ГЛАВА 12
ПАРАЛЛЕЛЬНЫЕ МЕТОДЫ МНОГОЭКСТРЕМАЛЬНОЙ ОПТИМИЗАЦИИ

12.1. Введение

Во всех областях своей целенаправленной деятельности перед человеком возникает проблема выбора наилучшего решения из множества всех возможных. Примерами могут служить экономика (управление экономическими объектами), техника (выбор оптимальной конструкции). К числу наиболее распространенных моделей рационального выбора относятся математические задачи оптимизации (максимизации или минимизации) некоторого функционала при ограничениях типа неравенств. При этом выполнение ограничений для некоторого вектора параметров, определяющего решение, интерпретируется как допустимость этого решения, т. е. как возможность его реализации при имеющихся ресурсах. Теорию и методы отыскания минимумов функций многих переменных при наличии дополнительных ограничений на эти переменные обычно рассматривают как отдельную дисциплину – математическое программирование (см., например, [6,14,52,60]).

Следует отметить, что если в главах 6–11 рассматриваемые задачи имели, скорее всего, учебный характер и не требовали для своего изучения каких-либо значительных усилий, то исследуемая в данной главе проблема глобальной оптимизации предполагает более глубокое погружение в тематику, и ее успешное освоение требует значительно больше времени. Изложение проблематики глобального поиска осуществляется в достаточно кратком виде; для получения более полной информации следует обратиться к основной литературе в данной области – см., например, [13,28,97].

12.2. Постановка задачи

В общем виде задачу математического программирования можно сформулировать следующим образом. Пусть $\varphi(y)$, $g_j(y) \le 0$, $1 \le j \le m$, есть действительные функции, определенные на множестве D N-мерного евклидова пространства R^N, и пусть точка y^* удовлетворяет условию

$$\varphi(y^*)=\min\{\varphi(y):\ y\in D,\ g_j(y) \le 0,\ 1 \le j \le m\}. \qquad (0.1)$$

Точка y^* из (0.1) обычно называется *глобально-оптимальной точкой* или *глобально-оптимальным решением*. При этом функцию $\varphi(y)$ называют

функцией цели, или *целевой функцией,* а функции $g_j(y) \le 0,\ 1 \le j \le m,$ – *ограничениями* задачи.

Область D называют *областью поиска* и обычно описывают как некоторый гиперинтервал из *N*-мерного евклидова пространства

$$D=\{y\in R^N:\ a_i \le y_i \le b_i,\ 1 \le i \le n\},$$

где $a,b\in R^N$ есть заданные векторы. Точки из области поиска, удовлетворяющие всем ограничениям, называются *допустимыми точками* или *допустимыми решениями*. Множество

$$Q=\{y:\ y\in D,\ g_j(y) \le 0,\ 1 \le\ j\le m\} \qquad (0.2)$$

всех таких точек называют *допустимой областью*.

Важный в прикладном отношении подкласс задач вида (0.1) характеризуется тем, что все функционалы, входящие в определение задачи, заданы некоторыми (программно реализуемыми) алгоритмами вычисления значений $\varphi(y),\ g_j(y)\ ,\ 1 \le j \le m,$ в точках области поиска D. При этом *решение задачи* (0.1) сводится к построению оценки $y_* \in Q$, отвечающей некоторому понятию близости к точке y^* (например, чтобы $\left\|y^* - y_*\right\| \le \varepsilon$ или $\left|\varphi(y^*) - \varphi(y_*)\right| \le \varepsilon$, где $\varepsilon > 0$ есть заданная точность) на основе некоторого числа k значений функционалов задачи, вычисленных в точках области D.

В задачах многоэкстремальной оптимизации возможность достоверной оценки глобального оптимума принципиально основана на наличии *априорной информации* о функции, позволяющей связать возможные значения минимизируемой функции с известными значениями в точках осуществленных поисковых итераций. Весьма часто такая априорная информация о задаче (0.1) представляется в виде предположения, что целевая функция φ (в дальнейшем обозначаемая также g_{m+1}) и левые части ограничений $g_j, 1 \le j \le m,$ удовлетворяют *условию Липшица* с соответствующими константами $L_j, 1 \le j \le m+1$, а именно

$$\left| g_j(y_1) - g_j(y_2) \right| \le L_j||y_1 - y_2||,\ 1 \le j \le m+1,\ y_1, y_2 \in D. \qquad (0.3)$$

В общем случае все эти функции могут быть *многоэкстремальными*.

Пример 0.1. Рассмотрим задачу минимизации функции

$$\varphi(y_1, y_2) = -1.5y_1^2 \exp\{1 - y_1^2 - 20.25(y_1 - y_2)^2\} -$$
$$-[0.5(y_1 - 1)(y_2 - 1)]^4 \exp\{2 - [0.5(y_1 - 1)]^4 - (y_2 - 1)^4\}$$

в области поиска $0 \le y_1 \le 4$, $-1 \le y_2 \le 3$, при ограничениях

$$g_1(y_1, y_2) = 0.01[(y_1 - 2.2)^2 + (y_2 - 1.2)^2 - 2.25] \le 0$$
$$g_2(y_1, y_2) = 100[1 - (y_1 - 2)^2 / 1.44 - (0.5 y_2)^2] \le 0$$
$$g_3(y_1, y_2) = 10[y_2 - 1.5 - 1.5\sin(6.283(y_1 - 1.75))] \le 0$$

Допустимые по первому ограничению точки образуют круг с границей $g_1(y_1, y_2) = 0$. Допустимые по второму ограничению точки находятся во внешности эллипса $g_2(y_1, y_2) = 0$. Точки, допустимые по третьему ограничению, находятся ниже синусоиды $g_3(y_1, y_2) = 0$. Следовательно, допустимая область является неодносвязной и состоит из трех невыпуклых подобластей (на рис. 0.1 допустимая область выделена цветом). Глобальный минимум $\varphi(y_1^*, y_2^*) = -1.489$ достигается в точке (y_1^*, y_2^*)=(0.942, 0.944).

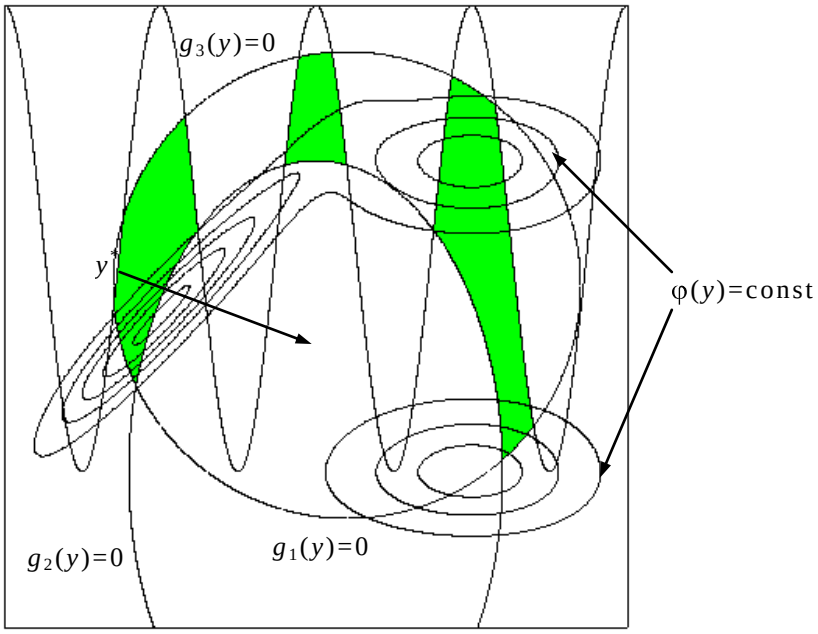

Рис. 0.1. Задача глобальной оптимизации

12.3. Методы решения задач многоэкстремальной оптимизации

Важно отметить, что в задачах многоэкстремальной оптимизации глобальный экстремум является интегральной характеристикой задачи, его определение, вообще говоря, связано с построением покрытия области по-

иска точками вычислений функции. Поэтому на сложность решения задач рассматриваемого класса решающее влияние оказывает размерность: они подвержены «проклятию размерности», состоящему в экспоненциальном росте вычислительных затрат при увеличении размерности. Использование простейших способов решения (таких, как перебор по равномерной сетке) является чрезмерно затратным. Требуется применение более экономных методов, которые порождают в области поиска существенно неравномерную сетку, более плотную в окрестности решения задачи. Рассмотрению вопросов, возникающих при построении подобного рода методов, посвящены, например, работы [28,52,66,74,82-83,97,101].

Фундаментальные результаты в этом направлении были получены Нижегородской школой глобальной оптимизации (коллектив исследователей под руководством Р.Г. Стронгина, Нижегородский государственный университет): разработан информационно-статистический подход к построению алгоритмов многоэкстремальной оптимизации ([28,97]). Общая информация о подходе содержится в указанных работах, здесь же кратко поясним следующее. Минимизируемая функция рассматривается как реализация некоторого случайного процесса, и решающие правила алгоритма конструируются таким образом, что очередная итерация проводится в точке глобального минимума математического ожидания значений функции.

В рамках информационно-статистического подхода был разработан эффективный способ учета ограничений в задачах условной оптимизации, получивший название *индексного метода* (см. [80,106]). Его характерной чертой является раздельный учет каждого из ограничений задачи, а штрафные функции не используются. В соответствии с правилами индексного метода каждая итерация, называемая *испытанием* в соответствующей точке области поиска, включает последовательную проверку выполнимости ограничений задачи в этой точке, а обнаружение первого нарушенного ограничения прерывает испытание и инициирует переход к точке следующей итерации.

Следует отметить, что регулярные поисковые методы решения многомерных оптимизационных задач (т. е. методы, отличные от стохастических процедур типа Монте-Карло) как правило (явно или неявно) сводят многомерную задачу к системе одномерных подзадач. Локальные методы, например, осуществляют такое сведение путем построения последовательности направлений спуска, вдоль которых осуществляется одномерная минимизация, что порождает траекторию, ведущую из начальной точки в окрестность решения (см., например, [51,60,66]). В многоэкстремальных задачах схема локального спуска, вообще говоря, не приводит к решению. Редукция размерности при решении таких задач может основываться на не-

которых фундаментальных свойствах многомерных функций и многомерных пространств.

Один из подходов сводит решение многомерной задачи к решению серии вложенных одномерных подзадач (т. н. *многошаговая схема*). Описание параллельного алгоритма решения многомерных задач многоэкстремальной оптимизации, основанного на многошаговой схеме редукции размерности, приводится, например, в работе [91].

Другой способ редукции размерности, о котором речь пойдет ниже, использует отображение многомерной области поиска на одномерный интервал с помощью *кривых Пеано*. Итальянским математиком Пеано было установлено (см. [79]), что может быть построено однозначное непрерывное отображение $y(x)=(y_1(x),y_2(x))$ отрезка [0,1] на единичный квадрат:

$$\{y\in R^2:\ -2^{-1}\le y_1,y_2\le 2^{-1}\}=\{y(x):\ 0\le x\le 1\}.$$

Этот результат был обобщен на многомерный случай, т. е. было доказано существование кривых $y(x)$, заданных непрерывными координатными функциями $y_i(x),\ x\in[0,1],\ 1\le i\le N$, однозначно отображающих отрезок $[0,1]$ на N-мерный гиперкуб D, т. е.

$$D=\{y\in R^N:\ a_i\le y_i\le b_i,\ 1\le i\le N\}=\{y(x):\ 0\le x\le 1\}.$$

Такие кривые, называемые также *развертками Пеано*, позволяют свести многомерную задачу условной минимизации в области D к одномерной задаче условной минимизации на отрезке $[0,1]$

$$\varphi(y(x^*))=\min\{\varphi(y(x)):\ x\in[0,1],g_j(y(x))\le 0,1\le j\le$$ (0.4)

Таким образом, рассмотренная схема сведения многомерной многоэкстремальной задачи условной оптимизации к эквивалентной ей одномерной задаче позволяет применить для ее решения эффективные одномерные методы поиска. Для более простого освоения материала изложим сначала индексную схему учета ограничений для одномерных задач.

12.4. Решение одномерных задач

12.4.1. Индексный метода учета ограничений

Рассмотрим одномерную задачу условной глобальной оптимизации вида

$$\varphi(x^*)=\min\{\varphi(x):\ x\in[a,b],\ g_j(x)\le 0,\ 1\le j\le m\}$$ (0.5)

в предположении, что целевая функция φ (в дальнейшем обозначаемая также g_{m+1}, т. е. $g_{m+1}(x)\equiv\varphi(x)$) и левые части ограничений $g_j, 1\le j\le m$, являются определенными на отрезке $[a,b]$ липшицевыми функциями с соответствующими константами $L_j, 1\le j\le m+1$. В общем случае все эти функции могут быть многоэкстремальными.

Пример 0.2. Рассмотрим задачу вида (0.5) при $x\in[0.6,2.2]$, $m=2$,

$$\varphi(x)=\cos(18x-3)\sin(10x-7)+1.5,$$
$$g_1(x)=\exp(-x/2)\sin(6x-1.5),$$
$$g_2(x)=|x|\sin(2\pi x-0.5).$$

Точное решение задачи $x^*=2.0795$, $\varphi(x^*)=0.565$. Графики функций $g_1(x)$, $g_2(x)$, $\varphi(x)$ представлены на рис. 0.2.

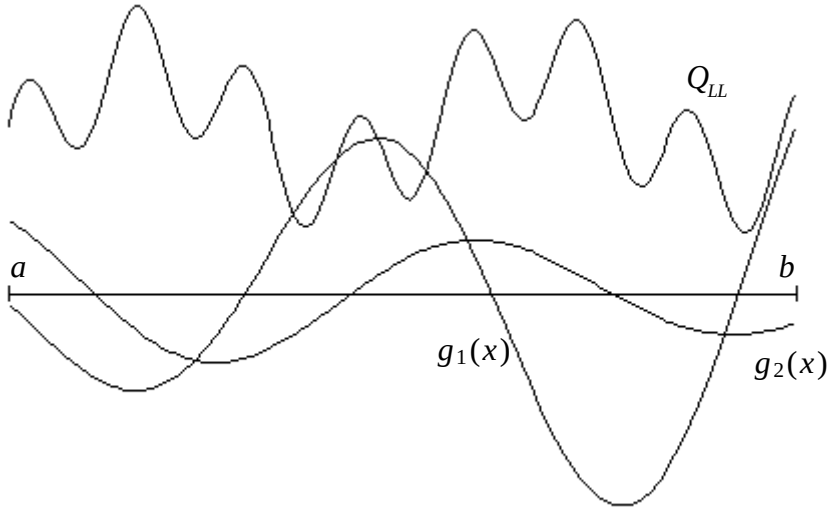

Рис. 0.2. Одномерная задача глобальной оптимизации

Задача (0.5) может быть рассмотрена в постановке, когда каждая функция g_j, $1\le j\le m+1$, определена и вычислима лишь в соответствующей подобласти $Q_j\subset[a,b]$, где

$$Q_1=[a,b],\ Q_{j+1}=\{x\in Q_j: g_j(x)\le 0\},\ 1\le j\le m. \qquad (0.6)$$

Так, например, в задачах оптимального проектирования некоторые характеристики технической системы оказываются неопределенными, если не выполнены представленные частью ограничений задачи (0.5) условия функционирования системы.

С учетом условий (0.6), исходная задача (0.5) может быть представлена в виде

$$\varphi(x^*)=\min\{g_{m+1}(x):\ x\in Q_{m+1}\}. \tag{0.7}$$

Для целей дальнейшего изложения введем классификацию точек x из области поиска $[a,b]$ с помощью *индекса* $\nu=\nu(x)$, где $\nu-1$ есть число ограничений, которые выполняются в этой точке. Указанный индекс ν определяется условиями

$$g_j(x)\le 0,\ 1\le j\le \nu-1,\ g_\nu(x)>0, \tag{0.8}$$

где последнее неравенство несущественно, если $\nu=m+1$, и удовлетворяет неравенствам

$$1\le \nu=\nu(x)\le m+1.$$

На рис. 0.3 приведена задача из примера 0.2 в предположении частичной вычислимости функций. Изображены дуги функций ограничений $g_1(x)$, $g_2(x)$ и целевой функции $\varphi(x)$, определенные на соответствующих множествах Q_j из (0.6). Изображена также точка x^1 с индексом $\nu(x^1)=1$, точка x^2 с индексом $\nu(x^2)=2$, и точка x^3 с индексом $\nu(x^3)=3$.

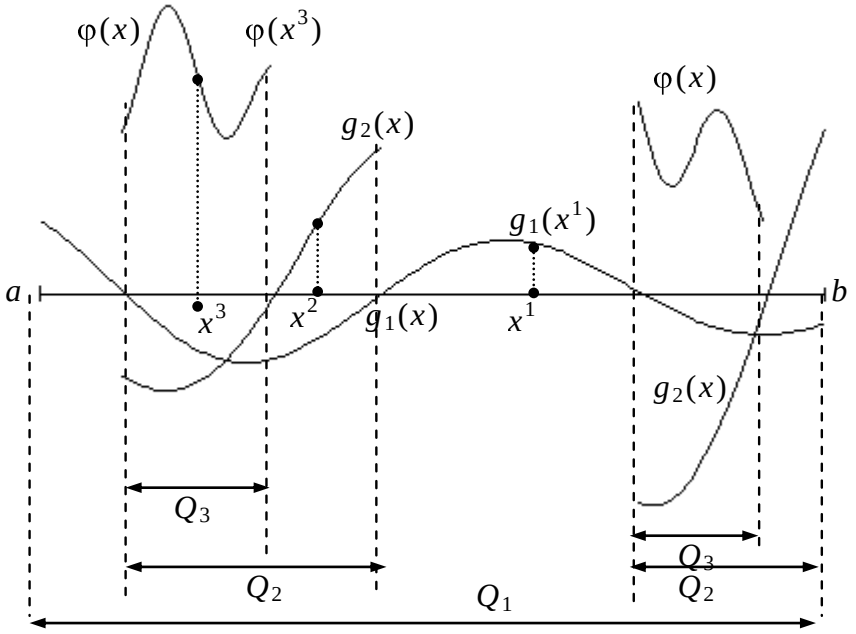

Рис. 0.3. Задача с частично вычислимыми функциями

Данная классификация порождает функцию

$$f(x)=g_{\nu}(x),\ \nu=\nu(x), \tag{0.9}$$

определенную и вычислимую всюду в $[a,b]$. Ее значение в точке x есть либо значение левой части ограничения, нарушенного в этой точке (случай, когда $\nu\le m$), либо значение минимизируемой функции (случай, когда $\nu=m+1$). Поэтому определение значения $f(x),\ x\in[a,b]$, сводится к последовательному вычислению величин $g_j(x),\ 1\le j\le\nu=\nu(x)$, т. е. последующее значение $g_{j+1}(x)$ вычисляется лишь в том случае, когда $g_j(x)\le 0$. Процесс вычислений завершается либо в результате установления неравенства $g_j(x)>0$, либо в результате достижения значения $\nu(x)=m+1$.

```
// Алгоритм определения индекса
  double value; // значение функции
  int index;    // индекс точки
  for( int i = 0; i< m+1; i++ ) {
    value=g[i](x);
    if( (i == m+1) || (value>0)){
        index = i;
        break;
    }
  }
```

Описанная процедура, названная *испытанием* в точке x, автоматически приводит к определению индекса ν этой точки. Пара значений

$$z=f(x)=g_{\nu}(x),\ \nu=\nu(x), \tag{0.10}$$

порожденная испытанием в точке $x\in[a,b]$, называется *результатом испытания*.

На рис. 0.4 приведен график функции $f(x)$ из (0.9), который образован дугами функций ограничений $g_1(x),\ g_2(x)$ и целевой функции $\varphi(x)$ для задачи из примера 0.2.

Поскольку в общем случае задача (0.7) может не иметь решения (т. е. допустимая область Q_{m+1} может оказаться пустой в силу несовместимости ограничений), с ней связывается некоторая вспомогательная задача, всегда имеющая решение. Так как условия (0.8) эквивалентны условиям

$$x\in Q_{\nu},\ x\notin Q_{\nu+1},$$

то вспомогательная задача, всегда имеющая решение, может быть записана в виде

$$g_M^*=g_M(x_M^*)=\min\{g_M(x):\ x\in Q_M\}, \tag{0.11}$$

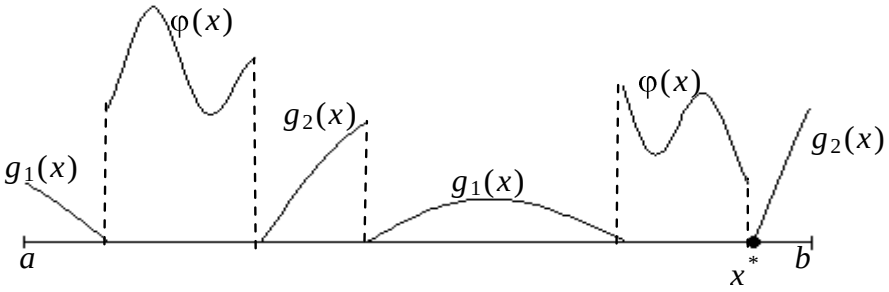

Рис. 0.4. «Индексная» функция

где M есть максимально возможное значение индекса, т. е.

$$1 \le M = \max\{\nu(x):\ x \in [a,b]\} \le m+1. \tag{0.12}$$

Поскольку множество Q_M всегда непусто, задача (0.11) всегда имеет решение. При $M = m+1$ решение $x^* = x^*_{m+1}$ является также решением исходной задачи (0.5). При $M<m+1$ необходимо выполняющееся неравенство $g^*_M < 0$ может использоваться как индикатор несовместимости ограничений.

Основная идея индексного подхода состоит в редукции условной задачи (0.11) к безусловной задаче

$$\psi(x^*) = \min\{\psi(x):\ x \in [a,b]\},$$

где

$$\psi(x) = \begin{cases} g_\nu(x)/L_\nu, & \nu < M, \\ (g_M - g^*_M)/L_M, & \nu = M. \end{cases} \tag{0.13}$$

Как результат, дуги функции $\psi(x)$ будут липшицевыми с константой $L=1$ на каждом множестве Q_ν, $1 \le \nu \le M$ (на рис. 0.5 приведен график функции $\psi(x)$ для задачи из примера 0.2). Эта новая функция будет иметь разрывы первого рода на граничных точках множеств Q_ν из (0.6). Но, тем не менее, можно оценить точку глобального оптимума x^*_M, используя результаты (0.10) k испытаний в точках $x^1, \ldots, x^k$ из $[a,b]$ ([97]).

В самом деле, из условия Липшица следует, что

$$x^*_M \in \{x \in [a,b]:\ |x - x^i| \ge \psi(x^i),\ 1 \le i \le k\}. \tag{0.14}$$

Так, на рис. 0.5 изображен случай $k=4$. Объединение отрезков, выделенных на рисунке жирной линией, не содержит оптимальную точку.

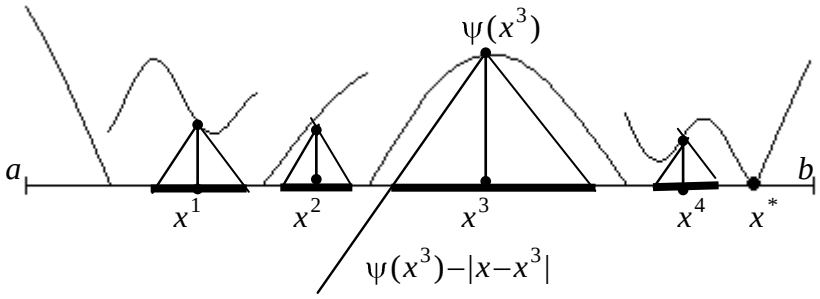

Рис. 0.5. Оценка оптимальной точки

Заметим, что максимальный индекс M, значения констант Липшица L_ν, $1\le\nu\le M$, и величина g_M^* являются неизвестными. Однако эти проблемы можно преодолеть, используя вместо этих величин их адаптивные оценки, получаемые в процессе решения задачи на основании результатов испытаний.

После того, как введены все требуемые понятия, перейдем к изложению индексного алгоритма.

12.4.2. Схема алгоритма

Первое испытание осуществляется в произвольной внутренней точке $x^1\in(a,b)$. Выбор точки x^{k+1}, $k\ge1$, любого последующего испытания определяется следующими правилами.

Правило 1. Перенумеровать точки $x^1,\ldots,x^k$ предшествующих испытаний нижними индексами в порядке увеличения значений координаты, т. е.

$$a=x_0<x_1<\ldots<x_i<\ldots<x_k<x_{k+1}=b, \tag{0.15}$$

и сопоставить им значения $z_i=g_\nu(x_i)$, $\nu=\nu(x_i)$, $1\le i\le k$, из (0.10), вычисленные в этих точках; точки $x_0=a$ и $x_{k+1}=b$ введены дополнительно (значения z_0 и z_{k+1} не определены) для удобства последующих обозначений.

Правило 2. Провести классификацию номеров $i,1\le i\le k$, точек из ряда (0.15) по числу ограничений задачи, выполняющихся в этих точках, путем построения множеств

$$I_\nu=\{i:\ 1\le i\le k,\ \nu=\nu(x_i)\},\ 1\le\nu\le m+1.$$

содержащих номера всех точек $x_i,1\le i\le k$, имеющих индексы, равные одному и тому же значению ν. Граничные точки $x_0=a$ и $x_{k+1}=b$ интерпре-

тируются как имеющие нулевые индексы, и им сопоставляется дополнительное множество $I_0=\{0,k+1\}$.

Определить максимальное значение индекса

$$V=\max\{\nu=\nu(x_i),1\le i\le k\}. \tag{0.16}$$

Правило 3. Вычислить текущие нижние границы

$$\mu_\nu=\max\left\{\frac{|z_i-z_j|}{x_i-x_j},\quad i,j\in I_\nu,\ \ i>j\right\} \tag{0.17}$$

для неизвестных констант Липшица L_ν функций $g_\nu, 1\le\nu\le m+1$. Если множество I_ν содержит менее двух элементов или если μ_ν из (0.17) оказывается равным нулю, то принять $\mu_\nu=1$. Из (0.17) следует, что оценки μ_ν являются неубывающими, начиная с момента, когда (0.17) порождает первое положительное значение μ_ν.

Правило 4. Для всех непустых множеств I_ν, $1\le\nu\le m+1$, вычислить оценки

$$z_\nu^*=\begin{cases}0, & \nu<V,\\ \min\{g_\nu(x_i):i\in I_\nu\}, & \nu=V.\end{cases} \tag{0.18}$$

Правило 5. Для каждого интервала (x_{i-1},x_i), $1\le i\le k+1$, вычислить *характеристику* $R(i)$, где

$$R(i)=\Delta_i+\frac{z_i-z_{i-1}^{\ 2}}{r_\nu^2\mu_\nu^2\Delta_i}-2\frac{z_i+z_{i-1}-2z_\nu^*}{r_\nu\mu_\nu},\qquad \nu=\nu(x_{i-1})=\nu(x_i).$$

$$R(i)=2\Delta_i-4\frac{(z_i-z_\nu^*)}{r_\nu\mu_\nu},\quad \nu=\nu(x_i)>\nu(x_{i-1}), \tag{0.19}$$

$$R(i)=2\Delta_i-4\frac{(z_{i-1}-z_\nu^*)}{r_\nu\mu_\nu},\quad \nu=\nu(x_{i-1})>\nu(x_i),$$

$$\Delta_i=x_i-x_{i-1}.$$

Величины $r_\nu>1$, $1\le\nu\le m+1$, являются параметрами алгоритма. Подходящий выбор значений r_ν позволяет использовать произведение $r_\nu\mu_\nu$ как оценку константы Липшица L_ν, $1\le\nu\le m+1$.

Правило 7. Определить интервал (x_{t-1},x_t), которому соответствует максимальная характеристика

$$R(t)=\max\{R(i):\ 1\le i\le k+1\}. \tag{0.20}$$

Правило 8. Провести очередное испытание в серединной точке интервала (x_{t-1},x_t), если индексы его концевых точек не совпадают, т. е.

$$x^{k+1}=\frac{x_t+x_{t-1}}{2},\ \nu(x_{t-1})\ne\nu(x_t).$$

В противном случае провести испытание в точке

$$x^{k+1}=\frac{x_t+x_{t-1}}{2}-\frac{z_t-z_{t-1}}{2r_\nu\mu_\nu},\ \nu=\nu(x_{t-1})=\nu(x_t). \tag{0.21}$$

Описанные правила можно дополнить условием остановки, прекращающим испытания, если

$$x_t-x_{t-1}\le\varepsilon, \tag{0.22}$$

где t из (0.20) и $\varepsilon>0$ есть заданная точность.

Условия сходимости алгоритма. Если рассмотренный индексный алгоритм применяется для решения задачи (0.5) и при этом выполняются условия:

1. каждая функция g_j, $1\le j\le m+1$, удовлетворяет на отрезке $[a,b]$ условию Липшица с константой L_j, т. е.

$$|g_j(x_1)-g_j(x_2)|\le L_j|x_1-x_2|,\ 1\le j\le m+1,\ x_1,x_2\in[a,b];$$

2. для величин μ_ν из (0.17), начиная с некоторого шага, справедливы неравенства

$$r_\nu\mu_\nu>2L_\nu,\ 1\le\nu\le m+1.$$

то множество предельных точек последовательности $\{x^k\}$, порождаемой индексным алгоритмом, совпадет с множеством решений задачи (0.5) при $\varepsilon=0$ в условии остановки (0.22), причем индекс каждой предельной точки равен M.

12.4.3. Программная реализация

Представим возможный вариант программной реализации индексного метода для решения одномерных задач условной оптимизации. В отличие от программ предыдущих разделов разработка выполнена на C++.

Заголовочный файл Method.h содержит объявление класса CMethod, реализующего индексный метод, а также объявление всех требуемых для работы структур и типов данных.

```
// Программ 12.1
// Одномерный индексный метод глобального поиска
```

```
#include <vector>
#include <set>
#define _USE_MATH_DEFINES
#include <math.h>

#define MaxFuncs 10
typedef double (*pFunc)(double);

// Точка испытания
struct CTrial {
  double Value;
  int index;
  double x;
};

inline bool operator<(const CTrial& t1,
  const CTrial& t2){return (t1.x<t2.x);}

typedef std::set<CTrial>::iterator pTrial;

// Индексный метод
class CMethod {
public:
  double a;   // Левая граница интервала поиска
  double b;   // Правая граница интервала поиска
  double eps; // Точность поиска
  double r[MaxFuncs];  // Параметр надежности
  int NumFuncs;        // Число функций задачи
  CMethod(){}   // Контруктор по умолчанию
  ~CMethod(){}  // Деструктор по умолчанию
  void Run();   // Запуск метода
  pFunc Funcs[MaxFuncs];  // Функции задачи
  CTrial BestTrial;       // Лучшее испытание
private:
  // Параметр Z алгоритма
  double Z[MaxFuncs];
  // Относительные первые разности
  double M[MaxFuncs];
  // Указатели на точки с фиксированным индексом
  std::vector<pTrial> I[MaxFuncs];
  // Точки испытаний
  std::set<CTrial> Trials;
  // Максимальный индекс
```

```
  int MaxIndex;
  // Инициализация процесса поиска
  void Init();
  // Определение множества I
  void CalculateI(void);
  // Вычисление значений относительных разностей
  void CalculateM();
  // Вычисление значений Z
  void CalculateZ();
  // Поиск интервала с максимальной характеристикой
  pTrial FindMaxR(void);
  // Проведение очередного испытания
  CTrial MakeTrial(double);
  // Вставка результатов очередного испытания
  bool InsertTrial(CTrial);
};
```

Файл *Method.cpp* содержит определение функций класса *CMethod*, реализующего индексный метод.

1. Функция *Run* – основная функция класса. Реализует схему индексного алгоритма, вызывает необходимые подпрограммы.

```
// Главная функция индексного метода
void CMethod::Run(){
  // Инициализация процесса поиска
  Init();
  // Условие остановки не выполнено
  bool stop=false;
  CTrial trial;
  while(!stop){
    // Определение множества I
    CalculateI();
    // Вычисление значений относительных разностей
    CalculateM();
    // Вычисление значений Z
    CalculateZ();
    // Поиск интервала с максимальной характеристикой
    pTrial t=FindMaxR();
    // Проведение испытания в интервале № t
    pTrial t1=t;
    t1--;
    if(t->index!=t1->index){
      trial=MakeTrial(0.5*(t->x+t1->x));
    }else{
```

```
      trial=MakeTrial(0.5*
        (t->x+t1->x)-(t->Value-t1->Value)/
        (M[t->index]*2*r[t->index]));
    }
  // Вставка результатов очередного испытания
    stop=InsertTrial(trial);
  }
}
```

2. Функция *Init* проводит начальную инициализацию текущих параметров алгоритма (максимального индекса и максимальных значений относительных первых разностей), а также выполняет вставку в массив испытаний граничных точек отрезка $[a,b]$ и проводит первое испытание в серединной точке отрезка.

```
// Инициализация процесса поиска
void CMethod::Init(){
  // Сначала максимальный индекс не определен
  MaxIndex=-1;
  // Первоначальное формирование данных
  for(int v=0;v<NumFuncs;v++)M[v]=1;
  // Заносим в массив испытаний граничные точки
  CTrial trial;
  trial.x=a;
  trial.index=-1;
  InsertTrial(trial);
  trial.x=b;
  trial.index=-1;
  InsertTrial(trial);
  // Проводим испытание во внутренней (средней) точке
  trial=MakeTrial((a+b)/2);
  InsertTrial(trial);
}
```

3. Функция *CalculateI* формирует множества I_ν, $1\le\nu\le m+1$, испытаний с фиксированным индексом ν из (0.16).

```
// Определение множества I
void CMethod::CalculateI(void){
  // Предварительная очистка множества I
  for(int v=0;v<NumFuncs;v++){
    I[v].clear();
  }
  // Заполнение множества I
```

```
  for(int v=0;v<NumFuncs;v++){
    pTrial i;
    for(i=Trials.begin();i!=Trials.end();i++){
      if(i->index==v){
        I[v].push_back(i);
      }
    }
  }
}
```

4. Функция *CalculateM* выполняет вычисление максимального значения относительных первых разностей μ_v из (0.17).

```
// Вычисление значений относительных разностей
void CMethod::CalculateM(){
  for(int v=0;v<NumFuncs;v++){
    if(I[v].size()<2){
      M[v]=1;
      continue;
    }
    double MaxM=-HUGE_VAL;
    double tM;
    for(unsigned i=1;i<I[v].size();i++){
      tM=fabs(I[v][i]->Value-I[v][i-1]-
>Value)/(I[v][i]->x-I[v][i-1]->x);
      if(MaxM<tM)MaxM=tM;
    }
    if(MaxM>0)M[v]=MaxM;
    else M[v]=1;
  }
}
```

5. Функция *CalculateZ* выполняет вычисление оценок z_v из (0.18).

```
// Вычисление значений Z
void CMethod::CalculateZ(){
  for(int v=0;v<NumFuncs;v++){
    if(v<MaxIndex){
      Z[v]=0;
      continue;
    }
    Z[v]=BestTrial.Value;
    break;
  }
}
```

6. Функция *FindMaxR* выполняет поиск интервала с максимальной характеристикой и возвращает указатель на этот интервал.

```
// Поиск интервала с максимальной характеристикой
pTrial CMethod::FindMaxR(void){
  pTrial t;
  pTrial i1=Trials.begin();
  pTrial i=Trials.begin();
  i++;
  double MaxR=-HUGE_VAL;
  double R;
  // Вычисление характеристик
  for(i;i!=Trials.end();i++,i1++){
    double deltax=i->x-i1->x;
    if(i->index == i1->index){
      int v=i->index;
      R=deltax+
       (i->Value-i1->Value)*(i->Value-i1->Value)/
       (deltax*M[v]*M[v]*r[v]*r[v])
       -2*(i->Value+i1->Value-2*Z[v])/(r[v]*M[v]);
    }
    if(i->index>i1->index){
      int v=i->index;
      R=2*deltax-4*(i->Value-Z[v])/(r[v]*M[v]);
    }
    if(i->index<i1->index){
      int v=i1->index;
      R=2*deltax-4*(i1->Value-Z[v])/(r[v]*M[v]);
    }
    if(R>MaxR){
      MaxR=R;
      t=i;
    }
  }
  return t;
}
```

7. Функция *MakeTrial* выполняет проведение очередного испытания в точке *x* и возвращает результаты испытания в виде структуры *CTrial*.

```
// Проведение очередного испытания
CTrial CMethod::MakeTrial(double x){
  CTrial Trial;
  Trial.x=x;
  // Проверка ограничений
```

```
  for(int i=0;i<NumFuncs;i++){
    Trial.Value=Funcs[i](x);
    if(i==NumFuncs-1||Trial.Value>0){
      Trial.index=i;
      break;
    }
  }
  return Trial;
}
```

8. Функция *InsertTrial* выполняет вставку результатов очередного испытания в упорядоченное множество испытаний. В случае выполнения критерия остановки функция возвращает *true*, иначе – *false*.

```
// Вставка результатов очередного испытания
bool CMethod::InsertTrial(CTrial Trial){
  // Вставка результатов
  std::pair<pTrial,bool> ins;
  ins=Trials.insert(Trial);
  // Точки итераций совпали - стоп
  if(!ins.second)return true;
  if(Trials.size()>2){
    pTrial j=ins.first,j1=ins.first;
    j++;
    j1--;
    // Выполнено условие остановки
    if((j->x-j1->x)<eps)return true;
  }
  // Оценка значений
  if(Trial.index>MaxIndex||Trial.index==
    MaxIndex&&Trial.Value<BestTrial.Value){
      BestTrial=Trial;
      MaxIndex=Trial.index;
  }
  return false;
}
```

12.4.4. Результаты численных экспериментов

В качестве иллюстрации рассмотрим задачу из примера 0.2: $x\in[0.6,2.2]$,

$$\varphi(x)=\cos(18x-3)\sin(10x-7)+1.5,$$
$$g_1(x)=\exp(-x/2)\sin(6x-1.5),$$

$$g_2(x)=|x|\sin(2\pi x-0.5).$$

В предположении частичной вычислимости дуги этих функций, соответствующие областям Q_j, $1\le j\le 2$, из (0.6), представлены на рис. 0.6.

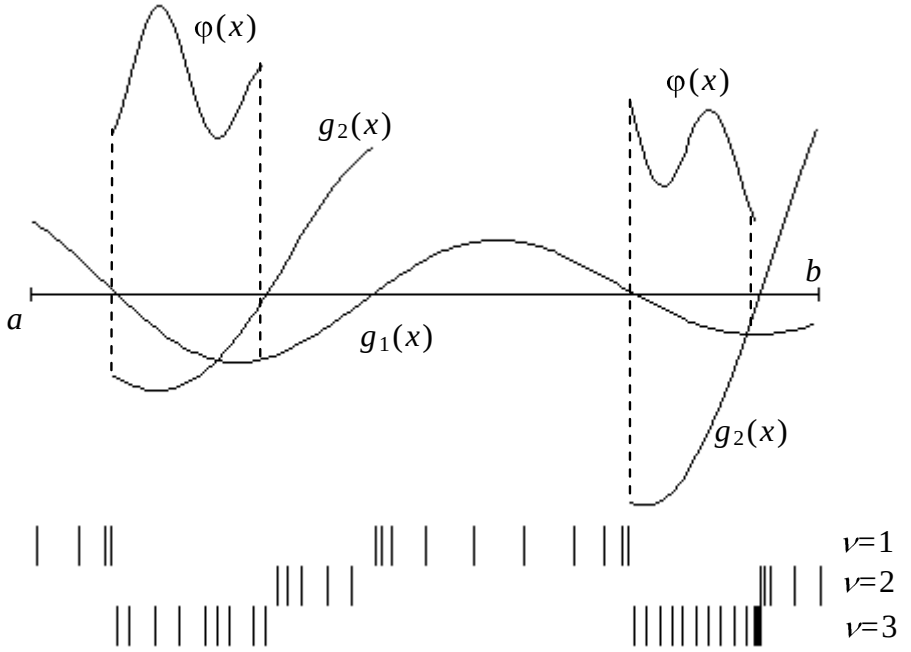

Рис. 0.6. Результаты решения задачи

Описанный индексный алгоритм был использован для решения этого примера при $r_\nu=2$, $1\le\nu\le 3$, и $\varepsilon=10^{-5}$ (исходный код программы см. ниже). Координаты точек испытаний, осуществленных алгоритмом в процессе решения задачи, отмечены на рис. 0.6 тремя рядами вертикальных штрихов. Штрихи верхнего ряда соответствуют точкам с единичным индексом, второго – точкам, индексы которых равны 2; точки, отмеченные штрихами нижнего ряда, являются допустимыми. Координаты испытаний, выполненных в близких точках, отмечены темным прямоугольником.

Следует отметить, что весьма часто в прикладных задачах оценка значений функций ограничений и целевой функции требует заметных вычислительных ресурсов. Результаты проведенного эксперимента подтверждают экономичность используемой индексной схемы учета ограничений, т. к. при этом значения функций g_1, g_2, $\varphi=g_3$ вычислялись соответственно $k_1=63$, $k_2=49$ и $k_3=35$ раз. Если бы поиск решения проводился бы на равномерной сетке, то для достижения такой же точности $\varepsilon=10^{-5}$ потре-

бовалось бы $k_1=1.6\cdot10^5$, $k_2=90280$ и $k_3=56476$ вычислений значений функций g_1, g_2, $\varphi=g_3$ соответственно.

```
// Решение тестового примера индексным методом
#include <iostream>
#include "method.h"
// Первое ограничение
double f1(double x){
  return exp(-0.5*x)*sin(6*x-1.5);
}
// Второе ограничение
double f2(double x){
  return fabs(x)*sin(2*M_PI*x-0.5);
}
// Целевая функция
double fi(double x){
  return cos(18*x-3)*sin(10*x-7)+1.5;
}

int main(int argc, char* argv[]){
  CMethod im;
  im.a=0.6;
  im.b=2.2;
  im.eps=0.00001;
  im.NumFuncs=3;
  im.Funcs[0]=f1;
  im.Funcs[1]=f2;
  im.Funcs[2]=fi;
  for(int i=0;i<3;i++)im.r[i]=2;
  im.Run();
  std::cout<<"Xmin="<<im.BestTrial.x<<"Ymin="<<
    im.BestTrial.Value<<std::endl;
  return 0;
}
```

12.5. Редукция размерности задачи

12.5.1. Использование отображений Пеано

Рассмотрим многомерную задачу глобальной оптимизации вида

$$\varphi(w^*)=\min\{\varphi(w):\ w\in S,\ g_j(w)\le 0,\ 1\le j\le m\},$$
$$S=\{s\in R^N:\ a_i\le s_i\le b_i,\ 1\le i\le N\}.$$

Нетрудно видеть, что с помощью преобразования

$$y_i=(w_i-(a_i+b_i)/2)/\rho,\quad \rho=\max\{b_i-a_i:1\le i\le N\},$$

и введения дополнительного ограничения

$$g_0(y)=\max\{|y_i|-(b_i-a_i)/2\rho : 1\le i\le N\}\le 0,$$

можно представить исходную задачу условной глобальной оптимизации, определенную на гиперинтервале S, как задачу на единичном N-мерном гиперкубе D:

$$\varphi(y^*)=\min\{\varphi(y):\ y\in D,\ g_j(y)\le 0,\ 1\le j\le m\},$$
$$D=\{y\in R^N:\ -2^{-1}\le y_i\le 2^{-1},\ 1\le i\le N\}=\{y(x):\ 0\le x\le 1\}. \qquad (0.23)$$

Использование развертки Пеано $y(x)$, однозначно отображающей единичный отрезок вещественной оси на единичный гиперкуб, позволяет свести многомерную задачу условной минимизации в области D к одномерной задаче условной минимизации на отрезке $[0,1]$

$$\varphi(y(x^*))=\min\{\varphi(y(x)):x\in[0,1],g_j(y(x))\le 0,1\le j\le m\}. \qquad (0.24)$$

Рассматриваемая схема редукции размерности сопоставляет многомерной задаче с липшицевой минимизируемой функцией и липшицевыми левыми частями ограничений и одномерную задачу, в которой соответствующие функции удовлетворяют равномерному условию Гельдера (см. [28,97]), т. е.

$$|g_j(x')-g_j(x'')|\le K_j|x'-x''|^{1/N},\ x',\ x''\in[0,1],\ 1\le j\le m+1,$$

где N есть размерность исходной многомерной задачи, а коэффициенты K_j связаны с константами Липшица L_j исходной задачи соотношениями $K_j\le 4L_j\sqrt{N}$. Следовательно, все длины интервалов, участвующих в правилах алгоритма поиска, должны быть заменены на длины в новой метрике, в которой расстояние определяется выражением

$$\Delta_i=|x_i-x_{i-1}|^{1/N}.$$

Для этого заменим в выражениях (0.17), (0.19) и (0.22) разности x_i-x_{i-1} величинами $|x_i-x_{i-1}|^{1/N}$, а вместо (0.21) введем выражение

$$x^{k+1}=(x_t+x_{t-1})/2-\operatorname{sign}(z_t-z_{t-1})\frac{1}{2r_\nu}\left[\frac{|z_t-z_{t-1}|}{\mu_\nu}\right]^N,$$
$$\nu=\nu(x_{t-1})=\nu(x_t).$$

В результате получаем алгоритм решения задачи (0.24), описание которого в целом совпадает со схемой одномерного индексного алгоритма (см. [94,97]).

Вопросы численного построения отображений типа кривой Пеано и соответствующая теория подробно рассмотрены в работах [28,97]. Здесь же отметим, что численно построенная кривая является приближением к теоретической кривой Пеано с точностью, не хуже 2^{-m} по каждой координате (параметр m называется *плотностью развертки*).

12.5.2. Программная реализация

Представим для рассмотрения функции, реализующие отображение типа кривой Пеано.

1. Функция *mapd* реализует прямое отображение: для заданной точки $x \in [0,1]$, плотности развертки m и размерности пространства n вычисляется образ $y(x) \in D$.

```
// Программа 12.2
// Реализация разверток
int n1,nexp,l,iq,iu[10],iv[10];
// Прямое отображение
void mapd( double x, int m, float* y, int n ) {
  double d,mne,dd,dr,tmp;
  float p,r;
  int iw[11];
  int it,is,i,j,k;

  p=0.0;
  n1=n-1;
  for ( nexp=1,i=0; i<n; nexp*=2,i++ );
  d=x;  r=0.5;  it=0;  dr=nexp;
  for ( mne=1,i=0; i<m; mne*=dr,i++ );
  for ( i=0; i<n; i++ ){ iw[i]=1; y[i]=0.0;}
  for ( j=0; j<m; j++ ) {
    iq=0;
    if ( x == 1.0 ) {
      is=nexp-1; d=0.0;
    } else {
      d=d*nexp;
      is=d;
      d=d-is;
    }
    i=is;
    node(i);
    i=iu[0];
    iu[0]=iu[it];
```

```
    iu[it]=i;
    i=iv[0];
    iv[0]=iv[it];
    iv[it]=i;
    if ( l == 0 ) l=it;
    else if ( l == it ) l=0;
    if ( (iq>0)||((iq==0)&&(is==0)) )  k=l;
    else if ( iq<0 ) k = ( it==n1 ) ? 0 : n1;
    r=r*0.5;
    it=l;
    for ( i=0; i<n; i++ ) {
      iu[i]=iu[i]*iw[i];
      iw[i]=-iv[i]*iw[i];
      p=r*iu[i];
      p=p+y[i];
      y[i]=p;
    }
  }
}
// Вспомогательная функция
void node ( int is ) {
  int n,i,j,k1,k2,iff;

  n=n1+1;
  if ( is == 0 ) {
    l=n1;
    for ( i=0; i<n; i++ ) { iu[i]=-1; iv[i]=-1;}
  } else if ( is == (nexp-1) ) {
    l=n1;
    iu[0]=1;
    iv[0]=1;
    for ( i=1; i<n; i++ ) { iu[i]=-1; iv[i]=-1;}
    iv[n1]=1;
  } else {
    iff=nexp;
    k1=-1;
    for ( i=0; i<n; i++ ) {
      iff=iff/2;
      if ( is >= iff ) {
      if ( (is==iff)&&(is != 1) )  { l=i; iq=-1; }
        is=is-iff;
        k2=1;
      }
```

```
      else {
        k2=-1;
        if ( (is==(iff-1))&&(is!= 0) )  { l=i; iq=1; }
      }
      j=-k1*k2;
      iv[i]=j;
      iu[i]=j;
      k1=k2;
    }
    iv[l]=iv[l]*iq;
    iv[n1]=-iv[n1];
  }
}
```

2. Функция *xyd* реализует обратное отображение: для заданной точки $y \in D$, плотности развертки m и размерности пространства n вычисляется ее прообраз $y^{-1}(x) \in [0,1]$.

```
// Программа 12.3
// Обратное отображение
void xyd ( double *xx, int m, float y[], int n) {
  double x,r1;
  float r;
  int iw[10];
  int i,j,it,is;

  n1=n-1;
  for ( nexp=1,i=0; i<n; i++ ) { nexp*=2; iw[i]=1; }
  r=0.5;  r1=1.0;  x=0.0;  it=0;
  for ( j=0;  j<m; j++ ) {
    r*=0.5;
    for ( i=0; i<n; i++ ) {
      iu[i] = ( y[i]<0 ) ? -1 : 1;
      y[i]-=r*iu[i];
      iu[i]*=iw[i];
    }
    i=iu[0];
    iu[0]=iu[it];
    iu[it]=i;
    numbr ( &is );
    i=iv[0];
    iv[0]=iv[it];
    iv[it]=i;
    for ( i=0; i<n; i++ ) iw[i]=-iw[i]*iv[i];
```

```
    if ( l == 0 ) l=it;
    else if ( l == it ) l=0;
    it=l;
    r1=r1/nexp;
    x+=r1*is;
  }
  *xx=x;
}
// Вспомогательная функция
void numbr ( int *iss) {
  int i,n,is,iff,k1,k2,l1;

  n=n1+1; iff=nexp;
  is=0; k1=-1;
  for ( i=0; i<n; i++ ) {
    iff=iff/2;
    k2=-k1*iu[i];
    iv[i]=iu[i];
    k1=k2;
    if ( k2<0 ) l1=i;
    else { is+=iff; l=i; }
  }
  if ( is == 0 ) l=n1;
  else {
    iv[n1]=-iv[n1];
    if ( is == (nexp-1) ) l=n1;
    else if ( l1 == n1 ) iv[l]=-iv[l];
    else l=l1;
  }
  *iss=is;
}
```

12.5.3. Результаты численных экспериментов

В качестве иллюстрации рассмотрим задачу из примера 0.1: минимизировать функцию

$$\varphi(y_1, y_2) = -1.5y_1^2 \exp\{1 - y_1^2 - 20.25(y_1 - y_2)^2\} -$$
$$-[0.5(y_1 - 1)(y_2 - 1)]^4 \exp\{2 - [0.5(y_1 - 1)]^4 - (y_2 - 1)^4\}$$

в области поиска $0 \le y_1 \le 4$, $-1 \le y_2 \le 3$, при ограничениях

$$g_1(y_1, y_2) = 0.01[(y_1 - 2.2)^2 + (y_2 - 1.2)^2 - 2.25] \le 0$$

$$g_2(y_1, y_2) = 100[1-(y_1-2)^2/1.44-(0.5y_2)^2] \le 0$$
$$g_3(y_1, y_2) = 10[y_2 - 1.5 - 1.5\sin(6.283(y_1 - 1.75))] \le 0$$

Напомним, что допустимые по первому ограничению точки образуют круг с границей $g_1(y_1,y_2)=0$. Допустимые по второму ограничению точки находятся во внешности эллипса $g_2(y_1,y_2)=0$. Точки, допустимые по третьему ограничению, находятся ниже синусоиды $g_3(y_1,y_2)=0$. Глобальный минимум $\varphi(y_1^*, y_2^*) = -1.489$ достигается в точке (y_1^*, y_2^*)=(0.942, 0.944).

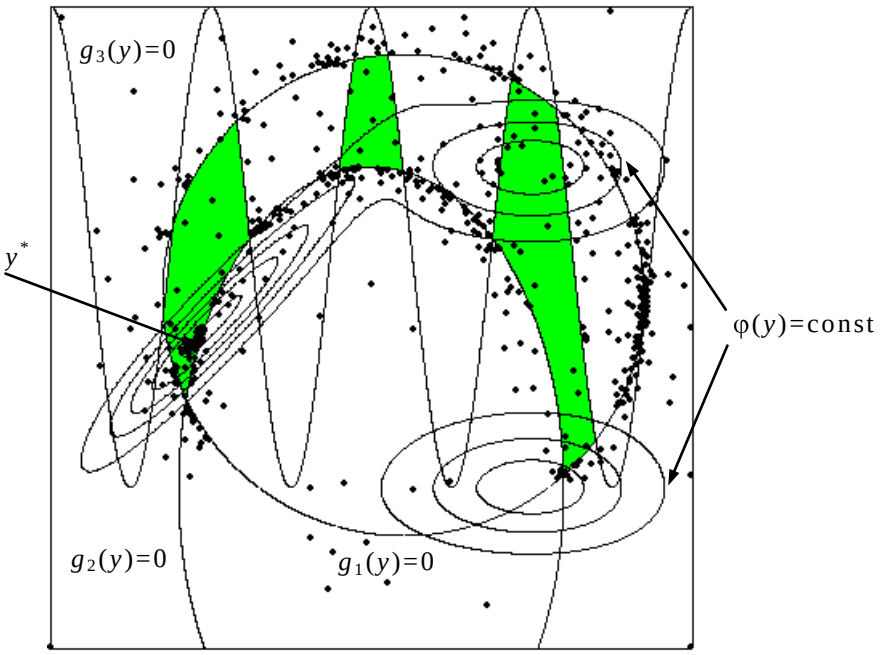

Рис. 0.7. Результаты решения задачи

На рис. 0.7 изображена область поиска – квадрат, где обозначены точки 1098 испытаний, полученные применением индексного метода со следующими параметрами: параметр надежности $r_1=\ldots=r_4=2$, точность решения $\varepsilon=10^{-3}$, плотность построения развертки $m=12$.

Результаты проведенного эксперимента подтверждают экономичность предлагаемого подхода, т. к. при этом значения функций g_1, g_2, g_3, $\varphi=g_4$ вычислялись соответственно $k_1=1098$, $k_2=623$, $k_3=392$ и $k_4=152$ раз. Если бы поиск решения проводился на равномерной сетке, то для достижения такой же точности $\varepsilon=10^{-3}$ потребовалось бы $k_1=1.6\cdot10^7$, $k_2=7\cdot10^6$

$k_3=3\cdot10^6$ и $k_4=1.4\cdot10^6$ вычислений значений функций g_1, g_2, g_3, $\varphi=g_4$ соответственно.

12.5.4. Способ построения развертки

Для желающих подробно познакомиться со способом построения кривых Пеано опишем схему построения развертки (см. [28,97]), отображающей единичный отрезок вещественной оси [0,1] на гиперкуб D,

$$D=\{y\in R^N:\ -2^{-1}\le y_i\le 2^{-1},\ 1\le i\le N\}.$$

1. Гиперкуб D, длина ребра которого равна 1, разделяется координатными плоскостями на 2^N гиперкубов первого разбиения (с длиной ребра, равной 1/2), которые нумеруются числами z_1 от 0 до 2^N-1, причем гиперкуб первого разбиения с номером z_1 условимся обозначать через $D(z_1)$.

Далее, каждый гиперкуб первого разбиения, в свою очередь, также разбивается на 2^N гиперкубов второго разбиения (с длиной ребра, равной 1/4) гиперплоскостями, параллельными координатным осям и проходящими через серединные точки ребер гиперкуба, ортогональных к этим гиперплоскостям. При этом гиперкубы второго разбиения, входящие в гиперкуб $D(z_1)$, нумеруются числами z_2 от 0 до 2^N-1, причем гиперкуб второго разбиения с номером z_2, входящий в $D(z_1)$, обозначается через $D(z_1, z_2)$. Случай $N=2$ изображен на рис. 0.8 для $m=1,\ m=2$.

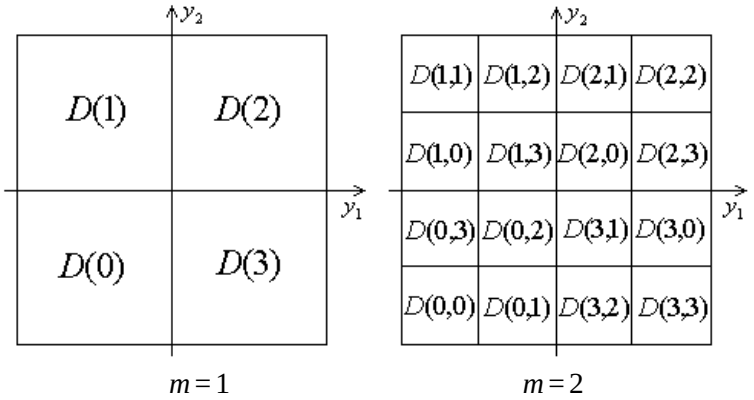

Рис. 0.8. Разбиение N-мерного гиперкуба

Продолжая указанный процесс, можно построить гиперкубы любого m-го разбиения с длиной ребра, равной $(1/2)^m$, которые обозначаются как $D(z_1,\ldots,z_m)$, причем $D(z_1)\supset D(z_1, z_2)\supset\ldots\supset D(z_1,\ldots,z_m)$ и $0\le z_j\le 2^N-1,\ \ 1\le j\le m$.

2. Теперь осуществим деление отрезка $[0,1]$ на 2^N равных частей, каждую из которых, в свою очередь, также разделим на 2^N равных частей и так далее, причем элементы каждого разбиения нумеруются слева направо числами z_j от 0 до 2^N-1, где j – номер разбиения. При этом интервалы m-го разбиения обозначим как $d(z_1,\ldots,z_m)$, где, например, $d(z_1,z_2)$ обозначает интервал второго разбиения с номером z_2, являющийся частью интервала $d(z_1)$ первого разбиения с номером z_1.

Можно отметить, что $d(z_1)\supset d(z_1, z_2)\supset\ldots\supset d(z_1,\ldots,z_m)$, и длина интервала $d(z_1,\ldots,z_m)$ равна $(1/2)^{mN}$. Предполагается, что интервал $d(z_1,\ldots,z_m)$ содержит свой левый конец. Он содержит правый конец тогда и только тогда, когда $z_1=z_2=\ldots=z_m=2^N-1$. Случай $N=2$ изображен на рис. 0.9 для $m=1$ и $m=2$.

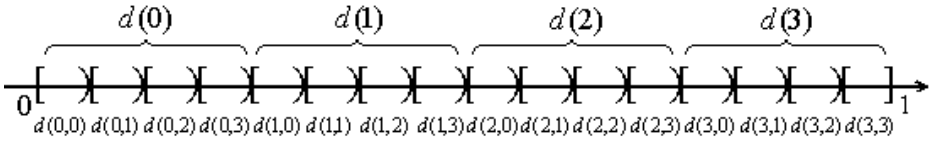

Рис. 0.9. Разбиение одномерного отрезка

3. Примем, что точка $y(x)\in D$, соответствующая точке $x\in[0,1]$, при любом $m\geq 1$ содержится в гиперкубе $D(z_1,\ldots,z_m)$, если x принадлежит интервалу $d(z_1,\ldots,z_m)$, т. е.

$$x\in d(z_1,\ldots,z_m)\ \rightarrow\ y(x)\in D(z_1,\ldots,z_m).$$

Построенное соответствие $y(x)$ является однозначным.

4. Чтобы построенное отображение было еще и непрерывным, наложим требования на порядок нумерации гиперкубов каждого разбиения. Так как 2^{mN} центров $y(z_1,\ldots,z_m)$ гиперкубов m-го разбиения $D(z_1,\ldots,z_m)$ образуют равномерную ортогональную сетку в области D (причем шаг этой сетки по любой координате равен 2^{-m}), то можно установить следующую нумерацию узлов этой сетки. Пронумеруем слева направо по i все интервалы, составляющие m-е разбиение отрезка $[0,1]$, т. е.

$$d(z_1,\ldots,z_m)=[x_i,\ x_{i+1}),\ 0\leq i<2^{mN}-1,$$

где через x_i обозначен левый конец интервала, имеющего номер i.

Будем считать, что центр гиперкуба $D(z_1,\ldots,z_m)$ имеет тот же номер i, что и соответствующий этому гиперкубу интервал $d(z_1,\ldots,z_m)$, т. е.

$$y_i=y(z_1,\ldots,z_m),\ 0\leq i<2^{mN}-1.$$

При этом центры y_i и y_{i+1} соответствуют смежным гиперкубам, имеющим общую грань.

Рассмотрим отображение $l(x)$ отрезка $[0,1]$ в гиперкуб D, определяемое выражениями

$$l(x) = y_i + (y_{i+1} - y_i)\left(\frac{w(x) - x_i}{x_{i+1} - x_i}\right),\ x_i \le w(x) \le x_{i+1},$$

$$w(x) = x(1 - 2^{-mN}),\ 0 \le x \le 1.$$

Образ любого подынтервала отрезка $[0,1]$ вида

$$[x_i(1-2^{-mN})^{-1},\ x_{i+1}(1-2^{-mN})^{-1}],\ 0 \le i < 2^{mN} - 1,$$

при соответствии $l(x)$ является линейным отрезком, соединяющим узлы y_i и y_{i+1}.

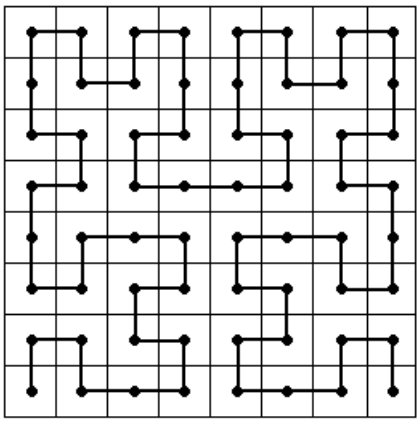

Рис. 0.10. Кусочно-линейная развертка

Таким образом, мы построили кусочно-линейную кривую $l_m(x)$, $0 \le x \le 1$, соединяющую узлы y_i, $0 \le i < 2^{mN} - 1$, в порядке их нумерации. Эта кривая является приближением к кривой Пеано с точностью, не хуже 2^{-m} по каждой координате. Для иллюстрации на рис. 0.10 изображен образ отрезка $[0,1]$ при соответствии $l(x)$ для случая $N = 2$, $m = 3$ (узлы сетки отмечены темными кружками).

12.6. Использование множественных отображений

12.6.1. Основная схема

Редукция многомерных задач к одномерным с помощью разверток имеет такие важные свойства, как непрерывность и сохранение равномерной ограниченности разностей функций при ограниченности вариации аргумента. Однако при этом происходит частичная потеря части информации о близости точек в многомерном пространстве, ибо точка $x \in [0,1]$ имеет лишь левых и правых соседей, а соответствующая ей точка $y(x) \in R^N$ имеет соседей по 2^N направлениям. Как результат, при использовании отображений типа кривой Пеано близким в N-мерном пространстве образам y', y'' могут соответствовать достаточно далекие прообразы x', x'' на отрезке [0,1].

Возможным способом преодоления этого недостатка является использование множественных отображений

$$Y_L(x)=\{y^0(x),\ y^1(x),\ldots,\ y^L(x)\} \qquad (0.25)$$

вместо применения единственной кривой Пеано $y(x)$ (см. [94,97]).

Каждая кривая Пеано $y^i(x)$ из $Y_L(x)$ может быть получена в результате некоторого сдвига вдоль главной диагонали гиперинтервала D.

Таким образом, сконструированное множество кривых Пеано позволяет получить для любых близких образов y', y'', отличающихся только по одной координате, близкие прообразы x', x'' для некоторого отображения $y^i(x)$.

Использование множества отображений приводит к формированию соответствующего множества одномерных многоэкстремальных задач

$$\min\{\varphi(y^l(x)): x \in [0,1],\ g_j(y^l(x)) \le 0,\ 1 \le j \le m\},\ 0 \le l \le L.$$

Каждая задача из данного набора может решаться независимо, при этом любое вычисленное значение $z=g_\nu(y')$, $\nu=\nu(y')$, $y'=y^i(x')$ функции $g_\nu(y)$ в i-й задаче может интерпретироваться как вычисление значения $z=g_\nu(y')$, $\nu=\nu(y')$, $y'=y^s(x'')$ для любой другой s-й задачи без повторных трудоемких вычислений функции $g_\nu(y)$. Подобное информационное единство дает возможность решать весь набор задач параллельно.

12.6.2. Программная реализация

Представим для рассмотрения функции, реализующие множественные отображения.

1. Функция *GetImage* реализует прямое отображение: для заданной точки $x \in [0,1]$, плотности развертки m, размерности пространства n и номера развертки l вычисляется ее образ $y^l(x)$.

```
// Программа 12.4
// Вычисление образа точки из отрезка [0,1]
void GetImage(double x,int n,int m,int l,float y[]){
  double del;
  int i;
  if (l == 0) del = 0.0;
  else for (i = 1, del = 1; i < l + 1; del /= 2, i++);
  mapd(x, m+1, y, n, 1);
  for (i = 0; i < n; i++)
    y[i] = 2 * y[i] + 0.5 - del;
}
```

2. Функция *GetPreimages* реализует обратное отображение: для заданной точки y из многомерного пространства, плотности развертки m, размерности пространства n и числа разверток L вычисляются все ее прообразы x^0, x^1,…, x^L.

```
// Программа 12.5
// Вычисление всех прообразов точки
void GetPreimages( float* p, int n, int m, int L,
  double xp[]){
  int i, j;
  double xx;
  double del;
  float* p2=new float[n];
  del = 0.5;
  for (i = 1; i < L + 1; i++){
    for (j = 0; j < n; j++)
      p2[j] = (p[j] + del - 0.5) * 0.5;
    xyd(&xx, m + 1, p2, n);
    xp[i] = xx;
    del *= 0.5;
  }
  del = 0.0;
  for (j = 0; j < n; j++)
    p2[j] = (p[j] + del - 0.5) * 0.5;
  xyd(&xx, m + 1, p2, n);
  xp[0] = xx;
}
```

12.6.3. Способ построения множественных отображений

Материал данного пункта предназначен для желающих подробно ознакомиться со способом построения множественных отображений. Итак, рассмотрим семейство гиперкубов

$$D_l=\{y\in R^N:\ -2^{-1}\le y_i+2^{-l}\le 3\cdot 2^{-1},\ 1\le i\le N\},\ 0\le l\le L, \tag{0.26}$$

где гиперкуб D_{l+1} получается путем «сдвига» гиперкуба D_l вдоль главной диагонали на шаг 2^{-l} по каждой координате. На рис. 0.11 изображены гиперкубы D_0, …, D_3 для случая L=3.

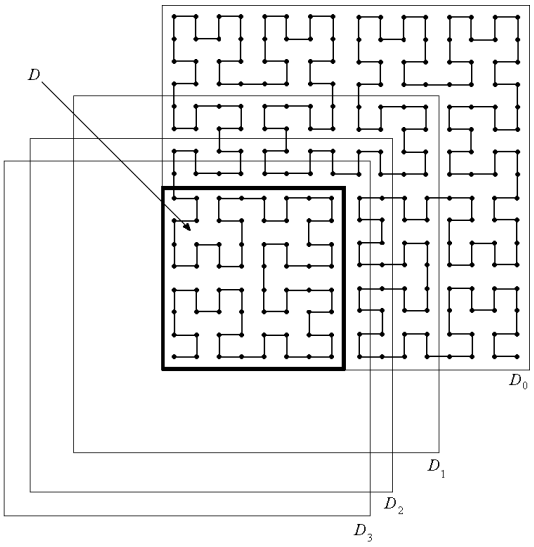

Рис. 0.11. Множественная развертка

Примем, что развертка $y^0(x)$ типа кривой Пеано отображает отрезок $[0,1]$ на гиперкуб D_0 из (0.26), т. е.

$$D_0=\{y^0(x):x\in[0,1]\}. \tag{0.27}$$

Тогда развертки $y^l(x)=\{\ y_1^l(x),...,y_N^l(x)\ \}$, координаты которых определяются условиями

$$y_i^l(x)=y_i^{l-1}(x)+2^{-l},\ 1\le i\le N,\ 1\le l\le L, \tag{0.28}$$

отображают отрезок $[0,1]$ на соответствующие гиперкубы $D_l, 0\le l\le L$ (на рис. 0.11 ломаной линией изображен образ отрезка $[0,1]$, который получается при использовании развертки $y^0(x), x\in[0,1]$). Поскольку гиперкуб D из (0.23) входит в общую часть семейства гиперкубов семейства (0.26) (граница гиперкуба D выделена на рис. 0.11), то, введя дополнительную функцию ограничения

$$g_0(y)=\max\{|y_i|-2^{-1}: 1\le i\le N\}, \tag{0.29}$$

исходный гиперкуб D можно представить в виде

$$D=\{y^l(x): x\in[0,1],\ g_0(y^l(x))\le 0\},\ 0\le l\le L, \tag{0.30}$$

т. е. $g_0(y)\le 0$, если $y\in D$, и $g_0(y)>0$ в противном случае. Следовательно, любая точка $y\in D$ имеет свой прообраз $x^l\in[0,1]$ при каждом соответствии $y^l(x)$, $0\le l\le L$.

Таким образом, каждая развертка $y^l(x)$, $0\le l\le L$, порождает свою задачу вида (0.4), характеризуемую своей расширенной по сравнению с D областью поиска D_l и дополненную ограничением с левой частью из (0.29):

$$\min\{\varphi(y^l(x)): x\in[0,1],\ g_j(y^l(x))\le 0,\ 0\le j\le m\},\ 0\le l\le L. \tag{0.31}$$

При этом задачам (0.31) соответствуют область $Q_0=[0,1]$ и области $Q_{j+1}^l, 0\le j\le m$, определяемые вторым выражением из (0.6) для соответствующих разверток $y^l(x)$. Применение множественного отображения определяет следующую связь окрестностей в многомерных и одномерных областях поиска.

Утверждение. Пусть y^* – произвольная точка из области поиска D, принадлежащая отрезку с концевыми точками $y',y''\in D$, различающимися значениями единственной координаты, и пусть

$$\left|y'_j-y''_j\right|\le 2^{-p},\ \ y'_i=y''_i=y^*_i,\ 1\le i\le N,\ i\ne j,$$

где p – целое число, $1\le p\le L-1$, и номер координаты, значения которой для точек y^*,y',y'' являются различными, обозначен через j. Тогда существует хотя бы одно соответствие $y^l(x)$, $0\le l\le L$, и прообразы $x^*,x',x''\in[0,1]$ такие, что

$$y^*=y^l(x^*),\ y'=y^l(x'),\ y''=y^l(x''),$$
$$\max\{|x'-x^*|,|x''-x^*|,|x'-x''|\}\le 2^{-pN}.$$

Доказательство данного утверждения изложено в работе [97].

Замечание. Условия утверждения выделяют специфическую окрестность точки y^*. Эта окрестность включает в себя лишь такие точки, которые могут быть получены путем смещения y^* параллельно одной из осей координат на расстояние, не превышающее 2^{-p}. Изменяя значение j, $1\le j\le N$, в условиях теоремы, можно выделить ближайших «соседей» точки y^* по N координатным направлениям. Согласно утверждению, близость точек в N-мерном пространстве по конкретному направлению будет

отражена близостью их прообразов в одной из одномерных подзадач. Сохранение информации о близости точек приводит, во-первых, к более точной оценке констант Липшица, и, во-вторых, к увеличению характеристик интервалов, образы граничных точек которых являются близкими в N-мерном пространстве.

12.7. Параллельный индексный метод

12.7.1. Организация параллельных вычислений

Использование множественных отображений позволяет решать исходную задачу (0.1) путем параллельного решения индексным методом $L+1$ задач вида (0.31) на наборе отрезков $[0,1]$. Каждая одномерная задача (или группа задач при недостаточном количестве процессоров) решается на отдельном процессоре. Результаты испытания в точке x^k, полученные конкретным процессором для решаемой им задачи, интерпретируются как результаты испытаний во всех остальных задачах (в соответствующих точках x^{k0}, $x^{k1},\ldots,x^{kL}$). При таком подходе испытание в точке $x^k\in[0,1]$, осуществляемое в s-й задаче, состоит в последовательности действий:

1. Определить образ $y^k=y^s(x^k)$ при соответствии $y^s(x)$.

2. Вычислить величину $g_0(y^k)$. Если $g_0(y^k)\le 0$, то есть $y^k\in D$, то проинформировать остальные процессоры о начале проведения испытания в точке y^k (*блокирование* точки y^k).

3. Вычислить величины $g_1(y^k),\ldots,g_\nu(y^k)$, где значения индекса $\nu\le m$ определяются условиями

$$g_j(y^k)\le 0,\ 1\le j<\nu,\ g_\nu(y^k)>0,\ \nu\le m.$$

Выявление первого нарушенного ограничения прерывает испытание в точке y^k. В случае, когда точка y^k допустима, т. е. когда $y^s(x^k)\in Q_{m+1}$, испытание включает вычисление значений всех функционалов задачи. При этом значение индекса принимается равным величине $\nu=m+1$, а тройка

$$x^k),\ \nu=\nu(x^k),\ z^k=g_\nu(y^s(x^k)), \tag{0.32}$$

является *результатом испытания* в точке x^k.

4. Если $\nu(x^k)>0$, то есть $y^k\in D$, то определить прообразы $x^{kl}\in[0,1]$, $0\le l\le L$, точки y^k, и интерпретировать испытание, проведенное в точке $y^k\in D$, как проведение испытаний в $L+1$ точке

$$x^{k0},\ x^{k1},\ldots,\ x^{kL}, \tag{0.33}$$

с одинаковыми результатами

$$\nu(x^{k0})=\nu(x^{k1})=\ldots=\nu(x^{kL})=\nu(x^k),$$
$$g_\nu(y^0(x^{k0}))=g_\nu(y^1(x^{k1}))=\ldots=g_\nu(y^L(x^{kL}))=z^k.$$

Проинформировать остальные процессоры о результатах испытания в точке y^k.

В случае, если $\nu(x^k)=0$, то есть $y^k \notin D$, результат испытания относится только к s-й задаче.

Каждый процессор имеет свою копию программных средств, реализующих вычисление функционалов задачи, и решающее правило алгоритма. Для организации взаимодействия на каждом процессоре создается $L+1$ очередь, в которые процессоры помещают информацию о выполненных итерациях в виде троек: точка очередной итерации, индекс и значение из (0.32), причем индекс заблокированной точки полагается равным –1, а значение функции в ней не определено. Связи между процессорами посредством очередей Q_{ls}, где l, $0\le l\le L$, номер передающего процессора и s, $0\le s\le L$, номер принимающего процессора, проиллюстрированы на рис. 0.12.

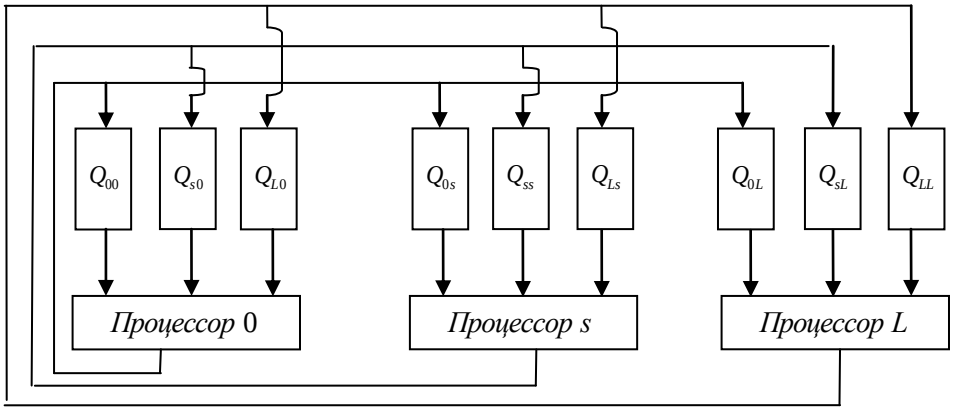

Рис. 0.12. Связи между процессорами

Предлагаемая схема не содержит какого-либо единого управляющего процессора, что увеличивает надежность выполняемых вычислений.

12.7.2. Схема алгоритма

Решающие правила предлагаемого параллельного алгоритма в целом совпадают с правилами последовательного алгоритма (кроме способа проведения испытания). Однако для целей более простого освоения материала схема изложена полностью.

Алгоритм выбора точек итераций для всех процессоров одинаков. Начальная итерация осуществляется в заданной точке $x^1 \in (0,1)$ (начальные точки для всех процессоров задаются различными). Выбор точки x^{q+1}, $q \geq 1$, любого последующего испытания определяется следующими правилами.

Правило 1. Изъять из всех очередей, закрепленных за данным процессором, записанные для него результаты, включающие множество $Y_q = \{y^{qi} : 1 \leq i \leq s_q\}$ точек итераций в области (0.23) и вычисленные в них значения индекса и величин из (0.32). Определить множество $X_q = \{x^{qi} : 1 \leq i \leq s_q\}$ прообразов точек множества Y_k при соответствии $y^l(x)$.

Правило 2. Точки множества итераций

$$\{x_1\} \cup X_1 \cup \ldots \cup X_q$$

перенумеровать нижними индексами в порядке увеличения значений координаты

$$0 = x_0 < x_1 < \ldots < x_i < \ldots < x_k < x_{k+1} = 1, \tag{0.34}$$

где $k = 1 + s_1 + \ldots + s_q$, и сопоставить им значения $z_i = g_\nu(x_i)$, $\nu = \nu(x_i)$, $1 \leq i \leq k$, вычисленные в этих точках. При этом индекс блокированной точки x_i (т. е. точки, в которой начато проведение испытания другим процессором) полагается равным –1, т. е. $\nu(x_i) = -1$, значение z_i является неопределенным. Точки x_0, x_{k+1} введены дополнительно для удобства последующего изложения, индекс данных точек полагается равным –2, т. е. $\nu(x_0) = \nu(x_{k+1}) = -2$, а значения z_0, z_{k+1} являются неопределенными.

Правило 3. Провести классификацию номеров $i, 1 \leq i \leq k$, точек из ряда (0.34) по числу ограничений задачи, выполняющихся в этих точках, путем построения множеств

$$I_{-2} = \{0,\ k+1\},$$
$$I_{-1} = \{i:\ 1 \leq i \leq k,\ \nu(x_i) = -1\},$$
$$I_\nu = \{i:\ 1 \leq i \leq k,\ \nu(x_i) = \nu\},\ 0 \leq \nu \leq m+1,$$

содержащих номера всех точек $x_i, 1 \leq i \leq k$, имеющих индексы, равные одному и тому же значению ν. Определить максимальное значение индекса

$$M = \max\{\nu = \nu(x_i),\ 1 \leq i \leq k\}.$$

Правило 3. Вычислить текущие нижние границы

$$\mu_\nu = \max\left\{\frac{|z_i - z_j|}{(x_i - x_j)^{1/N}} : j < i,\ j \in I_\nu, i \in I_\nu\right\}, \qquad (12.35)$$

для относительных разностей функций $g_\nu, 1 \le \nu \le m+1$. Если множество I_ν содержит менее двух элементов или если μ_ν из **Ошибка! Источник ссылки не найден.** оказывается равным нулю, то принять $\mu_\nu = 1$. Из **Ошибка! Источник ссылки не найден.** следует, что оценки μ_ν являются неубывающими, начиная с момента, когда **Ошибка! Источник ссылки не найден.** порождает первое положительное значение μ_ν.

Правило 4. Для всех непустых множеств I_ν, $1 \le \nu \le m+1$, вычислить оценки

$$z_\nu^* = \begin{cases} 0, & \nu < M, \\ \min\{g_\nu(x_i) : i \in I_\nu\}, & \nu = M, \end{cases}$$

Правило 5. Для каждого интервала $(x_{i-1}, x_i), 1 \le i \le k+1$, вычислить характеристику $R(i)$, где

$$R(i) = \Delta_i + \frac{(z_i - z_{i-1})^2}{r_\nu^2 \mu_\nu^2 \Delta_i} - 2\frac{(z_i + z_{i-1} - 2z_\nu^*)}{r_\nu \mu_\nu}, \quad \nu = \nu(x_{i-1}) = \nu($$

$$R(i) = 2\Delta_i - 4\frac{(z_i - z_\nu^*)}{r_\nu \mu_\nu}, \quad \nu(x_{i-1}) < \nu(x_i) = \nu,$$

$$R(i) = 2\Delta_i - 4\frac{(z_{i-1} - z_\nu^*)}{r_\nu \mu_\nu}, \quad \nu = \nu(x_{i-1}) > \nu(x_i),$$

$$\Delta_i = (x_i - x_{i-1})^{1/N}.$$

Величины $r_\nu > 1$, $1 \le \nu \le m+1$, являются параметрами алгоритма.

Правило 6. Определить интервал (x_{t-1}, x_t), которому соответствует максимальная характеристика

$$R(t) = \max\{R(i) :\ 1 \le i \le k+1\}. \qquad (0.35)$$

Правило 7. Провести очередное испытание в серединной точке интервала (x_{t-1}, x_t), если индексы его концевых точек не совпадают, т.е.

$$x^{q+1} = (x_t + x_{t-1})/2, \quad \nu(x_{t-1}) \ne \nu(x_t).$$

В противном случае провести испытание в точке

$$x^{q+1}=(x_t+x_{t-1})/2-\mathrm{sign}(z_t-z_{t-1})\frac{1}{2r_\nu}\left[\frac{|z_t-z_{t-1}|}{\mu_\nu}\right]^N,$$
$$\nu=\nu(x_{t-1})=\nu(x_t).$$

В случае если $\nu(x^{q+1})=0$, т. е. $y^q\notin D$, то результаты испытания занести только в очередь, закрепленной за данным процессором на нем самом. Если же $\nu(x^{q+1})>0$, т. е. $y^q\in D$, то результаты испытания занести во все очереди, закрепленные за данным процессором. Увеличив q на единицу, перейти к новой итерации.

Описанные правила можно дополнить условием остановки, прекращающим испытания, если

$$\Delta_t\le\Delta,$$

где t из (0.35) и $\Delta>0$ имеет порядок желаемой покоординатной точности в задаче.

12.7.3. Результаты численных экспериментов

Вычислительные эксперименты для оценки эффективности параллельного индексного метода проводились на кластере на базе процессоров Pentium III Xeon 1000 Мгц и сети Gigabit.

В качестве тестовой рассматривалась задача из примера 0.1: минимизировать функцию

$$\varphi(y_1,y_2)=-1.5y_1^2\exp\{1-y_1^2-20.25(y_1-y_2)^2\}-$$
$$-[0.5(y_1-1)(y_2-1)]^4\exp\{2-[0.5(y_1-1)]^4-(y_2-1)^4\}$$

в области поиска $0\le y_1\le 4$, $-1\le y_2\le 3$, при ограничениях

$$g_1(y_1,y_2)=0.01[(y_1-2.2)^2+(y_2-1.2)^2-2.25]\le 0$$
$$g_2(y_1,y_2)=100[1-(y_1-2)^2/1.44-(0.5y_2)^2]\le 0$$
$$g_3(y_1,y_2)=10[y_2-1.5-1.5\sin(6.283(y_1-1.75))]\le 0$$

Допустимые по первому ограничению точки образуют круг с границей $g_1(y_1,y_2)=0$. Допустимые по второму ограничению точки находятся во внешности эллипса $g_2(y_1,y_2)=0$. Точки, допустимые по третьему ограничению, находятся ниже синусоиды $g_3(y_1,y_2)=0$. Глобальный минимум $\varphi(y_1^*,y_2^*)=-1.489$ достигается в точке $(y_1^*,y_2^*)=(0.942,\ 0.944)$ (см. рис. 0.13).

Данная задача была решена с использованием параллельного индексного алгоритма со следующими параметрами: параметр надежности $r_1=\ldots=r_4=1.4$, точность решения $\varepsilon=10^{-3}$, плотность построения развертки $m=12$. Число процессоров и, соответственно, число разверток, варьировалось от 2 до 6. Во всех экспериментах была получена одна и та же оценка оптимума (y_1^*, y_2^*)=(0.942, 0.945).

На рис. 0.13 изображена область поиска – квадрат, где выделена допустимая область задачи и обозначены точки испытаний, полученные при применении параллельного индексного метода с шестью развертками.

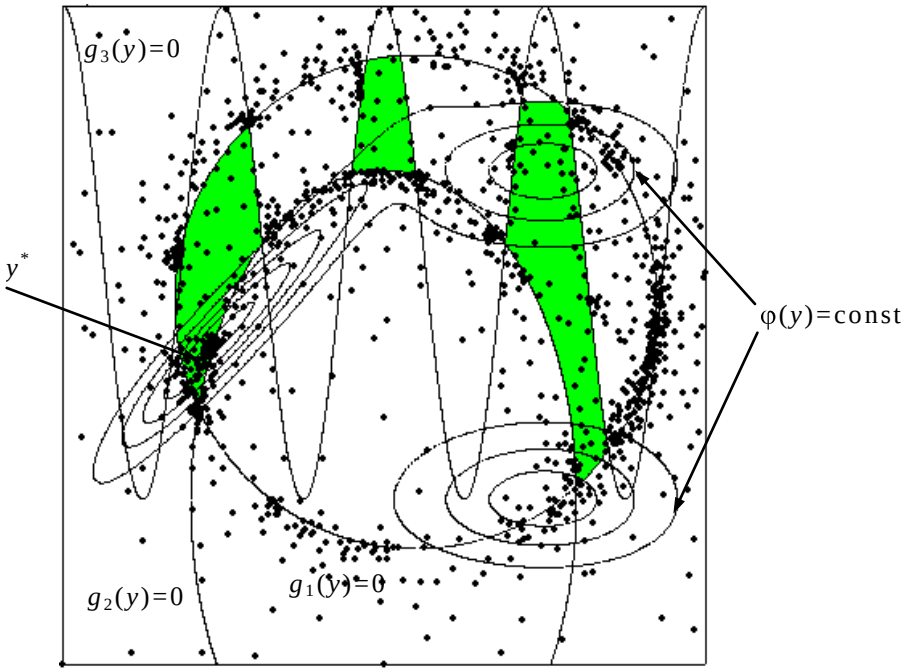

Рис. 0.13. Результаты решения задачи с использованием шести процессоров

В табл. **Ошибка! Источник ссылки не найден.**–Таблица.0.3 приведены подробные результаты экспериментов (табл. 0.1 соответствует последовательному алгоритму и приведена для сравнения).

Таблица 0.1.

Результаты решения задачи с использованием одного процессора

l	Вычислено значений функции

	g_1	g_2	g_3	$\varphi=g_4$
1	1098	623	392	152

Таблица 0.2.

Результаты решения задачи с использованием двух процессоров

l	Вычислено значений функции				
	g_0	g_1	g_2	g_3	$\varphi=g_4$
1	636	567	331	212	104
2	632	327	193	125	61
Всего	1268	894	524	337	165

Таблица 0.2.

Результаты решения задачи с использованием четырех процессоров

l	Вычислено значений функции				
	g_0	g_1	g_2	g_3	$\varphi=g_4$
1	585	508	311	206	88
2	579	262	157	86	36
3	579	293	174	119	41
4	579	284	178	116	48
Всего	2322	1347	820	527	213

Таблица.0.3.

Результаты решения задачи с использованием шести процессоров

l	Вычислено значений функции				
	g_0	g_1	g_2	g_3	$\varphi=g_4$
1	354	289	165	105	44
2	346	90	55	28	10
3	346	113	66	40	10
4	346	160	87	48	25

5	346	224	119	68	23
6	346	274	159	97	37
Всего	2084	1150	651	386	149

В табл. **Ошибка! Источник ссылки не найден.** приведена оценка ускорения, возникающего при использовании параллельного алгоритма. Ускорение определялось как отношение числа итераций (которое совпадает с числом проверок первого ограничения g_1), выполненных последовательным алгоритмом, и наибольшего числа итераций, выполненных параллельным алгоритмом на одном из процессоров. Данный способ определения ускорения актуален для задач, в которых оценка функционалов задачи требует заметных вычислительных ресурсов.

Таблица 0.5.

Результаты численных экспериментов

Число процессоров	Ускорение
1	––
2	1.93
4	2.16
6	3.8

Следует отметить, что результаты экспериментов демонстрируют (см. рис. 0.13) концентрацию точек испытаний не только в окрестности точки глобального оптимума, но и в окрестностях линий нулевого уровня ограничений (т. е. граничных точек допустимой области). Это обусловлено тем, что граничные точки допустимой области являются локально-оптимальными для индексной функции (0.13). Концентрация точек испытаний вне окрестности глобального оптимума замедляет сходимость метода при решении многомерных задач.

В рамках разработанной теории предложены способы ускорения сходимости индексного алгоритма, основанные на учете степени регулярности задачи, определяемой на основе введенного понятия ε-резервированных решений, а также путем использования адаптивно изменяемого порядка проверки ограничений. Рассмотрение модификаций индексного алгоритма выходит за рамки данного учебного материала, желающие могут их найти в работах [40,97]. Ниже мы лишь приведем результаты решения тестового примера с использованием ускоренного параллельного индексного алгоритма.

Таблица 0.6.

Результаты решения задачи с использованием шести процессоров

l	Вычислено значений функции				
	g_0	g_1	g_2	g_3	$\varphi=g_4$

1	122	102	57	36	18
2	114	48	30	18	10
3	114	44	31	21	10
4	114	49	33	15	11
5	114	56	28	11	14
6	114	66	49	32	17
Всего	692	365	228	133	80

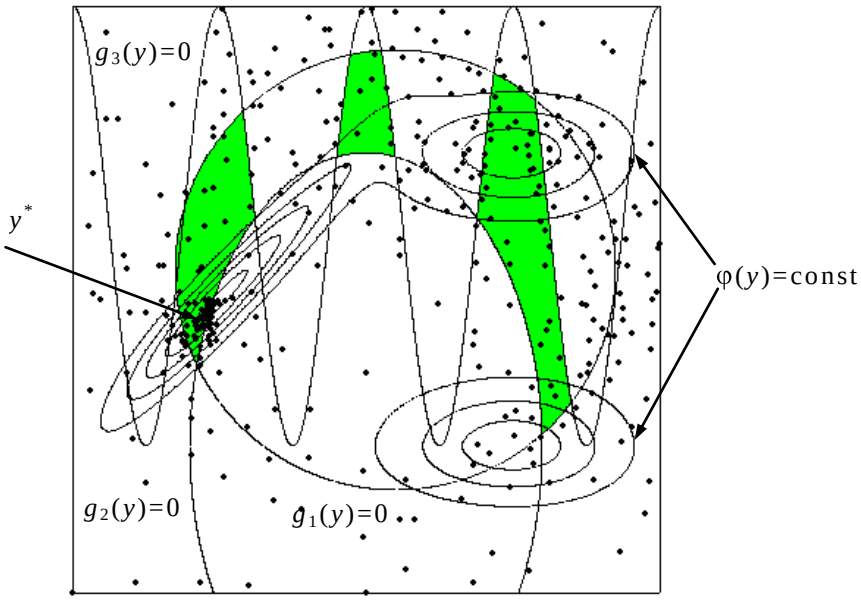

Рис. 0.14. Результаты решения задачи
с использованием шести процессоров

Приведенные на рис. 12.14 результаты решения задачи наглядно демонстрируют эффект снижения концентрации точек итераций вне допустимой области.