции является квадратичная аппроксимация, которая основана на том предположении, что функция, принимающая минимальное значение во внутренней точке интервала, должна быть, по крайней

мере, квадратичной. Если же функция линейная, то её оптимальное значение может достигаться только в одной из двух граничных точек интервала. Таким образом, при реализации метода оценивания с использованием квадратичной аппроксимации предполагается, что в ограниченном интервале можно аппроксимировать функцию квадратичным полиномом, а затем использовать построенную аппроксимирующую схему для оценивания координаты точки истинного минимума функции.

Если задана последовательность точек x_1, x_2, x_3 и известны соответствующие этим точкам значения функции f_1, f_2, и f_3, то можно определить постоянные величины a_0, a_1 и a_2 таким образом, что значения квадратичной функции

$$q(x) = a_0 + a_1 (x - x_1) + a_2 (x - x_1)(x - x_2) \qquad (16)$$

совпадут со значениями $f(x)$ в трёх указанных точках. Перейдём к вычислению $q(x)$ в каждой из трёх заданных точек. Прежде всего, так как

$$f_1 = f(x_1) = q(x_1) = a_0; \qquad (17)$$

имеем $a_0 = f_1$. Далее, поскольку

$$f_2 = f(x_2) = q(x_2) = f_1 + a_1 (x_2 - x_1); \qquad (18)$$

получаем $a_1 = (f_2 - f_1) / (x_2 - x_1)$. Наконец, при $x = x_3$:

$$f_3 = f(x_3) = q(x_3) = f_1 + (f_2 - f_1)/(x_2 - x_1)(x_3 - x_1) + a_2(x_3 - x_1)(x_3 - x_2). \qquad (19)$$

Разрешая последнее уравнение относительно a_2, получаем

$$a_2 = \frac{1}{x_3 - x_2}\left(\frac{f_3 - f_1}{x_3 - x_1} - \frac{f_2 - f_1}{x_2 - x_1}\right). \qquad (20)$$

Таким образом, по трём заданным точкам и соответствующим значениям функции можно оценить параметры a_0, a_1 и a_2 аппроксимирующего квадратичного полинома с помощью приведённых формул.

Если точность аппроксимации исследуемой функции в интервале от x_1 до x_3 с помощью квадратичного полинома оказывается достаточно высокой, то в соответствии с предложенной стратегией поиска построенный полином можно использовать для оценивания координаты точки оптимума. Напомним, что стационарные точки функции одной переменной определяются путём приравнивания к нулю её первой производной и последующего нахождения корней полученного таким образом уравнения, В данном случае из уравнения

$$\frac{dq}{dx} = a_1 + a_2(x - x_2) + a_2(x - x_1) = 0;$$

можно получить

$$\hat{x} = (x_2 + x_1)/2 - (a_1/2a_2).$$

Поскольку функция *f(x)* на рассматриваемом интервале обладает свойством унимодальности, а аппроксимирующий квадратичный полином также является унимодальной функцией, то можно ожидать, что величина $\hat{x}$ окажется приемлемой оценкой координаты точки истинного оптимума *x**.

Пример 6. Использование квадратичной аппроксимации

Покажем процедуру оценки координаты точки минимума функции $f(x)=2x^2+(16/x)$ в интервале $1 \le x \le 5$. Пусть $x_1 = 1$, $x_3 = 5$, а x_2 есть средняя точка интервала, то есть $x_2 = 3$. Вычислим соответствующие значения функции:

$f_1 = 18$; $f_2 = 23.33$; $f_3 = 53.2$.

Для того чтобы найти оценку $\hat{x}$, необходимо найти значения параметров a_1 и a_2 аппроксимирующей функции. Имеем:

$$a_1 = \frac{23.33 - 18}{3 - 1} = \frac{8}{3};$$

$$a_2 = \frac{1}{5-3}\left[\frac{53.2 - 18}{5 - 1} - \frac{8}{8}\right] = \frac{46}{15}.$$

Подстановка этих значений в формулу для x позволяет получить

$x = (3+1)/2—(8/3)/[2\ (46/15)] = 1.565.$

Точный минимум достигается при $x^* = 1.5874$.

Пауэллом [2] разработан метод, основанный на последовательном применении процедуры оценивания с использованием квадратичной аппроксимации. Схему алгоритма Пауэлла можно описать следующим образом. Пусть x_1 - начальная точка, Δx - выбранная величина шага по оси x.

Шаг 1. Вычислить $x_2 = x_1 + \Delta x$.

Шаг 2. Вычислить $f(x_1)$ и $f(x_2)$.

Шаг 3. Если $f(x_1) > f(x_2)$ положить $x_3 = x_1 + 2\ \Delta x$. Если $f(x_1) \leq f(x_2)$, то положить $x_3 = x_1 - \Delta x$.

Шаг 4. Вычислить $f(x_3)$ по формулам (17)-(20) и найти

$F_{min} = min\ (f_1, f_2, f_3)$;

x_{min} = точка x_i которая соответствует F_{min}.

Шаг 5. По трем точкам x_1, x_2, x_3 вычислить $\hat{x}$, используя формулу для оценивания с помощью квадратичной аппроксимации.

Шаг 6. Проверка на окончание поиска.

(а) Является ли разность $(F_{min} - f(\hat{x}))$ достаточно малой?

(б) Является ли разность $(x_{min} - \hat{x})$ достаточно малой?

Если оба условия выполняются, то закончить поиск. В противном случае перейти к шагу 7.

Шаг 7. Выбрать «наилучшую» точку из x_{min} и $\hat{x}$ и две точки по обе стороны от неё. Обозначить эти точки в естественном порядке x_1, x_2, x_3 и перейти к шагу 4.

Заметим, что при первой реализации шага 5 границы интервала, содержащего точку минимума, не обязательно оказываются внутри найденного интервала неопределённости. При этом полученная точка $\hat{x}$ может находиться за точкой x_3. Для того чтобы исключить возможность слишком большого экстраполяционного перемещения, следует провести после шага 5 дополнительную проверку и в случае, когда точка $\hat{x}$ находится слишком далеко от x_3, заменить $\hat{x}$ точкой, координата которой вычисляется с учётом заранее установленной длины шага.

Пример 7. Метод Пауэлла

Рассмотрим задачу из примера 6: минимизировать $f(x) = 2x^2 + (16/x)$.

Пусть начальная точка $x_1 = 1$ и длина шага $\Delta x = 1$. Для проверки на окончание поиска используются следующие параметры сходимости:

$$\left|\frac{\text{Разность значений } x}{x}\right| \le 3\times10^{-2}; \quad \left|\frac{\text{Разность значений } f}{f}\right| \le 3\times10^{-3}.$$

Итерация 1.

Шаг 1. $x_2 = x_1 + \Delta x = 2$.

Шаг 2. $f(x_1) = 18$; $f(x_2) = 16$.

Шаг 3. $f(x_1) > f(x_2)$, следовательно, положить $x_3 = 1 + 2 = 3$.

Шаг 4. $f(x_3) = 23.33$; $F_{min} = 16$; $x_{min} = x_2$.

Шаг 5.

$$a_1 = \frac{16-18}{2-1} = -2;$$

$$a_2 = \frac{1}{3-2}\left(\frac{23.33-18}{3-1} - a_1\right) = \frac{5.33}{2} + 2 = 4.665;$$

$$\hat{x} = \frac{1+2}{2} - \frac{(-2)}{2(4.665)} = 1.5 + \frac{1}{4.665} = 1.714;$$
$$f(\hat{x}) = 15.210.$$

Шаг 6. Проверка на окончание поиска:

(а) $$\left|\frac{16-15.210}{15.210}\right| = 0.0519 > 0.003.$$

Следовательно, продолжаем поиск.

Шаг 7. Выбираем $\hat{x}$ как «наилучшую» точку, а $x_1 = 1$ и $x_2 = 2$ - как точки, которые её окружают. Обозначаем эти точки в естественном порядке и переходим к итерации 2, которая начинается с шага 4.

Итерация 2

Шаг 4. $x_1 = 1$; $f_1 = 18$, $x_2 = 1.714$, $f_2 = 15.210 = F_{min}$; $x_{min} = x_2$; x_3=2; $f_3 = 16$.

Шаг 5.

$$a_1 = \frac{15.142-18}{1.65-1} = -4.397;$$
$$a_2 = \frac{1}{1.714-1.650}\left(\frac{15.210-18}{1.714-1} - (-4.397)\right) = 7.647;$$

$$\hat{x} = \frac{2.65}{2} - \frac{(-4.397)}{2(7.647)} = 1.6125;$$
$$f(\hat{x}) = 15.123.$$

Шаг 6. Проверка на окончание поиска:

(а) $$\left|\frac{15.142-15.123}{15.123}\right| = 0.0013 < 0.003.$$

(б) $\left|\frac{1.65-1.6125}{1.6125}\right| = 0.023 < 0.03.$

Следовательно, поиск закончен.

Программа, реализующая метод Пауэлла выглядит следующим образом.

```
#include <math.h>

double Powell(double *a, double *b)
{
// Программа поиска минимума на интервале [a, b]
// методом квадратичной аппроксимации Пауэлла.
// Значение функции f(x) вычисляется в подпрограмме Fx(x)
// В результате интервал неопределённости уменьшается и
// его границы доступны в вызывающей программе.
// Также «наверх» передаётся минимальное значение функции.

extern double Fx(double);

double  dx, x[4], f[4], xt, ft, dn, nm,
        tolerance=0.001;
int j,k, s1, s2, s3;

dx = 0.05; // Шаг
// Начать процесс с первых трёх точек
x[0] = *a;       f[0] = Fx(x[0]);
x[1] = *a + dx;  f[1] = Fx(x[1]);
if (f[0] < f[1]) x[2] = *a - dx; else x[2] = *a + 2.0*dx;
f[2] = Fx(x[2]);
//Первый интерполированный минимум
 dn = (x[1]-x[2])*f[0];
dn += (x[2]-x[0])*f[1] + (x[0]-x[1])*f[2];
 nm = (x[1]*x[1]-x[2]*x[2])*f[0];
nm += (x[2]*x[2]-x[0]*x[0])*f[1];
nm += (x[0]*x[0]-x[1]*x[1])*f[2];

x[3] = nm /(2.0*dn); f[3] = Fx(x[3]);

while(1) {//Бесконечный цикл интерполяции
       for (j = 0; j < 3; j++)
           for (k = 0;k < 4; k++) {
               if (f[j] > f[k] {//Упорядочить x и f
                   xt = x[j]; x[j] = x[k]; x[k] = xt;
                   ft = f[j]; f[j] = f[k]; f[k] = ft;
```

```
            }//Конец оператора if(  )
        } // Конец циклов по j и k

    if (fabs(x[0]-x[1]) < tolerance) {
        // Точность по величине интервала достигнута.
        // Выход из цикла интерполяции.
       goto EXIT_;
    }
    // Точность по интервалу не достигнута,
    // создаём новую точку
    s1 = sgn(x[1]-x[0]); //sng() =  1,  если скобки > 0
    s2 = sgn(x[2]-x[0]); //sng() =  0,  если скобки = 0
    s3 = sgn(x[3]-x[0]); //sng() = -1, если скобки < 0
    if (s1 == s2 && s1 == (-s3)) {
        x[2] = x[3];
        f[2] = f[3];
    }
    // Делаем вторую интерполяцию и
    // переходим на упорядочивание
    dn = (x[1]-x[2])*f[0]+(x[2]-x[0])*f[1]+(x[0]-
    x[1])*f[2];
    ft  = (f[0]-f[1])/(2.0*dn);
    ft *= (x[1]-x[2])*(x[2]-x[0]);
    x[3] = (x[0]+x[1])/2.0 + ft;
    f[3] = Fx(x[3]);

}//Конец бесконечного цикла интерполяции

EXIT_:
    *a = x[0];
    *b = x[1];
    return (f[0]);
}
```

Приведённая программа проверяет только достижение требуемой точности по величине интервала неопределённости, то есть по условию (а), однако дополнение для проверки условия (б) сделать достаточно просто, и эту модификацию программы мы предоставляем читателю.

3.3. Упражнения

1. Применить метод деления интервала пополам и выполнить 10 вычислений функции для определения минимума функции $2x^2+3e^{-x}$ на интервале (0; 1).

2. Применить метод Пауэлла для нахождения минимума функции $x^4 - 14x^3 + 60x^2 - 70x$ на интервале (5; 7) с точностью 0.01. Сколько раз необходимо вычислить функцию?

3. Использовать метод золотого сечения для определения минимума функции $2x^2+3e^{-x}$ с точностью до двух десятичных цифр. В качестве начального интервала неопределённости задать интервал (0; 1). Сколько раз необходимо вычислить функцию? Сравнить с упражнением 1.

4. Применить метод Пауэлла для определения точки минимума функции $(-e^{-x} \cdot \ln(x))$ на интервале (1; 3) с точностью 0.001. (Проверить, не используются ли отрицательные значения *x,* ибо в противном случае возникнет ошибка при работе программы.)

5. Являются ли методы исключения отрезков интервалов в целом более эффективными, чем методы оценивания с использованием полиномов? Почему?

6. При реализации поисковых методов рекомендуется принимать решение об окончании поиска на основе проверок как величины разности значений переменной, так и величины разности значений функции. Возможна ли ситуация, когда результат одной из проверок указывает на сходимость к точке минимума, тогда как полученная точка в действительности минимуму не соответствует? Пояснить ответ рисунком.

7. Транспортное средство, которое должно курсировать на заданное расстояние, конструируется с таким расчётом, чтобы с его помощью можно было перевозить заданное количество груза в тоннах *(G)* в течение дня. Пусть стоимость транспортного средства без двигателей прямо пропорциональна грузоподъёмности транспорта (g), а стоимость двигателей плюс стоимость расхода топлива прямо пропорциональна произведению грузоподъёмности на скорость движения *(v)* в кубе. Показать, что полная стоимость транспортного средства минимальна в том случае, когда стоимость транспорта без двигателей и топлива в два раза превышает стоимость двигателей плюс стоимость топлива. (Временем погрузки и разгрузки можно пренебречь, то есть считается, что транспорт курсирует без остановок.)

4. Поисковые методы, не использующие производные

В отличие от методов, использующих производные (или их аппроксимации), здесь рассмотрим методы оптимизации, не использующие производные. Эти методы обычно называют *методами поиска,* В типичном методе поиска направления минимизации полностью определяются на основании последовательных вычислений целевой функции *f (x)*.

Как правило, при решении задач нелинейного программирования при отсутствии ограничений градиентные методы и методы, использующие вторые производные, сходятся быстрее, чем методы поиска. Тем не менее, применяя на практике методы, использующие производные, приходится сталкиваться с двумя главными препятствиями. Во-первых, в задачах с достаточно большим числом переменных довольно трудно или даже невозможно получить производные в виде аналитических функций, необходимых для градиентного алгоритма или алгоритма, использующего производные второго порядка. Хотя вычисление аналитических производных можно заменить вычислением производных с помощью разностных схем, как в программе раздела 2.3, возникающая при этом ошибка, особенно в окрестности экстремума, может ограничить применение подобной аппроксимации. Во всяком случае, методы поиска не требуют регулярности и непрерывности целевой функции и существования производных. Вторым обстоятельством, правда, связанным с предыдущей проблемой, является то, что при использовании методов оптимизации, основанных на вычислении первых и при необходимости вторых производных, требуется по сравнению с методами поиска довольно большое время на подготовку задачи к решению.

Вследствие изложенных выше трудностей были разработаны алгоритмы оптимизации, использующие прямой поиск, которые, хотя и медленнее реализуются в случае простых задач, на практике могут оказаться более удовлетворительными с точки зрения пользователя, чем градиентные методы или методы, использующие производные. Кроме того, при преобразовании задачи с ограничениями (задачи условной оптимизации) к безусловно экстремальной форме (к задаче без ограничений) с помощью

штрафных функций [1] построенная целевая функция приобретает сложную топографию овражного характера, на которой методы, использующие производные, останавливаются преждевременно, не доходя даже до локального экстремума.

Целесообразность выбора того или иного алгоритма поиска экстремума для практического использования определяется эффективностью этого алгоритма при решении конкретного класса задач. К числу важных критериев, используемых при оценке эффективности поисковых алгоритмов, относятся следующие:

время, необходимое для реализации вычислительных процедур, или количество машинных операций;

потребная оперативная память для реализации алгоритма;

допустимая степень сложности задачи - предельное количество проектных переменных и ограничений;

точность найденного решения по отношению к оптимальному значению;

простота компьютерной программы, реализующей алгоритм;

необходимость использования производных, которые в технических задачах практически всегда приходится брать численно с помощью конечных разностей, с вынужденной потерей точности и увеличением времени поиска.

Комплексная оценка различных алгоритмов нелинейного программирования по перечисленным критериям привела к выбору для рассмотрения лишь двух алгоритмов, которые на сегодняшний день являются наиболее предпочтительными для решения организационно-технических задач с топографиями целевой функции любой сложности.

4.1. Прямой поиск. Метод Хука-Дживса

По существу методы поиска простейшего типа заключаются в изменении каждый раз одной переменной, тогда как другие

остаются постоянными, пока не будет достигнут минимум. Например, в одном из таких методов переменная x_1 устанавливается постоянной, а x_2 изменяют до тех пор, пока не будет получен минимум. Здесь применяются алгоритмы поиска минимума функции одной переменной, рассмотренные ранее. Затем, сохраняя новое значение x_2 постоянным, изменяют x_1 пока не будет достигнут оптимум при выбранном значении x_2 и т. д. Так работает метод покоординатного спуска. Однако такой алгоритм работает плохо, если имеет место взаимодействие между x_1 и x_2, то есть, если в выражение для целевой функции входят члены, содержащие произведение x_1x_2. Таким образом, такой метод нельзя рекомендовать, если пользователь не имеет дела с целевой функцией, в которой взаимодействия не существенны.

Хук и Дживс [2] предложили другую, логически простую стратегию поиска, использующую априорные сведения и в то же время отвергающую устаревшую информацию относительно характера топологии целевой функции в эвклидовом пространстве проектных переменных E^n [1]. Этот алгоритм включает два основных этапа: «исследующий поиск» вокруг базисной точки и «поиск по образцу», то есть в направлении, выбранном для минимизации.

Рассматриваемый алгоритм прямого поиска метода Хука-Дживса с небольшими нашими модификациями выглядит следующим образом.

1. В пространстве проектных переменных задать начальную точку $X_0 = \left\{ x_1^0, x_2^0 ... x_n^0 \right\}^T$ и вектор приращений $\Delta X = \left\{ \Delta x_1^0, \Delta x_2^0 ... \Delta x_n^0 \right\}^T$. Желательно, чтобы начальная точка лежала внутри допустимой области проектных переменных. Вычислить целевую функцию $f_0(X_0)$.

2. Из начальной точки провести исследующий поиск, при котором циклически изменяется КАЖДАЯ проектная переменная

$$x_i^1 = x_i^0 + \Delta x_i^0, \quad \text{i} = 1,2...\text{n}. \tag{21}$$

и каждый раз вычисляется целевая функция $f_i(X)$. Если приращение i-ой переменной не улучшает целевую функцию, то Δx_i^0 изменить на $\left(-\Delta x_i^0\right)$ и значение $f_i(X)$ проверить, как и ранее. Если значение $f_i(X)$ не улучшается ни при $\left(x_i^0+\Delta x_i^0\right)$, ни при $\left(x_i^0-\Delta x_i^0\right)$, то x_i^0 оставить без изменений.

На каждом шаге по независимой проектной переменной x_i значение целевой функции сравнивается с её значением в *предыдущей* точке. Если целевая функция улучшается на данном шаге, то её старое значение заменить на новое при последующих сравнениях. Однако, если проведённое возмущение по x_i неудачно, то сохранить прежнее значение $f(X)$.

Если ни одно изменение x_i, i=1,2,...n не улучшило целевую функцию, то пробные шаги Δx_i уменьшить в два раза: $\Delta x_i = \Delta x_i / 2.0$. Если

$$\| \Delta X \| \le \varepsilon , \qquad (22)$$

где ε - малая, наперёд заданная величина, то осуществить выход из алгоритма; в противном случае исследующий поиск начать вновь.

В результате отыскивается точка $X_1 = \left\{ x_1^1, x_2^1 ... x_n^1 \right\}^T$ с лучшим значением целевой функции $f_1(X_1)$.

При программной реализации этого пункта алгоритма пробные шаги по координатам x_i, i=1,2...n обычно делаются до тех пор, пока не начинается ухудшение целевой функции, см. раздел 3.1. Считается, что тем самым увеличивается скорость сходимости, поскольку новая точка X_1 будет отстоять от X_0 на максимально возможное расстояние. В овражной ситуации этот приём, напротив, уменьшает скорость сходимости алгоритма, поскольку диагональный шаг из следующего пункта, как правило, не приводит к успеху и алгоритм приближается по своей работе к методу координатного спуска [3]. Поэтому по каждой координате x_i, i=1,2...n; будем делать только один пробный шаг.

Взаимное расположение точек X_1 и X_0 указывает новое направление в пространстве проектных переменных, которое может привести к успеху. В этом направлении делается диагональный шаг, - в формулировке Хука-Дживса - *поиск по образцу*:

$$X_2 = 2\,X_1 - X_0\,; \qquad (23)$$

и в точке X_2 вычисляется целевая функция $f_2(X_2)$.

4. Успех или неудачу шага по образцу оценить по результатам исследующего поиска из новой базовой точки X_2. Он аналогичен поиску, описанному в пункте 2 настоящего алгоритма, и в результате определяется точка X_3 и $f_3(X_3)$.

5. Если значение $f_3(X_3)$ лучше, чем $f_1(X_1)$, то поиск по образцу удачен, его провести ещё раз с некоторой корректировкой направления: $X_0=X_1;\ \ X_1=X_3;\ \ f_0(X_0) = f_1(X_1);\ \ f_1(X_1) = f_3(X_3);$ и осуществить переход к пункту 3. В противном случае диагональный шаг считается неудачным и осуществить переход к пункту 2 с $X_0=X_1;$ и $f_0(X_0) = f_1(X_1)$.

Данный пункт алгоритма отличен от соответствующего пункта метода Хука-Дживса тем, что мы корректируем новое направление диагонального шага. В оригинале метода переход к пункту 3 проходит с $X_1=X_3;\ \ f_1(X_1) = f_3(X_3);$ и неизменными значениями X_0 и $f_0(X)$. Внесённая нами модификация метода делает его пригодным для поиска экстремума при овражной топографии целевой функции.

Упрощённая блок-схема алгоритма без детализации содержимого блоков представлена на рис. 9.

Пример 8. *Максимизация* (без ограничений) прямым поиском по Хуку и Дживсу.

Здесь мы специально показываем пример максимизации функции, чтобы подчеркнуть универсальность методов нелинейного программирования. Итак, максимизировать целевую функцию

$$f(X) = \frac{1}{(x_1 + 1)^2 + x_2^2};$$

начиная из $X^{(0)}$ = [2.00; 2.80]^T с начальным ΔX, равным [0.60; 0.84]^T. Исходное значение f (2.00; 2,80) в базисной точке $X^{(0)}$ равно 0.059.

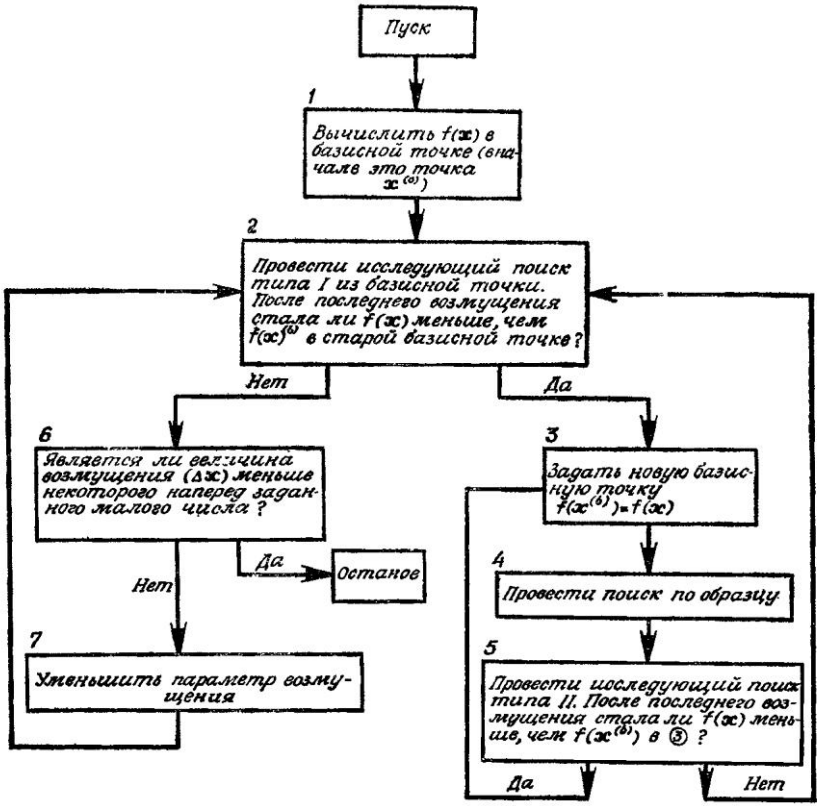

Рис. 9. Упрощённая блок-схема алгоритма Хука-Дживса

Сначала проведём исследующий поиск типа I для определения удачного направления. Такая процедура называется исследующим поиском типа I в противоположность исследующему поиску типа II, который следует за поиском по образцу. После проведения исследующего поиска типа II принимается решение по

поводу того, было ли предыдущее движение по образцу успешным или неудачным.

$x_1^{(1)} = 2.0 + 0.60 = 2.60;$ $f(2.60; 2.80) = 0.048$ (неудача);

$x_1^{(1)} = 2.0 – 0.60 = 1.40;$ $f(1.40; 2.80)$^0.073 (успех);

$x_2^{(1)} = 2.80 + 0.84 = 3.64;$ $f(1.50; 3.64) = 0.052$ (неудача);

$x_2^{(1)} = 2.80 - 0.84 = 1.96;$ $f(1.40; 1.96) = 0.104$ (успех).

Исследующий поиск оказался удачным. Заметим, что при каждом поиске выбирается последний удачный вектор X. Новым базисным вектором будет (1.40; 1.96).

Теперь из точки (1.40; 1.96) проводится диагональный шаг или поиск по образцу в соответствии с правилом:

$$x_i^{(k+1)} = 2\, x_i^{(k)} - x_i^{(b)};$$

где $x_i^{(b)}$ – i-й компонент предыдущего базисного вектора X. В данном случае это начальный вектор $X^{(0)}$.

$x_1^{(2)} = 2\ (1.40) - 2.00 = 0.80;$

$x_2^{(2)} = 2\ (1.96) - 2.80 = 1.12;$

$f(0.8; 1.12) = 0.22.$

Наконец проводится исследующий поиск типа II; неудача или успех его оценивается путём сравнения с $f(0.8; 1.12) = 0.22$:

$x_1^{(3)} = 0.80 + 0.60 = 1.40;$ $f(1.40; 1.12) = 0.14$ (неудача);

$x_1^{(3)} = 0.80 - 0.60 = 0.20;$ $f(0.20; 1.12) = 0.38$ (успех);

$x_2^{(3)} = 1.12 + 0.84 = 1.96;$ $f(0.20; 1.96) = 0.19$ (неудача);

$x_2^{(3)} = 1.12 - 0.84 = 0.28;$ $f(0.20; 0.28) = 0.67$ (успех).

Чтобы определить, оказался ли поиск по образцу успешным, сравнивают $f(0.20; 0.28) = 0.67$ с $f(1.40; 1.96) = 0.104$. Поскольку

поиск по образцу успешен, то новой базисной точкой будет $X^{(3)} = [0.20; 0.28]^T$; при этом старая базисная точка представлена вектором $X^{(1)} = [1.40; 1.96]^T$.

Затем вновь проводится поиск по образцу:

$x_1^{(4)} = 2\ (0.20) - 1.40 = -1.00;$

$x_2^{(4)} = 2\ (0.28) – 1.96 = -1.40;$

$f\ (-1.00; -1.40) = 0.51.$

После этого проводится исследующий поиск типа II:

$x_1^{(5)} = -1.00 + 0.60 = -0.40;$ f(-0.40; -1.40) = 0.43 (неудача);

$x_1^{(5)} = -1.00 - 0.60 = -1.60;$ f (-1.60; -1.40) = 0.43 (неудача);

$x_2^{(5)} = -1.40 + 0.84 = -0{,}56,$ f(-1.00; -0.56) = 3.18 (успех).

Поскольку $f\ (-1.00; -0.56) = 3.18 > f\ (0.20; 0.28) = 0.67$; поиск по образцу представляется успешным и $X^{(5)}= [-1.00; -0.56]^T$ становится новой базисной точкой, а $X^{(3)}$ — старой базисной точкой.

Эта последовательность поисков продолжается до тех пор, пока не будет достигнута ситуация, при которой в конце исследующего поиска типа II значение $f\ (X)$ окажется меньшим, чем значение $f\ (X^{(b)})$ в последней базисной точке. Тогда, если даже исследующий поиск типа II является успешным при одном или более возмущениях, говорят, что последний поиск по образцу неудачен и проводят из предыдущей базисной точки исследующий поиск типа I для определения нового удачного направления,

В иллюстративных целях продолжим поиск из $X^{(5)}= [-1.00; -0.56]^T$, см. рис. 10.

Поиск по образцу:

$x_1^{(6)} = 2\ (-1.00) – 0.20 = -2.20;$

$x_2^{(6)} = 2\ (-0.56) – 0.28 = -1.40;$

f (-2.20; -1.40) = 0.29.

Исследующий поиск типа II:

$x_1^{(7)}$ = -2.20 + 0.60 = -1.60; f (-1.60; -1.40) = 0.43 (успех);

$x_2^{(7)}$ = -1.40 + 0.84 = -0.56; f (-1.60; -0.56) = 1.49 (успех).

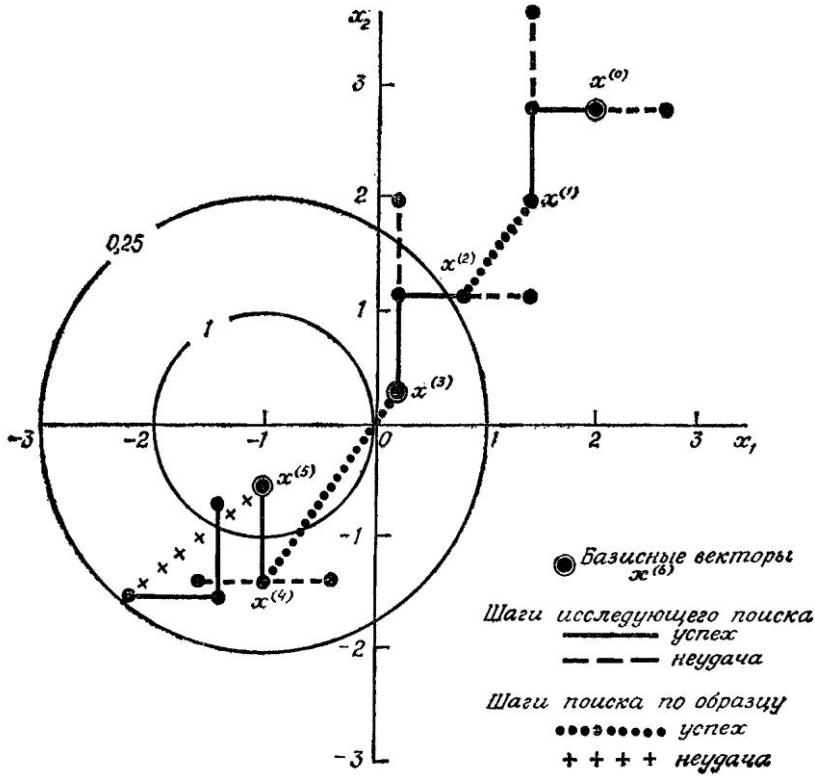

Рис. 10. Поиск максимума методом Хука-Дживса

Однако поскольку f (-1.60; -0.56) = 1.49 < f (-1.00; -0.56) = 3.18; то, несмотря на то, что исследующий поиск типа II оказался успешным, поиск по образцу считается неудачным, и из точки $X^{(5)}$ = [-1.00; -0.56] начинают исследующий поиск типа I.

Когда достигается стадия, на которой ни исследующий поиск типа I, ни поиск по образцу (вместе с исследующим поиском типа II) не являются успешными в любом координатном направлении, говорят, что они оба неудачны и возмущение ΔX в оригинале метода Хука-Дживса уменьшают следующим образом:

$$\Delta x_{i,\,новое} = \Delta x_{i,\,\text{предыдущее}} \frac{\Delta x_i^{(0)}}{e^{\xi}}; \qquad (24)$$

где ξ - число последовательных неудач в исследующих поисках при данной величине шага, начиная от последнего успешного исследующего поиска.

В этом примере максимум функции равен бесконечности $f(X) \to \infty$ при $x_1 \to (-1)$ и $x_2 \to 0$.

Программа на языке C++, реализующая алгоритм Хука-Дживса, как и ранее, представлена в виде подпрограммы, и, кроме того, исследующий поиск также реализован в виде подпрограммы. Для отладки программы даны строки комментариев, которые вполне работоспособны, если убрать // в начале комментария.

```
#include <io.h>
#include <stdio.h>
#include <fcntl.h>
#include <stat.h>
#include <process.h>
#include <math.h>

int Hooke_Jeevs(int np, double *x, double *dx, *fmin)
{
//*****************************************************
//  Программа минимизации функции методом Хука-Дживса
//*****************************************************
//     np   - количество проектных переменных
//     x[]  - значения проектных переменных
//     dx[] - приращения проектных переменных
//     fmin - найденный минимум целевой функции
//
//-----------------------------------------------------
// Вызываемые подпрограммы
//     Fx() - вычисление целевой функции
//     poisk() - исследующий поиск из базовой точки
```

```
//     pr_Fx() – отладочная печать
//-------------------------------------------------------

extern void pr_Fx(int ,double*, double, unsigned, char*);
extern int Fx(int, double*, double*, unsigned*);
extern int poisk(int, double*, double*, double*, double*,
                 double*, unsigned*);

       double *x0, *x1, *x2, *x3;
       double f0,f1,f2,f3, nor_dx,
       eps = 1.0e-04; //Предельно допустимый шаг поиска

int n, i, err;
unsigned jch = 0U;   // Счётчик количества вычислений
                     // целевой функции

// Выделение памяти под массивы
x0 = (double*)calloc(np, sizeof (double));
x1 = (double*)calloc(np, sizeof (double));
x2 = (double*)calloc(np, sizeof (double));
x3 = (double*)calloc(np, sizeof (double));

if (x0 == NULL || x1 == NULL || x2 == NULL || x3 == NULL)
       return (-1); //Аварийный выход: не хватает памяти

//***************** ОТЛАДКА ***********************
//     n = 2; x0[0] = -1.2; x0[1] = 1.0;
//     dx[0] =  0.8; dx[1] = 0.8;
//********** на функции Розенброка  ***************

//***************** ОТЛАДКА ***********************
//     n = 2; x0[0] = 0.5; x0[1] = 0.5;
//     dx[0] =  0.8; dx[1] = 0.8;
//******** на функции Изона-Фентона *************

//---------------------------------------------
// Начальные присвоения
//---------------------------------------------
n = np;
for(i = 0; i < n; i++) x0[i] = x[i];

   err = Fx(n,x0,&f0,&jch); // Вычисление целевой функции
   if (err < 0) goto ER0;

P0: err = poisk(n,dx,x0,&f0,x1,&f1,&jch); // Поиск из x0
       if (err < 0) goto ER0;

//  pr_Fx(n,x0,f0,jch,"Функция Розенброка"); // ОТЛАДКА
```

```
if (f1 == f0) { // Поиск неудачен
    for(i = 0; i < n; i++) dx[i] *= 0.5; // Уменьшаем шаг

       nor_dx = 0.0;                      // Не мал ли шаг ?
       for(i = 0; i < n; i++) nor_dx += dx[i] * dx[i];
       nor_dx = sqrt(nor_dx);
       if (nor_dx <= eps) goto VIX; // Выход по размеру шага

           goto P0;
}

// Диагональный шаг или поиск по образцу
 D0:   for(i = 0; i < n; i++) x2[i] = 2.0 * x1[i] - x0[i];
       err = Fx(n,x2,&f2,&jch); //Вычисление целевой функции
         if (err < 0) goto ER0;

// pr_Fx(n,x0,f0,jch,"Функция Розенброка");  // ОТЛАДКА

    /* Удачен ли диагональный шаг ? */
    err = poisk(n,dx,x2,&f2,x3,&f3,&jch); /* Поиск из т. X2
    if (err < 0) goto ER0;

       if (f3 >= f1) { // Диагональный шаг неудачен, ищем
                       // новое направление
           for(i = 0; i < n; i++) x0[i] = x1[i];
           f0 = f1;
           goto P0;
       } else {  // Диагональный шаг удачен, шагнем еще...
       for(i = 0; i < n; i++) {
             x0[i] = x1[i]; x1[i] = x3[i];
       }
           f0 = f1; f1 = f3;
           goto D0;
       }

 ER0: //Здесь должно быть диагностическое сообщение
       return (-1);

 VIX: //Нормальный выход из программы
       for(i = 0; i < n; i++) x[i] = x0[i];
       *fmin = f0;
return (0);

}

//===========================================================
int poisk(int n, double *dx, double *xo, double *fo,
```

```
                double *xn, double *fn, unsigned *ch)
{
/*********************************************************/
// Исследующий поиск из точки XO,FO и поиск точки XN,FN
      int j, flag, err;
      double f1;
//-------------------------------------------------------
      for (j = 0; j < n; j++) xn[j] = xo[j];
      *fn = *fo;

      for(j = 0; j < n; j++) {
// Циклическое изменение переменных и анализ последствий...
          xn[j] = xo[j] + dx[j];
          flag = 0;
  Z0:  err = Fx(n,xn,&f1,ch); // Вычисление целевой функции
          if (err < 0) return(-1);
          if (f1 < *fn) { *fn = f1; goto NEXT; }
          if (flag == 0) { flag = 1;
            xn[j] = xo[j] - dx[j];
            goto Z0;
          }
          xn[j] = xo[j];
   NEXT: ;
      }
    return(0);
}
//===========================================================
int Fx(int n,double *x, double *f, unsigned *ch)
      // Вычисление целевой функции
{
//------------- ОТЛАДКА на функции Розенброка -------------
//     *f = 100.0 * (x[1]-x[0]*x[0]) * (x[1]-x[0]*x[0]) +
//              (1.0-x[0])*(1.0-x[0]);
//-----------------------------------------------------------
//------------ ОТЛАДКА на функции Изона-Фентона -----------
//   *f = (12.0 + x[0]*x[0] + (1.0+x[1]*x[1])/(x[0]*x[0]) +
//        (x[0]*x[0]*x[1]*x[1]+100.0)/
//        (x[0]*x[0]*x[1]*x[1]*x[0]*x[0]*x[1]*x[1]))/10.0;
//-----------------------------------------------------------
// Здесь должна быть Ваша целевая функция
//
//-----------------------------------------------------------
      *ch += 1U;
      return(1);
}

//**** ОТЛАДОЧНАЯ ПЕЧАТЬ для тестовых функций *****
void pr_Fx(int n,double *x, double f, unsigned ch,
```

```
              char *txt)
{ // Печать текущих параметров
       FILE *Print;
       char t[64];

       Print = fopen("Debugging.txt","a+");

       fprintf(Print,"%s \n",txt);
       sprintf(Print," X[1] = %12g",x[0]);
       sprintf(Print," X[2] = %12g",x[1]);
       sprintf(Print," F    = %12g",f);
       sprintf(Print," Счетчик = %u \n",ch);


  return; //***** Конец ОТЛАДОЧНОЙ ПЕЧАТИ ****
}
```

Исследования программы на классических тестовых функциях, а именно, на функции Розенброка и функции Изона-Фентона [2] показали высокую скорость предложенной модификации метода. По данным [4, кн.1, стр.136-137] лучший метод достигает минимума функции Розенброка за 204, а функции Изона-Фентона за 92 вычисления целевой функции. Предложенный модифицированный алгоритм достиг экстремумов соответственно за 166 и 55 вычислений целевой функции.

4.2. Метод деформируемого многогранника.

Нелдер и Мид [2] предложили метод поиска, несколько более сложный по сравнению с прямым поиском Хука-Дживса, но оказавшийся весьма эффективным и легко осуществляемым на на компьютере. Чтобы читатель смог оценить стратегию Нелдера и Мида, кратко опишем симплексный поиск Спендли, Хекста и Химсворта [2], разработанный в связи со статистическим планированием эксперимента.

Вспомним, что регулярные многогранники в эвклидовом пространстве E^n являются симплексами. Например, как видно из рис. 11, для случая двух переменных регулярный симплекс представляет собой равносторонний треугольник (три точки); в случае трёх переменных регулярный симплекс представляет собой тетраэдр (четыре точки) и т. д.

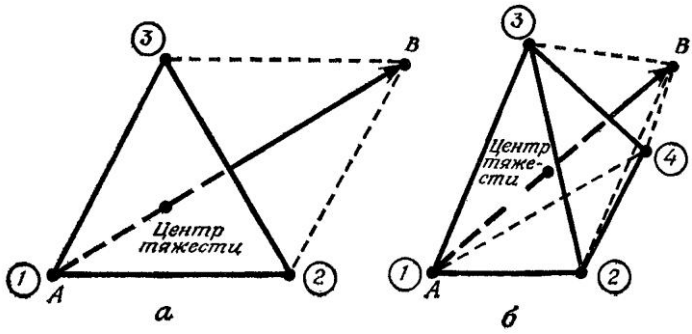

Рис. 11. Регулярные симплексы для двух (а) и трёх (б) переменных.
① обозначает вершину с наибольшим значением целевой функции *f(X)*; стрелка показывает направление наискорейшего уменьшения *f(X)*.

При поиске минимума целевой функции *f(X)* пробные векторы *X* могут быть выбраны в точках пространства проектных переменных E^n, находящихся в вершинах симплекса, как было первоначально предложено Спендли, Хекстом и Химсвортом. Из аналитической геометрии известно, что координаты вершин регулярного симплекса определяются следующей матрицей *D*, в которой столбцы представляют собой вершины, пронумерованные от 1 до *(n* + 1), а строчки - координаты; номер строки *i* принимает значения от 1 до *n:*

$$D = \begin{bmatrix} 0 & d_1 & d_2 & \dots & d_2 \\ 0 & d_2 & d_1 & \dots & d_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & d_2 & d_2 & \dots & d_1 \end{bmatrix}$$ - матрица $n \times (n+1)$,

где

$$d_1 = \frac{t}{n\sqrt{2}}\left(\sqrt{n+1} + n - 1\right);$$

$$d_2 = \frac{t}{n\sqrt{2}}\left(\sqrt{n+1} - 1\right);$$

t - расстояние между двумя вершинами. Например, для $n = 2$ и $t = 1$ треугольник, приведённый на рис. 11(а), имеет следующие координаты вершин.

Таблица 1. Координаты вершин плоского симплекса

№ вершины *(i)*	x_1^i	x_2^i
1	0	0
2	0.965	0.259
3	0.259	0.965

Целевая функция может быть вычислена в каждой из вершин симплекса; из вершины, где целевая функция максимальна (точка *A* на рис. 11), проводится проектирующая прямая через центр тяжести симплекса. Затем точка *A* исключается и строится новый симплекс, называемый *отражённым,* из оставшихся прежних точек и одной новой точки *B,* расположенной на проектирующей прямой на надлежащем расстоянии от центра тяжести. Продолжение этой процедуры, в которой каждый раз вычёркивается вершина, где целевая функция максимальна, а также использование правил уменьшения размера симплекса и предотвращения циклического движения в окрестности экстремума позволяют осуществить поиск, не использующий производные и в котором величина шага на любом этапе *k* фиксирована, а направление поиска можно изменять. На рис. 12 приведены последовательные симплексы, построенные в двумерном пространстве с «хорошей» целевой функцией.

Определённые практические трудности, встречающиеся при использовании регулярных симплексов, а именно отсутствие ускорения поиска и трудности при проведении поиска на искривлённых «оврагах» и «хребтах», привели к необходимости некоторых улучшений симплекс-метода. В этом разделе мы изложим метод Нелдера и Мида, в котором симплекс может изменять свою форму и, таким образом, уже не будет оставаться симплексом. Именно поэтому здесь использовано более подходящее название «деформируемый многогранник».

В методе Нелдера и Мида минимизируется функция *n* независимых переменных с использованием *n+1* вершин деформируемого многогранника в пространстве проектных переменных E^n.

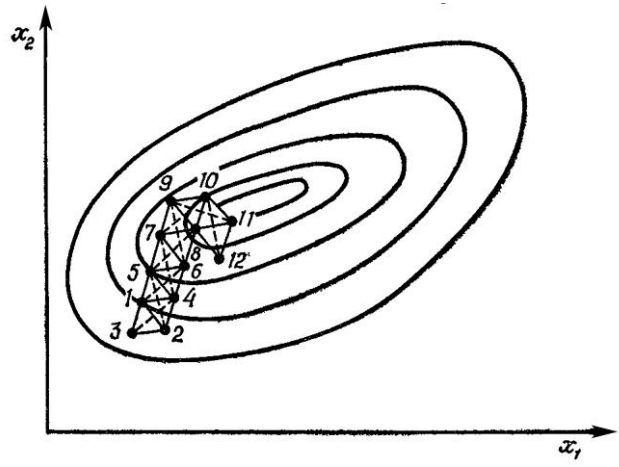

Рис. 12. Последовательность регулярных симплексов при минимизации функции $f(x_1, x_2)$.

Каждая вершина многогранника может быть идентифицирована вектором X. Вершина (точка) в которой значение $f(X)$ максимально, проектируется через центр тяжести многогранника. Улучшенные (более низкие) значения целевой функции находятся последовательной заменой точки с максимальным значением $f(X)$ на более «хорошие» точки, пока не будет найден минимум $f(X)$.

Более подробно этот алгоритм может быть описан следующим образом.

Пусть $X_i^{(k)} = \{ x_{i1}^{(k)} \ldots x_{ij}^{(k)}, \ldots x_{in}^{(k)} \}^{\mathrm{T}}$, $i = 1 \ldots (n + 1)$, является i-той вершиной (точкой) в пространстве проектных переменных на k-том этапе поиска, $k = 0, 1, \ldots$, и пусть значение целевой функции в i-той вершине $X_i^{(k)}$ равно $f(X_i^{(k)})$. Кроме того, отметим те векторы X многогранника, которые дают максимальное и минимальное значения $f(X)$. Определим

$$f(X_h^{(k)}) = \max \{ f(X_1^{(k)}), f(X_2^{(k)}) \ldots f(X_{n+1}^{(k)}) \},$$

где $X_h^{(k)} = X_i^{(k)}$, и

$f(X_l^{(k)}) = \min \{ f(X_1^{(k)}), f(X_2^{(k)}) \ldots f(X_{n+1}^{(k)}) \}$,

где $X_l^{(k)} = X_i^{(k)}$. Поскольку многогранник состоит из *(n+1)* вершин $X_1 \ldots X_{n+1}$, то пусть X_{n+2} будет центром тяжести всех вершин, с отсчётом от X_h.

Тогда координаты этого центра определяются формулой

$$x_{n+2,\,j}^{(k)} = \frac{1}{n}\left[\left(\sum_{i=1}^{n+1} x_{ij}^{(k)}\right) - x_{hj}^{(k)}\right], \qquad j = 1,2..n; \tag{25}$$

где индекс *j* обозначает координатное направление.

Начальный многогранник обычно выбирается в виде регулярного симплекса (но это не обязательно) с точкой *1* в качестве начала координат; можно начало координат поместить в центре тяжести. Процедура отыскания точки в пространстве проектных переменных, в которой *f(X)* имеет минимальное значение, состоит из следующих операций.

1. *Отражение* - проектирование $X_h^{(k)}$ через центр тяжести в соответствии с соотношением

$$X_{n+3}^{(k)} = X_{n+2}^{(k)} + \alpha \ (X_{n+2}^{(k)} - X_h^{(k)}) \ ; \tag{26}$$

где $\alpha > 0$ является коэффициентом отражения; $X_{n+2}^{(k)}$ - центр тяжести, вычисляемый по формуле (25); $X_h^{(k)}$ - вершина, в которой функция *f(X)* принимает наибольшее из $(n + 1)$ её значений на *k*-том этапе.

2. *Растяжение.* Эта операция заключается в следующем: если

$f(X_{n+3}^{(k)}) \le f(X_l^{(k)})$, то вектор $(X_{n+3}^{(k)} - X_{n+2}^{(k)})$ растягивается в соответствии с соотношением

$$X_{n+4}^{(k)} = X_{n+2}^{(k)} + \gamma \ (X_{n+3}^{(k)} - X_{n+2}^{(k)}) \ ; \tag{27}$$

где $\gamma > 1$ представляет собой коэффициент растяжения. Если $f(X_{n+4}^{(k)}) < f(X_l^{(k)})$, то $X_h^{(k)}$ заменяется на $X_{n+4}^{(k)}$ и процедура продолжается снова с операции 1 при $k = k + 1$. В противном случае $X_h^{(k)}$ заменяется на $X_{n+3}^{(k)}$ и также осуществляется переход к операции 1 при $k = k + 1$.

3. *Сжатие.* Если $f(X_{n+3}^{(k)}) > f(X_i^{(k)})$, для всех $i \neq h$, то вектор $(X_h^{(k)} - X_{n+2}^{(k)})$ сжимается в соответствии с формулой

$$X_{n+5}^{(k)} = X_{n+2}^{(k)} + \beta\ (X_h^{(k)} - X_{n+2}^{(k)})\ ; \qquad (28)$$

где $0 < \beta < 1$ представляет собой коэффициент сжатия. Затем $X_h^{(k)}$ заменяем на $X_{n+5}^{(k)}$ и возвращаемся к операции 1 для продолжения поиска на $(k + 1)$-м шаге.

4. *Редукция.* Если $f(X_{n+3}^{(k)}) > f(X_h^{(k)})$, то все векторы $(X_i^{(k)} - X_l^{(k)})$, $i = 1, 2 \ldots (n + 1)$; уменьшаются в 2 раза с отсчётом от $X_l^{(k)}$ в соответствии с формулой

$$X_i^{(k)} = X_l^{(k)} + 0.5\ (X_i^{(k)} - X_l^{(k)})\ , \quad i = 1, 2 \ldots (n + 1). \qquad (29)$$

Затем возвращаемся к операции 1 для продолжения поиска на $(k + 1)$ шаге.

Критерий окончания поиска, использованный Нелдером и Мидом, состоял в проверке условия

$$\left\{ \frac{1}{n+1} \sum_{i=1}^{n+1} \left[f(X_i^{(k)} - f(X_{n+2}^{(k)} \right]^2 \right\}^{1/2} \leq \varepsilon\ ; \qquad (30)$$

где ε - произвольное малое число, а $f(X_{n+2}^{(k)})$ - значение целевой функции в центре тяжести многогранника $X_{n+2}^{(k)}$.

На рис. 13 приведена блок-схема поиска методом деформируемого многогранника, а на рис. 14 показана последовательность поиска для функции Розенброка,

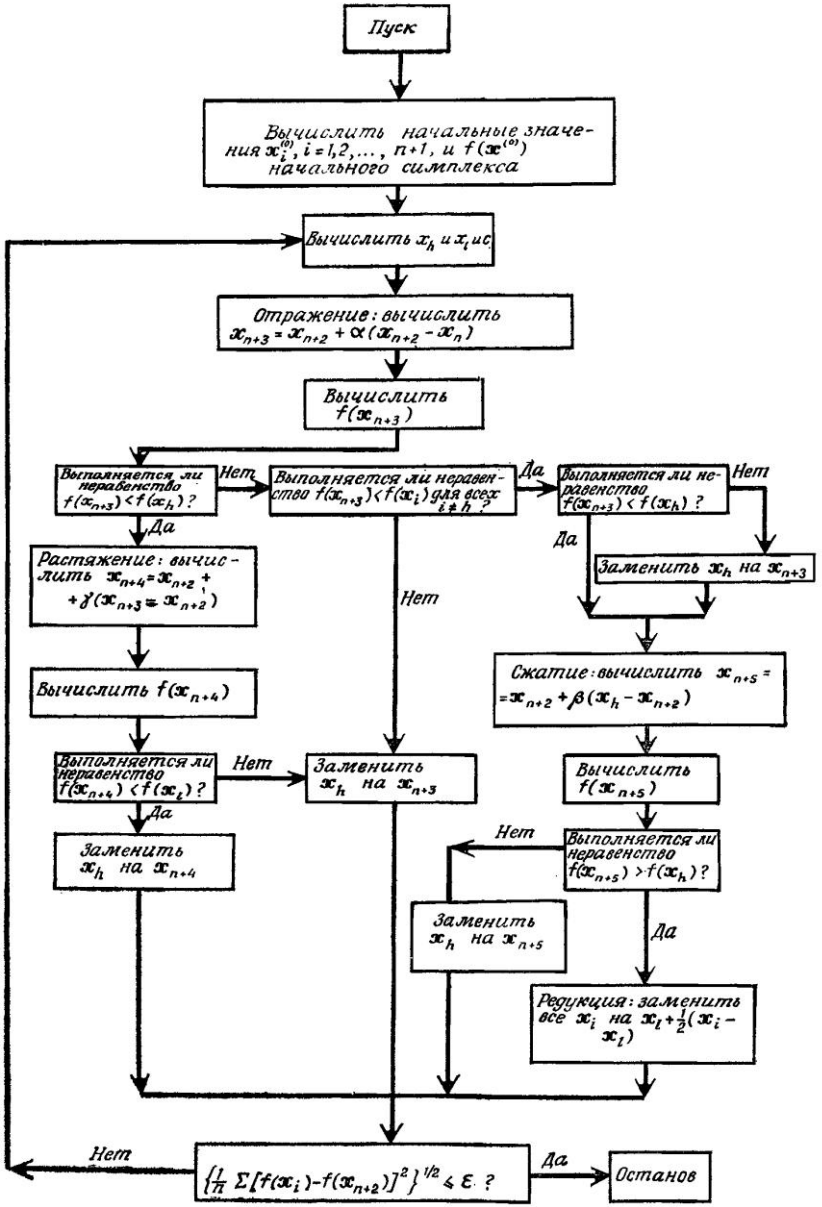

Рис. 13. Блок-схема метода Нелдера-Мида

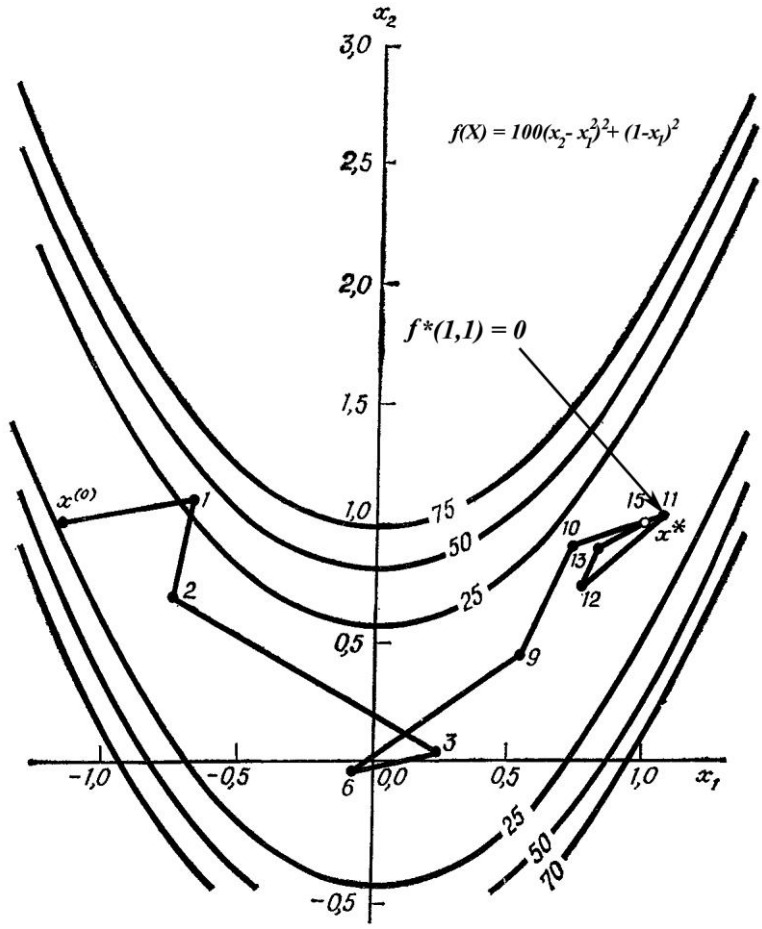

Рис. 14. Поиск минимума функции Розенброка

начиная из $X^{(0)} = \{-1.2 \quad 1.0\}^T$. Деформируемый многогранник в противоположность жёсткому симплексу адаптируется к топографии целевой функции, вытягиваясь вдоль длинных наклонных плоскостей, изменяя направление в изогнутых впадинах и сжимаясь в окрестности минимума. Коэффициент отражения ос используется для проектирования вершины с наибольшим значением *f(X)* через центр тяжести деформируемого многогранника. Коэффициент γ вводится для растяжения вектора поиска в случае, если отражение даёт вершину со значением *f(X),*

меньшим, чем наименьшее значение *f(X)*, полученное до отражения. Коэффициент сжатия β используется для уменьшения вектора поиска, если операция отражения не привела к вершине со значением *f(X)*, меньшим, чем второе по величине (после наибольшего) значение *f(X)*, полученное до отражения. Таким образом, с помощью операций растяжения или сжатия размеры и форма деформируемого многогранника масштабируются так, чтобы они удовлетворяли топологии решаемой задачи.

Естественно возникает вопрос, какие значения параметров α, β и γ должны быть выбраны. После того как форма и размер деформируемого многогранника подходящим образом выбран, его размеры должны поддерживаться неизменными, пока изменения в топологии задачи не потребуют применения многогранника другой формы. Это возможно реализовать только при $\alpha = 1$. Кроме того, Нелдер и Мид показали, что при решении задачи с $\alpha = 1$ требуется меньшее количество вычислений функции, чем при $\alpha < 1$. С другой стороны, значение α не должно быть много больше единицы, поскольку 1) деформируемый многогранник легче адаптируется к топологии задачи при меньших значениях α, особенно когда необходимо изменять направление поиска, столкнувшись с изогнутой впадиной, и 2) в области локального минимума размеры многогранника должны уменьшаться и большое α в этих условиях замедлит сходимость. Таким образом, значение $\alpha = 1$ выбирается как компромисс.

Чтобы выяснить, какое влияние на процедуру поиска имеет выбор β и γ, Нелдер и Мид, а также Павиани [2] провели решение нескольких тестовых задач, используя большое число различных комбинаций значений β и γ. В качестве удовлетворительных значений этих параметров при оптимизации без ограничений Нелдер и Мид рекомендовали $\alpha = 1$; $\beta = 0.5$ и $\gamma = 2.0$. Размеры и ориентация исходного многогранника в некоторой степени влияли на время решения, а значения α, β и γ оказывали значительно большее влияние. Павиани отмечает, что нельзя чётко решить вопрос относительно выбора β и γ и что влияние выбора β на эффективность поиска несколько более заметно, чем влияние γ. Павиани рекомендует следующие диапазоны значений для этих параметров: $0.4 \leq \beta \leq 0.6$; $2.8 \leq \gamma \leq 3.0$. При $0 < \beta < 0.4$ существует вероятность

того, что из-за уплощения многогранника будет иметь место преждевременное окончание процесса. При $\beta > 0.6$ может потребоваться избыточное число шагов и больше машинного времени для достижения окончательного решения.

Пример 9. Поиск методом деформируемого многогранника. Для иллюстрации метода Нелдера и Мида рассмотрим задачу минимизации функции $f(X) = 4 (x_1 - 5)^2 + (x_2 - 6)^2 = 4x_1^2 + x_2^2 - 40x_1 - 12x_2 + 136$; имеющей минимум в точке $X^* = \{5; 6\}^T$. Поскольку $f(x)$ зависит от двух переменных, то в начале поиска используется многоугольник с тремя вершинами. В этом примере в качестве начального многогранника взят треугольник с вершинами $X_1^{(0)} = \{8; 9\}^T$ $X_2^{(0)} = \{10; 11\}^T$ и $X_3^{(0)} = \{8; 11\}^T$, хотя можно было бы использовать любую другую конфигурацию из трёх точек.

На нулевом этапе поиска, $k = 0$, вычисляя значения функции, получаем $f(8, 9) = 45$, $f(10, 11) = 125$ и $f(8, 11) = 65$. Затем отражаем $X_2^{(0)} = \{10; 11\}^T$ через центр тяжести точек $X_1^{(0)}$ и $X_3^{(0)}$ по формуле (25), который обозначим через $X_4^{(0)}$:

$$x_{4,1}^{(0)} = 0.5 [(8 + 10 + 8) - 10] = 8;$$

$$x_{4,2}^{(0)} = 0.5 [(9 + 11 + 11) - 11] = 10;$$

с тем, чтобы получить $X_5^{(0)}$.

$$x_{5,1}^{(0)} = 8 + 1 (8 - 10) = 6;$$

$$x_{5,2}^{(0)} = 10 + 1 (10 - 11) = 9;$$

$$f(6, 9) = 13.$$

Поскольку $f(6,9) = 13 < f(8,9) = 45$, переходим к операции растяжения:

$$x_{6,1}^{(0)} = 8 + 2 (6 - 8) = 4;$$

$$x_{6,2}^{(0)} = 10 + 2 (9 - 10) = 8;$$

$$f(4,8) = 8.$$

Поскольку $f(4, 8) = 8 < f(8, 9) = 45$; то заменяем $X_2^{(0)}$ на $X_6^{(0)}$ и полагаем $X_6^{(0)} = X_2^{(1)}$ на следующем этапе поиска Наконец, поскольку

$$(1/3)\,[7^2 + 13^2 + 44^2]^{1/2} = 26.8 > 10^{-6};$$

то начинаем этап поиска $k = 1$. На рис. 15 показана траектория поиска на начальных этапах, а на рис. 16 изображена полная траектория поиска до его окончания. Для уменьшения целевой функции до $f(X) = 10^{-6}$ потребовалось 32 вычисления целевой функции.

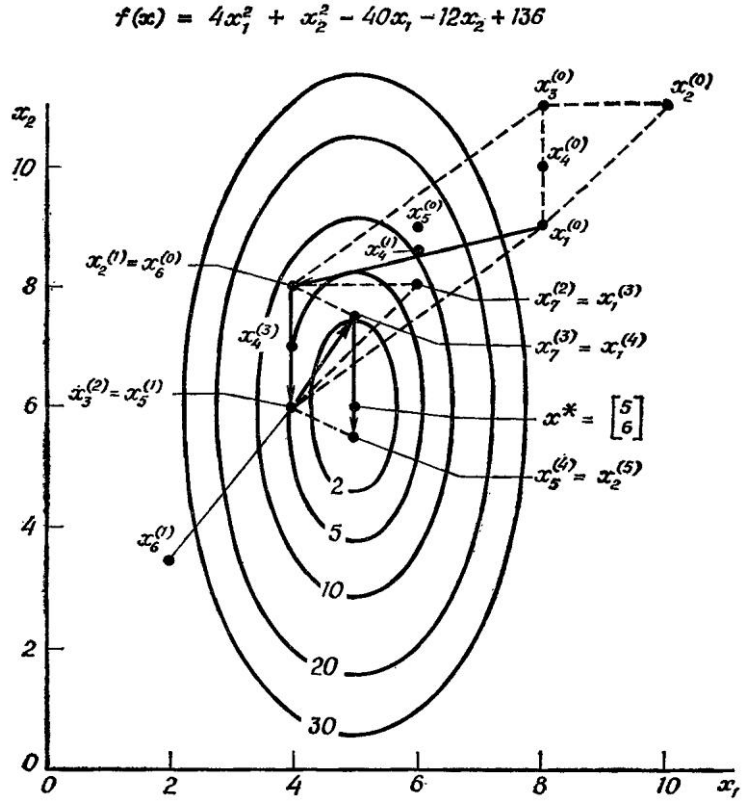

Рис. 15. Начало поиска методом Нелдера_Мида

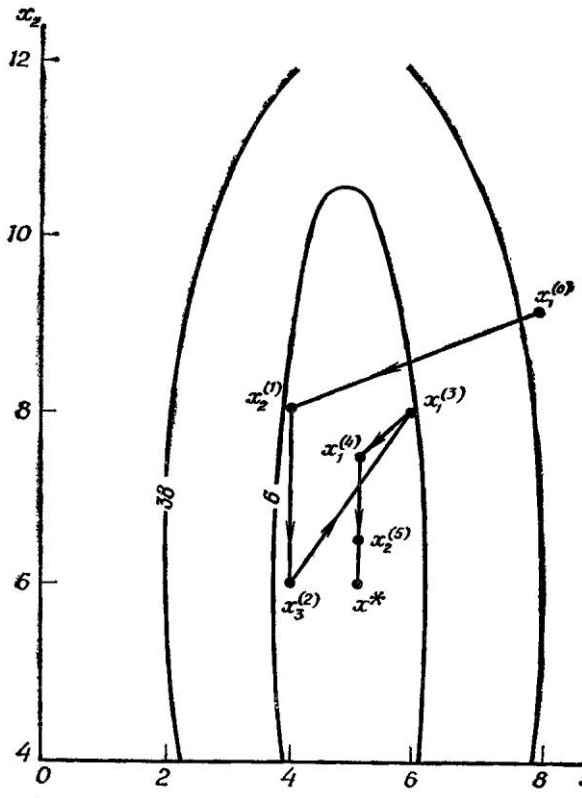

Рис. 16. Полная траектория поиска метода Нелдера-Мида.

Программа на языке C++, реализующая алгоритм Нелдера-Мида, как и ранее, представлена в виде подпрограммы, в качестве входных параметров она получает начальную точку поиска *X*, количество проектных переменных *nx* и точность поиска *eps*.

```
int Polyhedron(int nx, double x, double *opt, double eps)
//*******************************************************
//  Программа минимизации функции методом Нелдера-Мида
//*******************************************************
//     nx   - количество проектных переменных
//     x[]  - значения проектных переменных
//     opt  - значение функции в точке минимума
//     eps  - точность поиска
//-------------------------------------------------------
// Вызываемые подпрограммы
```

```
//     Fx() – вычисление целевой функции
//     start() – построение начального симплекса
//-------------------------------------------------------
{

extern double Fx(double);
extern void start(int, double, double*, double**);

double f[24], f_h, f_l, difer;
double x1[24][20], step, alfa, beta, gama, delta,
       xnx, sum2, sums,s;

int k1,k2,k3,k4, in,i,j, index, kount;
//-------------------------------------------------
// Начальные присвоения
k1 = nx + 1;
k2 = nx + 2;
k3 = nx + 3;
k4 = nx + 4;
//-------------------------------------------
       f[0] = Fx(x); // В начальной точке
//-------------------------------------------
       alfa  = 1.0; // Отражение
       beta  = 0.5; // Сжатие
       gama  = 2.0; // Растяжение
       delta = 0.5; // Редукция
       step  = 1.0; // Размер симплекса

       xnx = (double)nx;

// Построение симплекса в начальной точке
start(nx, step, x, x1);

       for(i = 0; i < k1; i++) {
           for(j = 0; j < nx; j++) x[j] = x1[i][j];
           f[i] = Fx(x);
       }
 M28:  f_h = f[0]; f_l = f[0];
       index = 0; kount = 0;
       for(i = 1; i < k1; i++) {
           if(f[i] > f_h) { f_h = f[i]; index = i; }
           if(f[i] < f_l) { f_l = f[i]; kount = i; }
       }
  // Вычисление центра симплекса
       for(j = 0; j < nx; j++) {
           sum2 = 0.0;
           for(i = 0; i < k1; i++) sum2 += x1[i][j];
           x1[k2-1][j] = (sum2 - x1[index][j]) / xnx;
```

```
      /* Отражение */
x1[k3-1][j] = (1.0 + alfa)*x1[k2-1][j] - alfa*x1[index][j];
             x[j] = x1[k3-1][j];
       }
       f[k3-1] = Fx(x);

// Выборка второго большого значения
       if (f[k3-1] >= f_l) {
           if (index == 0) sums = f[1]; else sums = f[0];
           for(i = 0; i < k1; i++) {
              if ((index == i) || (f[i] <= sums)) continue;
              sums = f[i];
           }
           if (f[k3-1] > sums) goto M13; else goto M14;
       }

       for(j = 0; j < nx; j++) { // Растяжение
           x1[k4-1][j] = (1.0 - gama) * x1[k2-1][j] + gama *
x1[k3-1][j];
           x[j] = x1[k4-1][j];
       }
       f[k4-1] = Fx(x);
       if (f[k4-1] < f_l) goto M16; else goto M14;

 M13:  if (f[k3-1] <= f_h)
       for(j = 0; j < nx; j++) x1[index][j] = x1[k3-1][j];
       for(j = 0; j < nx; j++) {
           x1[k4-1][j] = (1.0 - beta) * x1[k2-1][j] + beta *
x1[index][j];
           x[j] = x1[k4-1][j];
       }
       f[k4-1] = Fx(x);
       if (f_h > f[k4-1]) goto M16;

       for(j = 0; j < nx; j++)    // Редукция
           for(i = 0; i < k1; i++)
              x1[i][j] = delta * (x1[i][j] + x1[kount][j]);

       for(i = 0; i < k1; i++) {
          for(j = 0; j < nx; j++) x[j] = x1[i][j];
          f[i] = Fx(x);
       }

       goto M26;

 M16:  for(j = 0; j < nx; j++)
             x[j] = x1[index][j] = x1[k4-1][j];
       f[index] = Fx(x);
```

```
       goto M26;

 M14:   for(j = 0; j < nx; j++)
       x[j] = x1[index][j] = x1[k3-1][j];
       f[index] = Fx(x);

 M26:  for(j = 0; j < nx; j++) x[j] = x1[k2-1][j];
       f[k2-1] = Fx(x);

       difer = 0.0;
       for(i = 0; i < k1; i++) {
           s = f[i] - f[k2-1];
           difer += (s * s);
       }
       difer = sqrt(difer) / xnx;

//  print(x1[kount],f_l);

       if (difer >= eps) goto M28;

       *opt = f_l;
       for(j = 0; j < nx; j++) x[j] = x1[kount][j];

       return(0);
}
//=================================================
void start(nx, step, x, x1)
            int nx;
            double step, x[], x1[][20];
{
       double vn, step1, step2;
       int i,j;

       vn = (double)nx;
       step1 = (step/(1.414*vn))*(sqrt(vn+1.0) + vn - 1.0);
       step2 = (step/(1.414*vn))*(sqrt(vn+1.0) - 1.0);
       for(j = 0; j < nx; j++) x1[0][j] = 0.0;
       for(i = 1; i < nx+1; i++) {
           for(j = 0; j < nx; j++) x1[i][j] = step2;
           x1[i][i-1] = step1;
           }
       for(i = 0; i < nx+1; i++)
           for(j = 0; j < nx; j++) x1[i][j] += x[j];
       return;
}
//=================================================
```

5. Заключение

Мы рассмотрели ничтожно малое количество оптимизирующих алгоритмов. Этими алгоритмами мы показали стратегию методов поиска экстремума и теперь, на базе полученных знаний читатель может компетентно анализировать достоинства и недостатки других, существующих, а также новых методов оптимизации и уверенно выбирать подходящий алгоритм для конкретной организационно-технической задачи.

Камнем преткновения любых поисковых методов оптимизации является критерий окончания поиска. Таких критериев всего два: малое изменение функции и/или малое изменение проектных переменных. Рис. 17 иллюстрирует, почему необходимо учитывать ОБА этих критерия одновременно. Однако на практике обычно выбирается какой-либо один. Имейте это в виду.

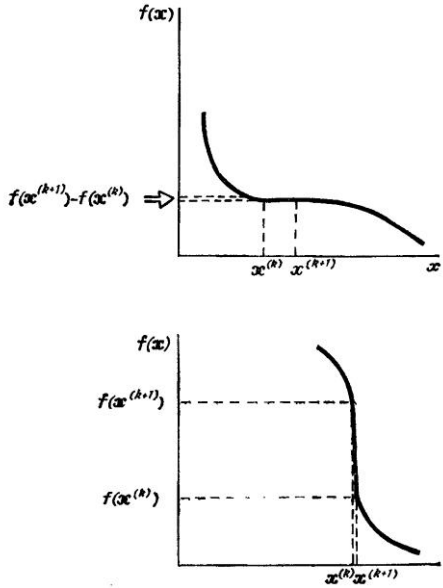

Рис. 17. Возможные ситуации при окончании поиска.

В пособии [1] рассмотрены постановки задач оптимизации и приёмы преобразования задачи с ограничениями к задаче без ограничений с помощью штрафных функций. При этом топография целевой функции безнадёжно портится и возникает проблема выбора даже не лучшего, а достаточно надёжного метода, который просто найдёт экстремум среди «оврагов» и «хребтов». Рассмотренные алгоритмы выбраны для включения в данное пособие именно с этой точки зрения. Они прошли апробацию на реальных технических задачах и со всеми испытаниями справились блестяще.

Литература

1. Данилин А.И. Основы теории оптимизации (постановки задач). Учебное пособие. –Самара: Изд-во Самарского государственного аэрокосмического ун-та, 2011. -57с.

2. Химмельблау Д. Прикладное нелинейное программирование. М: Мир, 1975. -534с.

3. Уайлд Д. Оптимальное проектирование. -М: Мир, 1981. -272с.

4. Реклейтис Г., Рейвиндран А., Рэгсделл К. Оптимизация в технике. В 2-х книгах. -М: Мир, 1986. -Кн.1 -350с.; -Кн.2 -320с.

Оглавление