МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский Нижегородский государственный
университет им. Н.И. Лобачевского»

М.В. Маркина
А.В. Судакова

ПРАКТИКУМ ПО РЕШЕНИЮ ЗАДАЧ ОПТИМИЗАЦИИ В ПАКЕТЕ MATLAB

Учебно-методическое пособие

Рекомендовано методической комиссией ИИТММ
для студентов ННГУ, обучающихся по направлению подготовки
01. 04. 02 “Прикладная математика и информатика”.

Нижний Новгород
2017

УДК 519.85
ББК В22.18

Маркина М.В., Судакова А.В. ПРАКТИКУМ ПО РЕШЕНИЮ ЗАДАЧ ОПТИМИЗАЦИИ В ПАКЕТЕ MATLAB: учебно-методическое пособие. – [электронный ресурс] Нижний Новгород: Нижегородский госуниверситет, 2017. - 49с.

Рецензент: к.э.н., доцент **Ефимова Л.А.**

Настоящее пособие содержит материалы по курсу «Численные методы оптимального проектирования механических конструкций», читаемого магистрантам направления подготовки «Прикладная математика и информатика» ИИТММ. Рассматриваются разделы: одномерная и многомерная безусловная оптимизация, условная оптимизация, многокритериальная оптимизация. В каждом разделе приведен необходимый минимум теоретических сведений, разобраны практические примеры.

Целью данного пособия является знакомство с алгоритмами оптимизации, а также формирование идейных основ этих алгоритмов в пакете Matlab. Пособие предназначено для студентов и магистров, специализирующихся в области теории математического моделирования, оптимизации и математического программирования.

УДК 519.85
ББК 22.18

Содержание

1. Общая постановка и методы решения задач оптимизации

Постановка задачи оптимизации трактуется следующим образом: при заданном множестве *X* и функции *f(x)*, определенной на *X*, необходимо найти точки *extr*, т.е.:

$$f(x) \to min,\ x \in X, \quad (1)$$

где *f(x)* - целевая функция, *X*- допустимое множество.

В основном в теории оптимизации рассматриваются конечномерные задачи оптимизации, в которых допустимое множество *X* лежит в евклидовом пространстве $R^n (x \in R^n)$.

Точка $x^* \in X$, являющаяся решением задачи (1), может быть точкой глобального или локального минимума.

Определение.

Точка $x^* \in X$ называется

1) точкой глобального минимума функции *f(x)* на множестве *X* или глобальным решением задачи (1), если

$$f(x^*) \le f(x)\ , x \in X, \quad (2)$$

2) точкой локального минимума функции *f(x)* на множестве *X* или локальным решением задачи (1), если

$$\exists \varepsilon > 0 \text{ такое, что для } \forall x \in X \cap U_\varepsilon(x^*)\ \ f(x^*) \le f(x) \quad (3)$$

где

$U_\varepsilon(x^*) = \{x \in R^n |\ ||x - x^*|| \le \varepsilon\}$ шар радиусом $\varepsilon>0$ с центром в точке *x*.

$(x*) \le fx\ , x \in X$ или в *Ошибка! Источник ссылки не найден.* выполняется как строгое при $x \ne x^*$, то x^* - точка строгого минимума (строгое решение) в глобальном или локальном смысле.

Множество всех точек глобального минимума *f(x)* на *X*, обозначают через

$$arg\,min_{x \in X} f(x) = \{x^* \in X \mid f(x^*) = f^*\}, \quad (4)$$

где *f* - минимальное значение функции *f(x)* на множестве *X*, $\arg\min f(x)$ - произвольная точка из множества $\arg\min_{x \in X} f(x)$.

Решения задач (1), **Ошибка! Источник ссылки не найден.**, т.е. точки минимума и максимума функции *f(x)* на множестве *X*, называются также ***точками экстремума***.

Поиск экстремума функции одной переменной.

При решении задач оптимизации выделяют локальный (с указанием границ) и глобальный максимум (минимум). Для нахождения локального минимума функции одной переменной в пакете Matlab возможно использовать функцию [x, y]=fminbnd(name, a, b, options), где:

- name - имя функции,
- a, b - границы поиска,
- options – параметры, управляющие ходом решения,
- x, y - координаты точки минимума.

К примеру, рассмотрим функцию $y(x)=x^4-0.5x^3-28x^2+140$ на интервале [4,2].

[x,y]=fminbnd(@ext,-4,-2),

где @ext - m-функция.

Поиск экстремума функции нескольких переменных.

Минимум функции нескольких переменных $z=f(x_1, x_2, \ldots, x_n)$ осуществляет встроенная функция [X,Z]=fminsearch(name, x0, options) где:

- name m-функция, вычисляющая z=f(x1, x2, …, xn),
- x0 вектор из n элементов, содержащих координаты точки начального приближения,
- options параметры управляющие ходом решения,
- X вектор из n элементов, содержащий значения переменных, при которых функция z=f(x1, x2, …, xn) минимальна,
- Z минимальное значение функции.

Пример использования данной функции в Matlab:

[X, Z]=fminsearch(@extr, [3 2]);

где @extr обращение к m-функции, описывающий целевую функцию

$z=x_1^2+x_2^2-6x_2-2x_1+11$,

где заданы начальные приближения [3 2] для x_1 и x_2 соответственно, в результате которого вычисляются значения переменных в точке минимума и значение функции в extr.

Таблица 1.

Основные функции MatLab для решения задач оптимизации

Функция	**Описание**	**Алгоритм**
Безусловная оптимизация		
fmin('F',x1,x2,…,options,P1,P2)	Поиск локального минимума функции одной переменной. 'F'-имя файл-функции; x1,x2 – левая и правая границы интервала, на котором ищется минимум; options – параметры управления процессом решения; P1,P2 – независимые параметры	Метод золотого сечения и параболической интерполяции
fminbnd('F',x1,x2,…,options,P1,P2)	Поиск локального минимума функции одной переменной на заданном интервале, применение аналогично fmin	Метод золотого сечения и параболической интерполяции
fminsearch('F',X0,…,options,P1,P2)	Поиск локального минимума функции нескольких переменных, применение аналогично fmins	Метод симплексного поиска

fminunc(‘F’,X0, …,options,P1,P2)	Поиск минимума нелинейной функции нескольких переменных без ограничения на переменные, применение аналогично fmins	Квазиньютоновские методы
Условная оптимизация		
linprog(f,A,b,…, Aeq,Beq,LB,…,UB,X0,options)	Решение задач линейного программирования. f – вектор коэффициентов; A, b, Aeq, Beq – вектора коэффициентов нелинейных и линейных ограничений; LB, UB – вектора верхних и нижних границ изменения переменных; X0 – вектор начальных приближений; options – параметры управления процессом решения	Метод симплексного поиска
quadprog(H,f,A,b,…,Aeq,Beq,LB,UB, …, X0,options)	Решение задач квадратичного программирования. H – вектор соответствующих коэффициентов; остальные параметры – аналогично linprog	Метод предварительно сопряженных градиентов(PCG)
fmincon(‘fun’,X0,A,B,…,Aeq,Beq,LB,UB,…,’nonlcon’,options,…,P1,P2)	Решение задач нелинейного программирования. ‘fun’ – имя файл-функции, содержащей целевую функцию; ’nonlcon’ – имя файл-функции, содержащей нелинейные ограничения; остальные параметры – аналогично linprog	Метод Ньютона и метод последовательного квадратичного программирования (SQP)
fminimax(‘fun’,X0,…,A,B,Aeq,Beq,,LB,UB,…,’nonlcon’,options,…,P1,P2)	Решение минимаксной задачи, применение аналогично fmincon	Метод последовательного квадратичного программирования (SQP)
fgoalattain(‘fun’, X0,…,g,w,A,B,Aeq,B,eq,…,LB, UB,’nonlcon’,…, options, P1,P2)	Решение задачи о достижении границы. Решается задача вида $F(x) - w\gamma \le g$. Остальные параметры аналогичны fmincon.	Метод последовательного квадратичного программирования (SQP)

2. Методы безусловной оптимизации

Рассматривается задача

$$f(x) \to extr,\ x \in R^n \qquad (5)$$

Метод поиска безусловного экстремума основывается на следующих утверждениях:

Пусть функция $f(x)$ дифференцируема в точке $x^* \in R^n$. Тогда если x^* - локальное решение задачи (5), то

$$grad\, f(x^*) = 0 \qquad (6)$$

Пусть функция $f(x)$ дважды дифференцируема в точке $x^* \in R^n$. Тогда

а) если x^* - точка локального минимума в задаче (5), то матрица Гессе $H(x^*)$ неотрицательно определена, т.е. $\forall p \in R^n$ выполняется неравенство $(H(x^*)p,p) \geq 0$;

б) если x^* - точка локального минимума в задаче (5), то матрица $H(x^*)$ неположительно определена, т.е. $\forall p \in R^n$ выполняется неравенство $(H(x^*)p,p) \leq 0$.

Пусть функция $f(x)$ дважды дифференцируема в точке $x^* \in R^n$ и

$\text{grad}\, f(x^*) = 0$.

Тогда

а) если матрица $H(x^*)$ положительно определена, т.е. $\forall p \in R^n$, $p \neq 0$, $(H(x^*)p,p) > 0$, то x^* - точка строгого локального минимума функции $f(x)$ на R^n;

б) если матрица $H(x^*)$ отрицательно определена, т.е. $\forall p \in R$, $p \neq 0$, $(H(x^*)p,p) < 0$, то x^* - точка строгого локального максимума функции $f(x)$ на R^n.

Если $\text{grad}\, f(x^*) = 0$, то x^* называется стационарной точкой. Для выпуклой (вогнутой) на R^n функции стационарные точки являются точками ее глобального минимума (максимума). Строго выпуклые (вогнутые) функции имеют единственный глобальный минимум (максимум).

Критерий выпуклости функции. Дважды непрерывно дифференцируемая на выпуклом множестве X с непустой внутренностью функция является выпуклой (вогнутой) на этом множестве в том и только том случае, когда матрица Гессе $H(x^*)$ неотрицательно (не положительно) определена для всех $x \in X$.

При исследовании на знакоопределенность матрицы вторых производных функции рекомендуется применять критерий Сильвестра или анализ собственных значений матрицы.

Схема поиска безусловных экстремумов функции:

Составить и решить систему алгебраических уравнений (6).

В стационарных точках (точках, являющихся решением системы (6)) исследовать на знакоопределенность матрицу вторых производных; точки, в которых $H(x) > 0$, являются точками глобального минимума; стационарные точки, в которых $H(x) < 0$, являются точками глобального максимума.

Исходя из вида исследуемой функции, проанализировать стационарные точки, в которых матрица вторых производных не является строго знакоопределенной.

Найденные точки локального экстремума исследуются на глобальный экстремум (если это возможно). В частности, если матрица Гессе неотрицательно (не положительно) определена на всем пространстве E^n, то все стационарные точки функции являются точками глобального минимума (максимума).

2.1. Итерационные методы безусловной оптимизации

Численное решение задачи минимизации (5), как правило, связано с построением минимизирующей последовательности точек $x^0, x^1, x^2, \ldots, x^n, \ldots$, обладающих свойством

$$f(x^k) < f(x^{k-1}),\ k=0,1,\ldots \qquad (7)$$

Общее правило построения минимизирующей последовательности имеет вид

$$x^{k+1} = x^k + t_k d^k,\ k=0,1,\ldots,$$

где x^0 - начальная точка поиска; d^k - приемлемое направление перехода из точки x^k в точку x^{k+1}, которое обеспечивает выполнение условий (7) и называется направлением спуска; t_k - величина шага. Начальная точка поиска задается исходя из физического содержания решаемой задачи и априорных данных о существовании и положении точек экстремума.

При решении вопроса о выборе численного метода рекомендуется оценить поведение линий уровня целевой функции в окрестностях предполагаемой точки экстремума. Число $m = L/l$, где L и l - максимальное и минимальное собственные значения гессиана функции f в предполагаемой точке экстремума x^0 (характеризующее разброс собственных значений оператора $f(x)$), называется *числом обусловленности* гессиана функции f в точке x^0. Если $m >> 1$, то функция f называется *плохо обусловленной* или *овражной*. *Овражность*, то есть вытянутость линий уровня вдоль одного направления, приводит к тому, что градиентные методы поиска экстремума функции сходятся медленно.

В зависимости от наивысшего порядка частных производных функции $f(x)$, используемых для формирования d^k и t_k, численные методы принято делить на три группы:

Методы нулевого порядка, использующие информацию только о значениях функции $f(x)$ (методы деформируемого многогранника, конфигураций). Эти методы могут применяться в тех случаях, когда функция задана неявно или не задана аналитически, но известен ряд значений функции или эти значения вычисляются непосредственно в ходе реализации алгоритма. Они также могут быть полезны в случаях, когда производные функции могут быть заданы аналитически, но их выражения очень громоздки.

Методы первого порядка, использующие информацию о значениях самой функции *f (x)* и ее первых производных (методы наискорейшего градиентного спуска, дробления шага, Гаусса-Зейделя, Флетчера-Ривса).

Методы второго порядка, использующие, кроме того, и информацию о вторых производных функции *f (x) (*метод Ньютона и его модификации).

2.1.1. Метод прямого поиска (метод Хука-Дживса)

Следует выделить два этапа метода конфигураций:

1) исследование с циклическим изменением переменных

2) ускорение поиска по образцам.

Исследующий поиск начинается в точке x^0, называемой старым базисом. Направления поиска - координатные направления. По каждому направлению поочередно с шагом $+t_0$ ($-t_0$) проверяется выполнение условия (6) и в качестве нового базиса берется точка с координатами, полученными в результате удачных шагов из начальной точки по каждому направлению.

Направление от старого базиса к новому задает направление ускорения поиска: в качестве следующей точки минимизирующей последовательности проверяется точка $y^1=x^0+\lambda\ (x^1-x^0)$. Здесь λ - ускоряющий множитель, задаваемый пользователем. Если полученная точка является удачной, то она берется в качестве следующей точки для исследования. В противном случае исследование ведется из точки x^1.

2.1.2. Метод деформируемого многогранника (метод Нелдера—Мида)

При решении задачи поиска минимума функции *f (x)* методом Нелдера-Мида строится последовательность множеств из $n+1$ точек, которые являются вершинами выпуклого многогранника. На каждом последующем $k+1$-м шаге из системы точек $x^i\ (k),\ i = \overline{1, n+1}$, полученной на k-м шаге, выводится точка $x^h\ (k)$, в которой функция *f (x)* имеет наибольшее значение (худшая точка). Вместо $x^h\ (k)$ в систему вводится новая точка, выбираемая на отрезке прямой, проходящей через худшую точку и центр тяжести оставшихся n вершин многогранника:

$x^{n+2}=\frac{1}{n}\left[\sum_{j=1}^{n+1} x^j - x^h\right]$ - центр тяжести;

$x^{n+3}= x^{n+2}+\alpha\ (x^{n+2} - x^h)$ - новая точка (“растянутое” отражение наихудшей вершины).

2.2. Градиентные методы безусловной оптимизации

2.2.1. Метод наискорейшего спуска

Как и в предыдущем методе, точки релаксационной последовательности $\{x^k\}$, k=0,1,… вычисляются по правилу (7). Точка x^0 задается пользователем; величина шага t_k определяется из условия минимума одномерной функции $\varphi(t_k) = f(x^k - t_k \operatorname{grad} f(x^k))$. Задача минимизации функции $\varphi(t_k)$ может быть решена с использованием необходимого условия минимума $\frac{d\varphi}{dt^k}=0$ с последующей проверкой достаточного условия минимума $\frac{d^2\varphi}{dt_k^2}>0$ или с использованием численных методов.

2.2.2. Метод сопряженных направлений (Флетчера - Ривса)

В данном методе используются свойства векторов, сопряженных относительно некоторой матрицы.

Определение. Векторы p и q называются сопряженными относительно матрицы Q, если выполняется равенство pQq=0.

Точки релаксационной последовательности $\{x^k\}$, k=0,1,… вычисляются по правилу

$x^{k+1}=x^k-t_kd^k$, k=0,1,…;

$d^k = -\operatorname{grad} f(x^k) + \beta_{k-1} d^{k-1}$; (8)

$d^0 = -\operatorname{grad} f(x^0)$;

$\beta_{k-1}=\| \operatorname{grad} f(x^k) \|^2 / \| \operatorname{grad} f(x^{k-1}) \|^2$.

Точка x^0 задается пользователем; величина шага t_k определяется из условия минимума функции $\varphi(t) = f(x^k - td^k)$. Задача минимизации одномерной функции $\varphi(t_k)$ может быть решена с использованием необходимого условия минимума $\frac{d\varphi}{dt}=0$ с последующей проверкой достаточного условия минимума $\frac{d^2\varphi}{dt^2}>0$ или с использованием численных методов. Коэффициент β_{k-1} вычисляется из условия сопряженности направлений d^k и d^{k-1}.

2.3. Метод Ньютона

Строится последовательность точек $\{x^k\}$, k=0,1,…, таких, что $f(x^k) < f(x^{k-1})$, k=0,1,… Точки последовательности $\{x^k\}$ вычисляются по правилу $x^{k+1}=x^k+d^k$, k=0,1,… Точка x^0 задается пользователем с учетом знакопостоянства и невырожденности матрицы Гессе в задаваемой начальной точке и близости

выбранной точки к предполагаемой точке минимума. Направление спуска определяется для каждого значения k по формуле $d^k = -H^{-1}(x^k)$ grad $f(x^k)$, где H - матрица Гессе.

2.4. Практическое решение задач безусловной оптимизации

Функция fminbnd

fminbnd – функция скалярной нелинейной минимизации с ограничениями вида $x_1 < x < x_2$. Алгоритм базируется на методе золотого сечения и квадратичной (параболической) интерполяции. Запись функции:

```
x = fminbnd(fun,x1,x2,options,P1,P2,...)
[x,fval,exitflag,output] = fminbnd(...)
```

Аргументы и возвращаемые величины практически аналогичны рассмотренным для предыдущей функции за тем исключением, что число опций здесь меньше: возможны только опции Display, MaxFunEvals, MaIter и TolX.

Пример 1. Необходимо найти точку минимума функции sin(x) на интервале (0, 2π)

```
>> x = fminbnd('sin',0,2*pi)
x =4.7124
```

Значение функции при этом

```
>> y = sin(x)
y =-1.0000
```

Пример 2. Для минимизации функции $f(x) = (x-3)^2 - 1$ на интервале (0, 5) сначала необходимо представить данную функцию и виде m-файла:

```
function f = myfun(x)
f =(x-3).^2-1;
```

Для решения задачи возможно использовать функцию fminbnd:

```
>> x = fminbnd('myfun',0,5)  или x = fminbnd(@myfun,0,5)
x =3
```

Функция fminsearch

Функция fmlnsearch позволяет найти минимум функции нескольких переменных без ограничений, то есть решение задачи безусловной оптимизации с использованием симплексного метода.

Формы записи, аргументы и возвращаемые величины аналогичны рассмотренным ранее:

```
x = fminsearch(fun,x0,options,P1,P2,...)
[x,fval,exitflag,output] = fminsearch(...)
```

fminsearch находит минимум скалярной функции нескольких переменных, стартуя с некоторой начальной точки и ,таким образом, задача относится к нелинейной оптимизации без ограничений

x = fminsearch(fun,x0,options,P1,P2,...) передает зависимые от задачи параметры P1, P2 и т.д. непосредственно в функцию fun. Используется опция = [] как структурный ноль, если опции не определены.

[x,fval,exitflag,output] = fminsearch(...) возвращает структурный выход с информацией об оптимизации.

Пример. Ниже демонстрируется нахождение минимума функции одного аргумента $f(x) = \sin(x) + 3$:

```
>> f = inline('sin(x)+3');
>> x = fminsearch(f,2)
x = 4.7124
```

Функция fminunc

Функция fminunc предназначена для тех же целей, что и предыдущая функция, но, в отличие от последней, имеет большее число представлений:

```
x = fminunc(fun,x0,options,P1,P2,...)
 [x,fval,exitflag,output,grad,hessian] = fminunc(...)
```

Набор опций в данном случае такой же, как и у функции fmincon, то есть здесь возможно использование как алгоритма средней размерности, так и алгоритма большой размерности.

Пример 1. Найдем минимум функции $f(x) = 3x_1^2 + 2x_1x_2 + x_2^2$.

Способ 1. Создадим m-файл

```
function f = myfun(x)
f = 3*x(1)^2+2*x(1)*x(2)+x(2)^2;
```

Затем используем рассматриваемую функцию при начальном значении [1, 1]. Результат возвращается буквально после пары итераций:

```
>> x0 = [1,1];
>> [x,fval] = fminunc('myfun',x0)
x =l.0e-008 *
-0.8356    0.2899
fval =1.6943e-016
```

Как видно, найденное значение достаточно близко к истинной точке минимума [0, 0].

Способ 2. Функция также записывается в m-файл, но с заданием информации о градиенте целевой функции.

```
function [f,g] = myfun(x)
f = 3*x(1)^2+2*x(1)*x(2)+x(2)^2;
if nargout > 1  % проверка количества аргументов функции
g(1) = 6*x(1)+2*x(2);
g(2) = 2*x(1)+2*x(2);
end;
```

Решение задачи:

```
>> % Разрешение использования градиента пользователя
>> options = optimset('GradObj','on');
```

```
>> x0 = [1,1]; % Стартовое значение
>> [x,fval] = fminunc('myfun',x0,options) % Нахождение решения
x =1.0e-015 *
0.1110    0.4441
fval =3.3280e-031
```

Пример 2. Найдем минимум функции $f(x) = \sin(x) + 3$:

```
>> f = inline('sin(x)+3');
>> x = fminunc(f,4)
x =4.7124
```

Пример 3. Рассмотрим задачу нахождения значении переменных x_1 и x_2, обеспечивающих решение задачи минимизации

$$\min_x f(x) = e^{x_1}(4{x_1}^2 + 2{x_2}^2 + 4x_1x_2 + 1)$$

Нахождение решения производится в соответствии со следующими этапами.

Этап 1. Составление m-файла (с именем objfun), реализующего вычисление значения целевой функции:

```
function f = objfun(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
```

Этап 2. Составление программы с использованием подходящей функции пакета (в данном случае – функции fminunc):

```
>> % стартовое значение
>> x0 = [-1,1];
>> % задание опции использования алгоритма средней размерности
>> options = optimset('LargeScale','off');
>> % Поиск решения
>> [x,fval,exitflag,output] = fminunc('objfun',x0,options)
```

Выполнение программы приведет к следующему результату:

```
Optimization terminated successfully:
Current search direction is a descent direction, and magnitude of directional derivative in search direction less than 2*options.TolFun
x =0.5000      -1.0000
fval =1.3031e-010
exitflag =1
output =iterations: 7
funcCount: 40
stepsize: 1
firstorderopt: 8.1995e-004
algorithm: 'medium-scale: Quasi-Newton line search'
```

Значения x = [0.5000 -1.0000] и fval = 1.3031e-010 – искомое решение задачи. Значение exitflag = 1 дает информацию, что найдена точка минимума (возможно, локального).

Выходная структура (информация о результатах оптимизации) определяется идентификатором output:

число выполненных итераций (iterations): 7;

число вычислений функции (funcCount): 40;

шаг поиска (stepsize): 1;

степень оптимальности найденного решения (firstorderopt) – норма вектора-градиента в точке найденного решения: 8.1995e-004;

использованный алгоритм (algorithm): квазиньютоновский с одномерной оптимизацией ('medium-scale: Quasi-Newton line search'), относящийся к числу алгоритмов средней размерности.

Пример 4. Пусть требуется найти минимум функции вида

$$f(x) = \sum_{i=1}^{n-1} (x_i^2)^{x_i^2+1} + (x_{i+1}^2)^{x_i^2+1}, n = 1000.$$

В соответствии с таблицей 1 данная задача является задачей безусловной оптимизации, и ее целесообразно решать с помощью функции fminunc.

Этап 1. Составление m-файла (с именем brownfgh) для вычислений значений целевой функции, ее градиента и разреженной трехдиагональной матрицы Гессе:

```
function [f,g,H] = brownfgh(x)
% вычисление функции
n = length(x); y = zeros(n,1);
i = 1:(n-1);
y(i) = (x(i).^2).^(x(i+1).^2+1)+(x(i+1).^2).^(x(i).^2+1);
f = sum(y);
% вычисление градиента
i=1:(n-1); g = zeros(n,1);
g(i)= 2*(x(i+1).^2+1).*x(i).*((x(i).^2).^(x(i+1).^2))+
2*x(i).*((x(i+1).^2).^(x(i).^2+1)).*log(x(i+1).^2);
g(i+1)=g(i+1)+2*x(i+1).*((x(i).^2).^(x(i+1).^2+1)).*log(x(i).^2)+...
2*(x(i).^2+1).*x(i+1).*((x(i+1).^2).^(x(i).^2));
 % вычисление (разреженной, симметричной) матрицы Гессе
v = zeros(n,1);
i = 1:(n-1);
v(i) = 2*(x(i+1).^2+1).*((x(i).^2).^(x(i+1).^2)+ ...
4*(x(i+1).^2+1).*(x(i+1).^2).*(x(i).^2).*...
((x(i).^2).^((x(i+1).^2)-1))+2*((x(i+1).^2).^(x(i).^2)).*(log(x(i+1).^2)));
v(i) = v(i)+4*(x(i).^2).* ((x(i+1).^2).^ (x(i).^2 +1) ).*((log(x(i+1).^2)).^2);
v(i+1) = v(i+1)+2*(x(i).^2).^(x(i+1).^2+1).*log(x(i).^2)+...
4*(x(i+1).^2).*((x(i).^2).^(x(i+1).^2+1)).*((log(x(i).^2)).^2)+...
2*(x(i).^2+1).*((x(i+1).^2).^(x(i).^2));
v(i+1) = v(i+1)+4*(x(i).^2+1).*(x(i+1).^2).*(x(i).^2).*((x(i+1).^2).^(x(i).^2-1));
v0 = v;
v = zeros(n-1,1);
```

```
v(i) = 4*x(i+1).*x(i).*((x(i).^2).^(x(i+1).^2))+
4*x(i+1).*(x(i+1).^2+1).*x(i).*((x(i).^2).^(x(i+1).^2)).*log(x(i).^2);
  v(i) = v(i)+4*x(i+1).*x(i).*(x(i+1).^2).^(x(i).^2).* log(x(i+1).^2);
  v(i) = v(i)+4*x(i).*((x(i+1).^2).^(x(i).^2)).*x(i+1);
  v1 = v;
  i = [(1:n)';(1:(n-1))'];
  j = [(1:n)';(2:n)'];
  s = [v0;2*v1];
  H = sparse(i,j,s,n,n);
  H = (H+H')/2;
```

Этап 2. Составление оптимизирующей программы:

```
>> xstart = -ones(n,1);
>> xstart(2:2:n,1) = 1;
>> % задание использования градиента и гессиана пользователя
>> options = optimset('GradObj','on','Hessian','on');
>> % поиск решения
>> [x,fval,exitflag,output] = fminunc('brownfgh', xstart, options);
```

Результаты вычислений свидетельствуют о получении корректного решения:

Optimization terminated successfully:

First-order optimality less than OPTIONS.TolFun. and no negative/zero curvature

```
>> exitflag
exitflag =1
>> fval
fval =2.8709e-017
>> output.iterations
ans =8
>> output.cgiterations
ans =7
>> output.firstorderopt
ans =4.7948e-010
```

Пример 5. Следующий пример иллюстрирует решение задачи минимизации большой размерности с заданием пользовательского градиента, но с приближенным вычислением гессиана, при котором для ускорения расчетов использован его разреженный образ. Данный образ должен быть заранее подготовлен в виде некоторой разреженной матрицы (в рассматриваемом примере это матрица Hstr, сохраненная в файле brownhstr.mat).

Необходимо также (с помощью функции optimset) установить значение 'on' для опции GradObj, поскольку градиент задается пользователем (вычисляется в m-файле).

Этап 1. Создание m-файла (с именем brownfg) для расчета значений минимизируемой функции и ее градиента.

```
function [f,g] = brownfg(x,dummy)
% вычисление функции
n=length(x); y=zeros(n,1);
i=1:(n-1);
y(i)=(x(i).^2).^(x(i+1).^2+1) + (x(i+1).^2).^(x(i).^2+1);
  f=sum(y);
% вычисление градиента
    i=1:(n-1); g = zeros(n,1);
    g(i) = 2*(x(i+1).^2+1).*x(i).*((x(i).^2).^(x(i+1).^2))+
          2*x(i).*((x(i+1).^2).^(x(i).^2+1)).*log(x(i+1).^2);
    g(i+1) = g(i+1) +2*x(i+1).*((x(i).^2).^(x(i+1).^2+1)).* ...
          log(x(i).^2) + 2*(x(i).^2+1).*x(i+1).*((x(i+1).^2).^(x(i).^2));
```

Этап 2. Создание программы поиска решения:

```
>> fun = 'brownfg'; % задание имени функции (M-файла)
>> load brownhstr % загрузка разреженного образа гессиана
>> % графическое представление разреженной матрицы
>> spy(Hstr)
>> n = 1000;
>> xstart = -ones(n,1);
>> xstart(2:2:n,1) = 1;
>> % задание опций
>> options = optimset('GradObj','on','HessPattern',Hstr);
>> [x,fval,exitflag,output] = fminunc(fun,xstart,options);
```

Результаты расчетов:

```
Optimization terminated successfully:
```

First-order optimality less than OPTIONS.TolFun. and no negative/zero curvature detected

```
>> exitflag
exitflag =1
>> fval
fval =7.4739e-017
>> output.iterations
ans =8
>> output.firstorderopt
ans =7.9822e-010
```

Данные результаты говорят об успешном решении задачи. Команда spy(Hstr) позволяет графически представить структуру разреженной матрицы.

3. Методы условной оптимизации

Общая задача нахождения экстремума функции при наличии ограничений - равенств и ограничений – неравенств записывается в следующем виде:

$$f(x) \to extr, \qquad (9)$$

$x \in X = \{x \in E^n : g_i(x) \le 0,\ i=1,2,\ldots,r;\ g_i(x) = 0,\ i=r+1,\ \ldots,\ m,\ m-r<n\}$,

где среди функций $f(x)$ и $g_i(x)$ могут быть нелинейные.

Активные ограничения - неравенства в точке x^* — это ограничения, которые выполняются в данной точке в виде равенства.

Пассивные ограничения - неравенства в точке x^* — это ограничения, которые выполняются в данной точке в виде строгого неравенства.

Если градиенты активных ограничений-неравенств и ограничений-равенств в точке x^* линейно независимы, то говорят, что в оптимальной точке выполнено условие регулярности.

Обобщенная функция Лагранжа для задачи со смешанными ограничениями задается как

$$L(x,\lambda_0,\lambda) = \lambda_0 f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) \qquad (10)$$

При выполнении условия регулярности $\lambda_0 \neq 0$ и можно положить этот коэффициент равным 1.

Теорема Куна - Таккера (дифференциальная форма необходимого условия минимума). Пусть точка x^* - точка локального минимума в задаче математического программирования, функции $f, g_{r+1},\ldots,g_m$ дважды непрерывно дифференцируемы в точке x, функции $g_1,\ldots,g_r$ дважды непрерывно дифференцируемы в некоторой окрестности точки x. Тогда существует число λ_0^* и вектор λ^* такие, что выполняются следующие условия:

условие стационарности обобщенной функции Лагранжа по x:

$\mathrm{grad}_x L(x^*,\ \lambda_0^*, \lambda^*) = 0$;

условие нетривиальности:

$\lambda_0^{*\,2} + \lambda^{*\,2} > 0$, т.е. хотя бы один из множителей Лагранжа отличен от нуля;

условие неотрицательности:

$\lambda_0^* \ge 0,\ \lambda_i^* \ge 0,\ i=1,\ \ldots,\ r$,

т.е. множители Лагранжа, соответствующие целевой функции и ограничениям - неравенствам, неотрицательны;

условия дополняющей нежесткости:

$g_i(x^*) = 0,\ i=1,\ 2,\ \ldots,\ r$.

Если при этом выполнено условие регулярности, то для выпуклых функций $f,\ g_{r+1},\ldots,\ g_m$ и линейных функций $g_1,\ldots,\ g_r$ условия теоремы Куна - Таккера являются одновременно необходимыми и достаточными условиями глобального минимума.

Достаточное условие минимума первого порядка.

Пусть имеется точка (x^*, λ^*), удовлетворяющая условию стационарности обобщенной функции Лагранжа по x при $\lambda_0^* \neq 0$, суммарное число активных ограничений-неравенств в точке x^* и ограничений-равенств совпадает с числом переменных n. Если $\lambda_j^* > 0$ для всех активных ограничений $g_j(x)$, то точка x^* - точка условного локального минимума в задаче.

Достаточное условие минимума второго порядка.

Пусть имеется точка (x^*, λ^*), удовлетворяющая условию стационарности обобщенной функции Лагранжа по x при $\lambda_0^* \neq 0$. Если в этой точке $d^2L(x^*, \lambda^*) > 0$ для всех ненулевых dx таких, что для активных в точке x^* ограничений-неравенств $dg_j(x^*) = 0$, $\lambda_j^* > 0$ и $dg_j(x^*) \leq 0$, $\lambda_j^* = 0$, то точка x^* является точкой локального минимума.

Общая схема решения задачи условной минимизации функции:

Составляется обобщенная функция Лагранжа. Выписываются необходимые условия минимума, сформулированные в теореме Куна - Таккера. К этим условиям добавляются ограничения, задающие допустимое множество X. Полученная система алгебраических уравнений и неравенств используется для поиска условно-стационарных (подозрительных на экстремум) точек. Целесообразно проанализировать отдельно случаи $\lambda_0=0$ и $\lambda_0=1$ (или λ_0 - любое положительное число). Однако если выполнено одно из условий регулярности, то вариант $\lambda_0=0$ рассматривать не надо.

В найденных точках проверяется выполнение достаточных условий минимума и проводится анализ на глобальный экстремум.

Множители Лагранжа могут быть использованы для оценивания влияния малых изменений правых частей ограничений на оптимальное решение задачи нелинейного программирования. Пусть $x^* = x^*(b)$ - решение ЗНП

$$f(x) \to min \qquad (11)$$

$$x \in X = \{x \in E^n : g_i(x) \leq b_i,\ i=1,2,\ldots,m;\ x \geq 0\}$$

при некотором векторе b свободных членов в ограничениях - неравенствах, а $v(b)$ соответственно значение целевой функции при этом решении ЗНП, т.е. $v(b) = f(x^*)$. Тогда справедлива следующая оценка изменения целевой функции: $\Delta v = f(b+\Delta b) - f(b)$ при изменении вектора b на некоторый малый вектор-приращение Δb:

$$\Delta f \approx (\Delta b, \lambda^*) \qquad (12)$$

где λ^* - вектор множителей Лагранжа, соответствующий решению $x^*(b)$.

3.1. Прямые методы условной оптимизации Основные определения

Задача условной оптимизации заключается в поиске минимального или максимального значения скалярной функции $f(x)$ n-мерного векторного

аргументах (в дальнейшем без ограничения общности будут рассматриваться задачи поиска минимального значения функции):

$f(x)$ -> min

при ограничениях:

$g_i(x) = 0, i = 1, ..., k;$

$h_j(x) \le 0, j = 1, .., m;$

$a \le x \le b.$

Здесь x, a, b — векторы-столбцы:

$$x = \begin{vmatrix} x_1 \\ \dots \\ x_n \end{vmatrix}, \quad a = \begin{vmatrix} a_1 \\ \dots \\ a_n \end{vmatrix}, \quad b = \begin{vmatrix} b_1 \\ \dots \\ b_n \end{vmatrix},$$

Оптимизируемую функцию $f(x)$ называют *целевой функцией*. Каждая точка x в n-мерном пространстве переменных $x_1, ..., x,$ в которой выполняются ограничения задачи, называется *допустимой точкой задачи.* Множество всех допустимых точек называется *допустимой областью* G . Будем считать, что это множество не пусто. *Решением задачи* считается допустимая точка x^*, в которой целеваяфункция $f(x)$ достигает своего минимального значения. Вектор x^* называют *оптимальным.* Если целевая функция $f(x)$ и ограничения задачи представляют собой линейные функции независимых переменных $x_1, ..., x_n,$ то соответствующая задача является *задачей линейного программировании,* в противном случае - *задачей нелинейного программирования.* В дальнейшем будем полагать, что функции $f(x), g(x), i = 1, ..., k , h_j(x), j = 1, ..., m,$ - непрерывные и дифференцируемые.

В общем случае численные методы решения задач нелинейного программирования можно разделить на прямые и непрямые. *Прямые методы* оперируют непосредственно с исходными задачами оптимизации и генерируют последовательности точек $\{x[k]\}$, таких, что $f(x[k+1]) < f(x[k])$. В силу этого такие методы часто называют *методами спуска.* Математически переход на некотором k-м шаге $(k. = 0, 1, 2, ...)$ от точки $x[k]$ к точке x[k+1] можно записать в следующем виде:

$x[k+l] = x[k] + a_k p[k],$

где $p[k]$ — вектор, определяющий направление спуска; a_k — длина шага вдоль данного направления. При этом в одних алгоритмах прямых методов точки $x[k]$ выбираются так, чтобы для них выполнялись все ограничения задачи, в других эти ограничения могут нарушаться на некоторых или всех итерациях. Таким образом, в прямых методах при выборе направления спуска ограничения, определяющие допустимую область $G,$ учитываются в явном виде.

Непрямые методы сводят исходную задачу нелинейного программирования к последовательности задач безусловной оптимизации некоторых вспомогательных функций. При этих методах ограничения исходной задачи учитываются в неявном виде.

3.1.1. Метод проекции градиента

Рассмотрим данный метод применительно к задаче оптимизации с ограничениями-неравенствами. В качестве начальной выбирается некоторая точка допустимой области *G.* Если *x*[0] - внутренняя точка множества *G* (рис.1), то рассматриваемый метод является обычным градиентным методом:

$x[k+1] = x[k] - a_k f'(x[k]), k = 0, 1, 2, ...,$

где $f(x[k]) = \left(\frac{\partial([k])}{\partial x_1}, \dots, \frac{\partial([k])}{\partial x_k}\right)^T$ градиент целевой функции *f(x)* в точке *x*[*k*].

После выхода на границу области *G* в некоторой граничной точке *x*[*k*] , *k* = 0, 1, 2,..., движение в направлении антиградиента *-f'(x*[*k*]*)* может вывести за пределы допустимого множества (рис.1). Поэтому антиградиент проецируется на линейное многообразие *M,* аппроксимирующее участок границы в окрестности точки *x*[*k*]. Двигаясь в направлении проекции вектора *-f'(x*[*k*]) на многообразие *M,* отыскивают новую точку *x*[*k*+1], в которой *f(x*[*k*+1]*)* < *f(x*[*k*]*)*, принимают *x*[*k*+1] за исходное приближение и продолжают процесс. Проведем более подробный анализ данной процедуры.

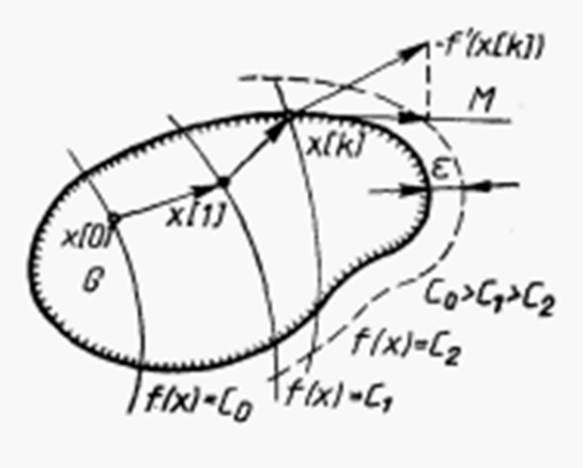

Рис. 1. Геометрическая интерпретация метода проекции градиента

В точке *x*[*k*] часть ограничений-неравенств удовлетворяется как равенство: $h_i(x) = 0, j = 1, ..., l; l < m.$

Такие ограничения называют *активными.*

Обозначим через *J* набор индексов j($1 \le j \le l$) этих ограничений. Их уравнения соответствуют гиперповерхностям, образующим границу области *G* в окрестности точки *x*[*k*] . В общем случае эта граница является нелинейной (см. рис. 1). Ограничения $h_j(x)$, $j \in J$, аппроксимируются гиперплоскостями, касательными к ним в точке *x*[*k*]:

$$\sum_{i=1}^{N} \frac{\partial h_j x[k]}{\partial x} (x_i - x_i[x]) = 0, j \epsilon J \qquad (13)$$

Полученные гиперплоскости ограничивают некоторый многогранник *M,* аппроксимирующий допустимую область *G* в окрестности точки *x*[*k*] (рис. 1).

Проекция *p*[*k*] антиградиента *-f'(x*[*k*]*)* на многогранник вычисляется по формуле

p[*k*] = P[*-f'(x*[*k*]*)*].

Здесь P - оператор ортогонального проектирования, определяемый выражением

$P = E - A^T(AA^T)^{-1}A$,

где *E* - единичная матрица размеров *n*; *A* - матрица размеров *l*x*n* . Она образуется транспонированными векторами-градиентами a_j, $j = 1, ..., l$, активных ограничений. Далее осуществляется спуск в выбранном направлении:

$x[k+1] = x[k] + a_k p[k]$.

Можно показать, что точка *x*[*k*+1] является решением задачи минимизации функции *f(x)* в области *G* тогда и только тогда, когда *P[-f'(x*[*k*]*)]* = 0,

т. Е. $-f'(x[k]) = \sum_{j=1}^{l} u_j a_j$, и $u = (u_1, ..., u_l) = (A^TA)^{-1}A^T(-f'(x[k])) > 0$.

Эти условия означают, что антиградиент *(-f'(x*[*k*]*))* целевой функции является линейной комбинацией с неотрицательными коэффициентами градиентов ограничений $h_i(x) = 0$.

В соответствии с изложенным алгоритм метода проекции градиента состоит из следующих операций.

1. В точке *x*[*k*] определяется направление спуска *p*[*k*].
2. Находится величина шага a_k.
3. Определяется новое приближение *x*[*k*+1].

3.1.2. Комплексный метод Бокса

Этот метод представляет модификацию метода деформируемого многогранника и предназначен для решения задачи нелинейного программирования с ограничениями-неравенствами. Для минимизации функции *n* переменных *f(x)* в *n*-мерном пространстве строят многогранники, содержащие $q > n+1$ вершин. Эти многогранники называют *комплексами,* что и определило наименование метода.

Введем следующие обозначения:

$x[j, k] = (x_1[j, k], \ldots, x_i[j, k], \ldots, x_n[j, k])^T$,

где $j = 1, ..., q$; $k = 0, 1, 2, ...$ - *j*-я вершина комплекса на *k*-м этапе поиска;

x[*h, k*] - вершина, в которой значение целевой функции максимально, т. е. $f(x[h, k]) = \max\{f(x[1, k]), ..., f(x[q, k])\}$; *x*[*h, k*]- центр тяжести всех вершин, за исключением *x*[*h, k*] . Координаты центра тяжести вычисляются по формуле

$$x_i[l,k] = \frac{1}{q}\left(\sum_{j=1}^{q} x_i[j,k] - x_i[h,k]\right), i = \overline{1,n} \qquad (14)$$

Алгоритм комплексного поиска состоит в следующем. В качестве первой вершины начального комплекса выбирается некоторая допустимая точка $x[1,0]$. Координаты остальных q-1 вершин комплекса определяются соотношением

$x_j[j, 0] = a_i + r_i(b_i - a_i), i = 1, ..., n; j = 2, ..., q.$

Здесь a_i, b_i - соответственно нижнее и верхнее ограничения на переменную x_i', r_i - псевдослучайные числа, равномерно распределенные на интервале [0,1]. Полученные таким образом точки удовлетворяют ограничениям $a \le x \le b$, однако ограничения $h_j(x) \le 0$ могут быть нарушены. В этом случае недопустимая точка заменяется новой, лежащей в середине отрезка, соединяющего недопустимую точку с центром тяжести выбранных допустимых вершин. Данная операция повторяется до тех пор, пока не будут выполнены все ограничения задачи. Далее, как и в методе деформируемого многогранника, на каждой итерации заменяется вершина $x[h, k]$, в которой значение целевой функции имеет наибольшую величину. Для этого $x[h, k]$ отражается относительно центра тяжести $x[l, k]$ остальных вершин комплекса. Точка $x[p, k]$, заменяющая вершину $x[h, k]$, определяется по формуле

$x[p, k] = (a+1)x[l, k] + ax[h, k],$

где $a > 0$ - некоторая константа, называемая *коэффициентом отражения*. Наиболее удовлетворительные результаты дает значение $a = 1{,}3$. При этом новые вершины комплекса отыскиваются за небольшое количество шагов, а значения целевой функции уменьшаются достаточно быстро.

Достоинствами комплексного метода Бокса являются его простота, удобство для программирования, надежность в работе. Метод на каждом шаге использует информацию только о значениях целевой функции и функций ограничений задачи. Все это обусловливает успешное применение его для решения различных задач нелинейного программирования.

3.2. Методы штрафных функций

Методы штрафных функций относятся к группе непрямых методов решения задач нелинейного программирования:

$f(x) \to \min;$

$g_i(x) = 0, i = 1, ..., k;$

$h_j(x) < 0, j = 1, ..., m;$

$a \le x \le b.$

Они преобразуют задачу с ограничениями в последовательность задач безусловной оптимизации некоторых вспомогательных функций. Последние получаются путем модификации целевой функции с помощью функций-ограничений таким образом, чтобы ограничения в явном виде в задаче оптимизации не фигурировали. Это обеспечивает возможность применения методов безусловной оптимизации. В общем случае вспомогательная функция имеет вид

$F(x,a) = f(x) + \Phi(x, a)$.

Здесь $f(x)$ - целевая функция задачи оптимизации; $\Phi(x, a)$ - “штрафная” функция; параметр $a > 0$. Точку безусловного минимума функции $F(x, a)$ будем обозначать через $x(a)$. В зависимости от вида $\Phi(x, a)$ различают методы внутренних штрафных, или барьерных, функций и методы внешних штрафных функций.

3.2.1. Методы внутренних штрафных функций

Эти методы применяются для решения задач нелинейного программирования с ограничениями-неравенствами. В рассматриваемых методах функции $\Phi(x,a)$ подбирают такими, чтобы их значения неограниченно возрастали при приближении к границе допустимой области G (рис. 2). Иными словами, приближение к границе “штрафуется” резким увеличением значения функции $F(x,a)$. На границе G построен “барьер”, препятствующий нарушению ограничении в процессе безусловной минимизации $F(x, a)$. Поиск минимума вспомогательной функции $F(x,a)$ необходимо начинать с внутренней точки области G . При этом в процессе оптимизации траектория спуска никогда не выйдет за пределы допустимой области. Все перечисленные особенности функции $\Phi(x, a)$ определили наименование рассматриваемой группы методов.

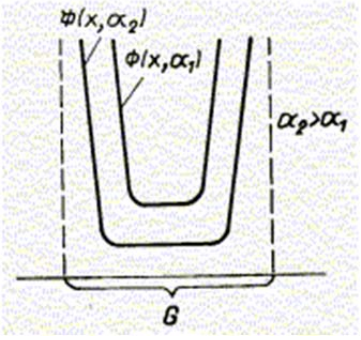

Рис. 2. Внутренняя штрафная функция

Таким образом, внутренняя штрафная функция $\Phi(x,a)$ может быть определена следующим образом:

$$\Phi(\mathrm{x},\mathrm{a}) = \begin{cases} \to \infty, & \text{если } \mathrm{x}\epsilon G \\ \to 0, & \text{если } \mathrm{x}\epsilon G, x \notin dG, a \to 0 \\ \to \infty, & \text{если } \mathrm{x}\epsilon G, x \to dG \end{cases} \qquad (15)$$

Здесь dG -граница области G.

Общий вид внутренней штрафной функции

$$\Phi(\mathrm{x},\mathrm{a}) = \mathrm{a}\left(\sum_{j=1}^{m} \varphi_j h_J(x)\right) \qquad (16)$$

где φ_j - непрерывные дифференцируемые функции, определяемые ограничениями-неравенствами исходной задачи нелинейного программирования. Вспомогательная функция $F(x, a)$ при этом имеет форму

$$F(\mathrm{x},\mathrm{a}) = f(x) + \mathrm{a}\left(\sum_{j=1}^{m} \varphi_j h_j(x)\right) \qquad (17)$$

Она определена в области *G* и неограниченно возрастает, если $h_j\ (x) \to 0$ для некоторого *j*. В качестве внутренних штрафных функций используют, например, такие:

$$\Phi(\mathrm{x},\mathrm{a}) = \mathrm{a}\left(\sum_{j=1}^{m} \frac{1}{h_j(x)}\right);\ \Phi(\mathrm{x},\mathrm{a}) = -\mathrm{a}\left(\sum_{j=1}^{m} ln\, h_j(x)\right) \qquad (18)$$

Алгоритм метода внутренних штрафных функций состоит в следующем. В качестве начальной точки *x*[0] выбирается произвольная внутренняя точка области *G*. Задается некоторая монотонно убывающая сходящаяся к нулю последовательность $\{a_k\}$, $k = 1, 2, ...,$ положительных чисел. Для первого элемента a_1этой последовательности решается задача безусловной минимизации функции *F(x, a),* в результате чего определяется точка $x(a_1)$. Эта точка используется в качестве начальной для решения задачи поиска минимума функции *F(x, a2),* где $a_2 < a_1$, и т. д. Таким образом, решается последовательность задач безусловной минимизации функций $F(x, a_k)$, $k = 1, 2, ...,$ причем решение предыдущей задачи $x(a_k)$ используется в качестве начальной точки для поиска последующего вектора $x(a_{k+}1)$. Последовательность полученных таким образом точек $x(a_k)$ сходится к оптимальному решению исходной задачи - локальному минимуму *x**. Вычисления прекращают при выполнении условий:

$|f(x[k]) - f(x[k\text{-}1])| \le \varepsilon$;

$||x[k] - x[k\text{-}1]|| \le \beta$;

Здесь ε, β - заданные числа, определяющие точность вычислений.

Можно показать, что рассмотренный метод внутренних штрафных функций обладает следующими свойствами:

1)$\lim_{k\to\infty} a_k \sum_{j=1}^{m} \varphi_j \left(h_j(x[k])\right) = 0$

2)$\lim_{k\to\infty} f(x[k]) = f(x^*)$ и *f (x[k])* монотонно убывает

3)$\lim_{k\to\infty} F(x[k]a_k) = f(x^*)$

Эти свойства справедливы для задач, содержащих непрерывные функции и имеющих локальные минимумы внутри области *G*.

3.2.2. Методы внешних штрафных функций

Данные методы применяются для решения задачи оптимизации в общей постановке, т. е. при наличии как ограничений-неравенств, так и ограничений-равенств. В рассматриваемых методах функции Ф*(x, a*) выбирают такими, что их значения равны нулю внутри и на границе допустимой области *G,* а вне ее - положительны и возрастают тем больше, чем сильнее нарушаются ограничения (рис. 3). Таким образом, здесь “штрафуется” удаление от допустимой области *G*.

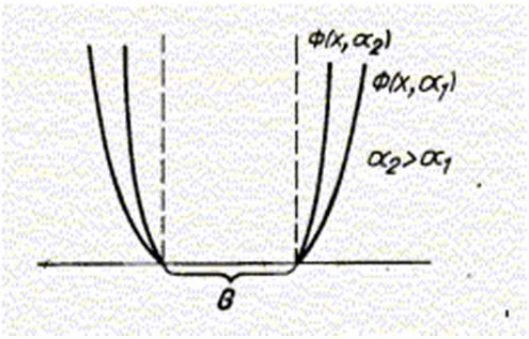

Рис. 3. Внешняя штрафная функция

Внешняя штрафная функция Ф(x,a) в общем случае может быть определена следующим образом:

$$\Phi(\mathrm{x},\mathrm{a}) = \begin{cases} \to 0, & \text{если } \mathrm{x}\epsilon G \\ \to \infty, & \text{если } \mathrm{x}\epsilon G, a \to \infty. \end{cases} \qquad (19)$$

Поиск минимума вспомогательной функции *F(x, a)* можно начинать из произвольной точки. В большинстве случаев она является недопустимой, поэтому траектория спуска располагается частично вне допустимой области. Если минимум целевой функции расположен на границе допустимой области, то эта траектория полностью находится снаружи области *G*. Перечисленные особенности функции *Ф(x, a)* определили название данной группы методов. Общий вид внешней штрафной функции:

$$\Phi(\mathrm{x},\mathrm{a}) = \mathrm{a}\left(\sum_{i=1}^{k}\psi_i\left(g_i(x)\right) + \sum_{j=1}^{m}\varphi_j\left(h_j(x)\right)\right), \qquad (20)$$

где ψ_i, φ_j - функции, определяемые соответственно ограничениями-равенствами и неравенствами исходной задачи нелинейного программирования. Вспомогательная функция *F(x, a)* при этом имеет форму

$$F(\mathrm{x},\mathrm{a}) = f(\mathrm{a}) + a\left(\sum_{i=1}^{k}\psi_i\left(g_i(x)\right) + \sum_{j=1}^{m}\varphi_j\left(h_j(x)\right)\right), \qquad (21)$$

Одна из применяемых внешних штрафных функций имеет вид

$$\Phi(\mathrm{x},\mathrm{a}) = f(\mathrm{a}) + a\left(\sum_{i=1}^{k}(max(0, g_i(x)))^2 + \sum_{j=1}^{m}\left(h_j(x)\right)^2\right), \qquad (22)$$

где $\max\left(0, g_i(x)\right) = \begin{cases} 0, & \text{если } g_i(x) \le 0 \\ g_i(x), & \text{если } g_i(x) > 0 \end{cases}$.

Алгоритм метода внешних штрафных функций формулируется так же, как и алгоритм метода внутренних штрафных функций, и обладает аналогичными свойствами. Однако в этом случае не требуется, чтобы начальная точка $x[0] \in G$, а последовательность $\{a_k\}$, $k = 1, 2, \ldots$, положительных чисел должна быть монотонно возрастающей.

3.3. Практическое решение задач безусловной оптимизации

Функция fmincon

fmincon – функция поиска минимума скалярной функции многих переменных при наличии ограничений вида

c(**x**) < 0, **ceq**(**x**) = 0,

A x < **b**, **Aeq x** = **beq**,

lb < **x** < **ub**

(задача нелинейною программирования). Функция записывается в виде

x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2, ...)

[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(...)

Аргументы и возвращаемые величины практически аналогичны рассмотренным для функции fgoalattain за следующими исключениями:

имеется дополнительная возвращаемая величина grad – градиент функции в точке минимума;

имеется возможность задания вычисления гессиана **H** (вводом функции options = optimset ('Hessian','on'), что должно быть отражено в m-файле:

function [f,g,H] = myfun(x)

f = … % Вычисление целевой функции

g = … % Вычисление градиента

H = … % Вычисление гессиана

имеется дополнительная возвращаемая величина hessian – гессиан **H** функции в точке минимума;

возможны дополнительные опции и имеются различия в их использовании:

LargeScale – может принимать значения 'off' (no умолчанию) и 'on'. В первом случае используется алгоритм средней размерности, во втором – алгоритм большой размерности.

Следующие опции используются только при работе с алгоритмом средней размерности (описание см. иыше):

DerivativeCheck;

DiffWaxChange;

DiffMinChange;

LineSearchType – задание вида алгоритма одномерной оптимизации.

Опции, используемые только в алгоритме большой размерности:

Hessian – гессиан (в случае матрицы Гессе, задаваемой пользователем, см. выше);

HessPattern – задание гессиана как разреженной матрицы (это может привести к существенному ускорению поиска минимума);

MaxPCGIter – максимальное число итераций PCG-алгоритма (preconditioned conjugate gradient, см. выше);

PrecondBandWidth – верхняя величина начальных условий для PCG-алгоритма;

TolPCG – допуск на завершение PCG-итераций;

TypicalX – типовые величины x;

5) возвращаемая величина output в данном случае имеет дополнительные компоненты:

output.cgiterations – число PCG-итераций (только при использовании алгоритма большой размерности);

output.stepsize – величина конечного шага поиска (только при использовании алгоритма средней размерности);

output.firstorderopt – мера оптимальности первого порядка (норма вектора градиента в точке минимума) – только при использовании алгоритма большой размерности).

Пример 1. Пусть требуется найти минимум функции $f(x) = -x_1x_2x_3$ при начальном значении **x** = [10;10;10] и при наличии ограничении $0 \le x_1 + 2x_2 + 2x_3 \le 72$,

Решение. Вначале создадим m-файл, определяющий целевую функцию:

```
function f = myfun(x)
f = - x(1)*x(2)*x(3);
```

Затем запишем ограничения в виде неравенств:

$$-x_1 - 2x_2 - 2x_3 \le 0$$
$$x_1 + 2x_2 + 2x_3 \le 72$$

или в матричной форме:

A x ≤ **b**,

где $A = \begin{bmatrix} -1 & -2 & -2 \\ 1 & 2 & 2 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 72 \end{bmatrix}$.

Теперь нахождение решения представляется следующим образом:

```
>> A = [-1 -2 -2; 1 2 2];
>> b = [0; 72];
>> x0 = [10; 10; 10]; % Стартовое значение
>> [x,fval] = fmincon('myfun',x0,A,b)
x =      24.0000
12.0000
12.0000
fval =-3.4560e+003
```

Пример 2. Теперь рассмотрим задачу минимизации той же целевой функции, но при наличии ограничений в форме нелинейных неравенств:

$x_1x_2 - 2x_1 - x_2 \le -1.5,$

$x_1x_2 \ge -10.$

Данная задача (см. таблицу 1) может быть решена с применением функции fmincon в соответствии с теми же этапами, что и предыдущая. Поскольку при использовании данной функции нелинейные ограничения должны быть представлены и виде **c**(**x**) ≤ 0, перепишем их соответствующим образом:

$x_1x_2 - x_1 - x_2 + 1.5 \le 0,$

$-x_1x_2 - 10 \ge 0.$

Этап 1. Составление m-файла (с именем confun), возвращающего значения левых частей ограничивающих неравенств:

```
function [c,ceq] = confun(x)
% нелинейные ограничения в форме неравенств
c = [1.5 + x(1)*x(2) - x(1) - x(2); -x(1)*x(2) - 10];
% нелинейные ограничения в форме равенств
ceq = [];
```

Этап 2. Составление программы минимизации:

```
>> x0 = [-1,1]; % Задание начальных значений
>> options = optimset('LargeScale','off'); % Задание опций
>> % поиск решения
>> [x,fval] = fmincon('objfun',x0,[],[],[],[],[], [],'confun', options)
```

Результаты вычислений сообщают о найденном решении, количестве ограничении и т. п.:

```
Active Constraints:
1
2
x =-9.5474     1.0474
fval =0.0236
```

Пример 3. Функция fmincon может быть применена и для поиска решения в задаче минимизации, в которой допустимые значения переменных ограничены некоторыми диапазонами: **lb ≤ x ≤ ub**.

Продолжим рассмотрение примера, введя дополнительные ограничения $x_1 \geq 0$, $x_2 \geq 0$, что эквивалентно заданию lb = [0 0], ub = [].

Программа оптимизации и результаты вычислений для данного случая приведены ниже:

```
>> x0 = [-1,1]; % начальных значений
>> lb = [0,0]; % нижние границы переменных
>> ub = []; % сверху переменные не ограничены
>> options = optimset('LargeScale','off'); % опции
>> % поиск решения
>> [x,fval] = fmincon('objfun',x0,[],[],[],[],lb, ub,'confun', options)
Active Constraints:
1
3
x =0     1.5000
fval =8.5000
```

Найденное решение удовлетворяет ограничениям, наложенным на диапазоны изменения переменных. Проверим теперь выполнение ограничений в форме неравенств:

```
>> % проверка выполнения ограничений в форме неравенств
>> [c,ceq] = confun(x)
```

c =[0 -10]

ceq =[]

Как следует из приведенного результата, ограничения в форме неравенств выполнены (ограничения в форме равенств отсутствуют).

Пример 4. По умолчанию функции пакета заменяют точные значения производных целевой функции и ограничений (в требуемых случаях) их оценками в виде первых и/или вторых разностей. Иногда целесообразно задать аналитическое вычисление данных производных – это может привести к ускорению процесса поиска решения. Рассмотрим такой подход при задании векторов первых производных – векторов-градиентов.

Продолжая решение в условиях приведенного примера, получим следующие аналитические выражения:

градиент целевой функции:

$$\frac{df(x)}{dx} = \begin{bmatrix} e^{x_1}(4{x_1}^2 + 2{x_2}^2 + 4x_1x_2 + 2x_2 + 1) + e^{x_1}(8x_1 + 4x_2) \\ e^{x_1}(4x_1 + 4x_2 + 2 \end{bmatrix}$$

матрица, столбцы которой являются градиентами функций в левых частях ограничений-неравенств:

$$\begin{bmatrix} \frac{dc_1}{dx_1} & \frac{dc_2}{dx_1} \\ \frac{dc_1}{dx_2} & \frac{dc_2}{dx_2} \end{bmatrix} = \begin{bmatrix} x_2 & -1 & -x_2 \\ x_1 & -1 & -x_1 \end{bmatrix}$$

Используя данные выражения, создадим требуемые m-файлы.

Этап 1. Создание m-файла (с именем objfungrad) для расчета значений целевой функции и ее градиента:

```
function [f,G] = objfungrad(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
% градиент целевой функции
t = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
G = [t+exp(x(1))*(8*x(1)+4*x(2)),exp(x(1))*(4*x(1)+4*x(2) +2)];
```

Этап 2. Создание m-файла для нелинейных ограничений и их градиента:

```
function [c,ceq,DC,DCeq] = confungrad(x)
% нелинейные ограничения-неравенства
c(1) = 1.5+x(1)*x(2)-x(1)-x(2);
c(2) = -x(1)*x(2)-10;
% градиент ограничений-неравенств:
DC = [x(2)-1, -x(2); x(1)-1, -x(1)];
% нелинейные ограничения-равенства отсутствуют
ceq = [];
DCeq = [];
```

Этап 3. Создание программы оптимизации.

```
>> x0 = [-1,1];
>> options = optimset('LargeScale','off');
```

```
>> % разрешение использования градиентов
>> options = optimset(options,'GradObj','on','GradConstr', 'on');
>> lb = []; ub = []; % границы диапазонов не заданы
>> % поиск решения
>> [x,fval] = fmincon ('objfungrad', x0,[],[],[],[],lb,ub, 'confungrad',options)
Результаты вычислений:
Active Constraints:1
                2
x =-9.5474     1.0474
fval =0.0236
Проверка выполнения ограничений:
>> c
c =         0
-10
```

Как видно, в точке решения заданные ограничения выполняются.

Пример 5. Найти минимум функции

$$f = 3(1- x_1)^2 exp\,(-x_1^2 - (x_2 + 1)^2) - 10(x_1/5 - x_1^3 - x_2^5)exp\,(-x_1^2 - -x_2^2) - -1/3exp\,(-(x_1 + 1)^2 - x_2^2)$$

на параллелепипеде

$-3 \le x_j \le 3, j = 1, 2.$

```
function f=myfun(x)
f=3*(1-x(1))^2*exp(-x(1)^2-(x(2)+1)^2)-10*(x(1)/5-x(1)^3-x(2)^5)*exp(-
x(1)^2-x(2)^2)-1/3*exp(-(x(1)+1)^2-x(2)^2);
end
```

Затем составляем программу поиска минимума и отображения целевой функции, ее линий уровня и траектории поиска в виде функции от начальной точки:

```
function history = runfmincon5_1(x0)
history.x = [];
history.fval = [];
% call optimization
options = optimset('outputfcn',@outfun,'display','off','Algorithm',...
'interiorpoint');
[x,fval,exitflag,output]= fmincon(@myfun,x0,[],[],[],[],[-3 -3],[3 3],[],options)
[X,Y]=meshgrid(-3:0.02:3);Z=3*(1-X).^2.*exp(-X.^2-(Y+1).^2)-10*(X/5 ...
-X.^3-Y.^5).*exp(-X.^2-Y.^2)-1/3*exp(-(X+1).^2-Y.^2);
meshc(X,Y,Z);
y=history.x(:,1);z=history.x(:,2);v=history.fval;
hold on;plot3(y,z,v,'black.',y,z,v,'red-');
v=-10*ones(size(z));hold on;plot3(y,z,v,'red-');
xlabel('x1'); ylabel('x2');zlabel('f');
title('Sequence of Points Computed by fmincon');
function stop = outfun(x,optimValues,state)
```

stop = false; switch state
case 'init' hold off case 'iter'
% Concatenate current point and objective function
history.fval = [history.fval; optimValues.fval];
history.x = [history.x; x];
end end end end
Результат выполнения команды
>> x0=[1 1]; history = runfmincon5_1(x0)
получаем
x = 0.2283 -1.6255
fval =-6.5511, exitflag =1
output =
iterations: 16, funcCount: 56, constrviolation: 0, stepsize: 4.8893e-008
algorithm: 'interior-point', firstorderopt: 1.2023e-007
cgiterations: 0, message: [1x782 char]

На рис. 4 показан процесс поиска минимума. Как видно из представленных результатов, он оказался успешным.

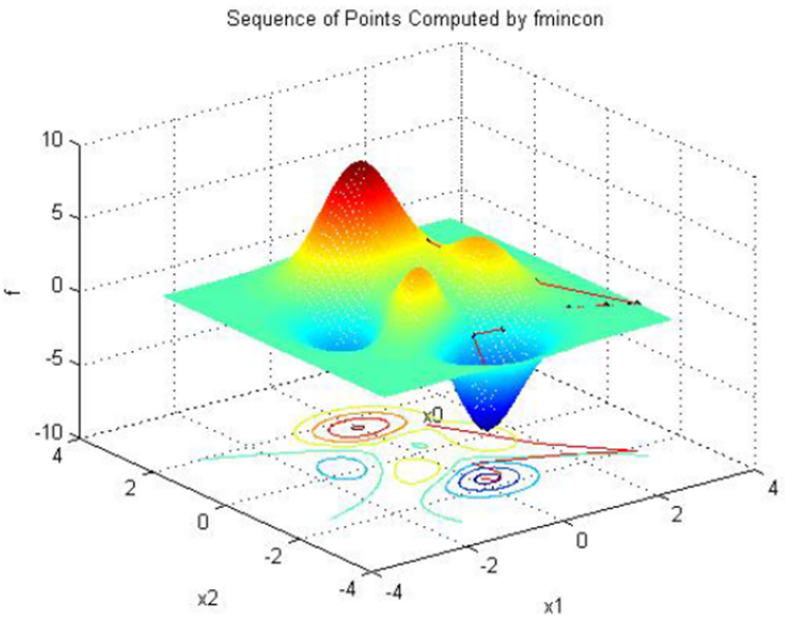

Рис. 4. Вид целевой функции и траектория движения к минимуму

Однако этот результат следует считать случайным, так как случайно выбрана данная начальная точка.

Повторив поиск из другой начальной точки, получаем
>> x0=[-0.5 3]; history = runfmincon5_1(x0)
x =-2.9984 2.9976
fval =
3.2869e-005
exitflag =1
output =

iterations: 27, funcCount: 84, constrviolation: 0
stepsize: 7.2753e-004
algorithm: 'interior-point'
firstorderopt: 3.2032e-007
cgiterations: 0, message: [1x782 char]

Из показателей поиска (признака exitflag, критериев точности) как будто следует, что получено решение задачи. На самом деле поиск завершился даже не в точке одного из локальных минимумов, что хорошо видно на рис. 5.

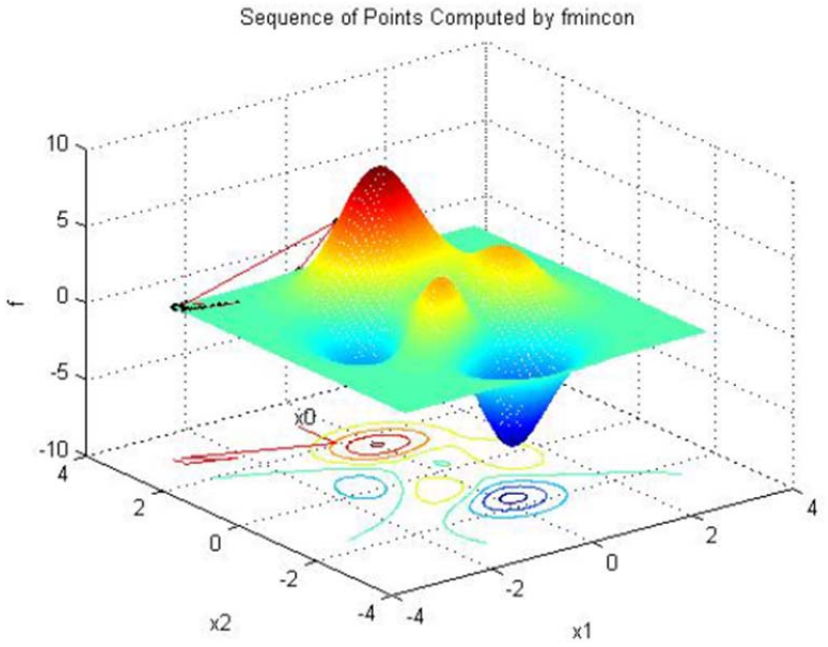

Рис. 5. Поиск минимума из точки (–0,5 3)

Примечание. В Toolbox Optimization имеется функция rtrlink, по функциональности аналогичная fmincon. Она применима для решения задач без ограничений и с линейными и нелинейными ограничениями. В ней также используются алгоритмы active-set и interior-point. Однако она требует установки самостоятельной библиотеки KNITRO, к которой происходит обращение при вызове rtrlink.

Функция fseminf

Функция fseminf предназначена для нахождения решения задачи полубесконечной минимизации с ограничениями (см. таблицу 1).

Запись:

x = fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq,lb,ub,options,P1,P2,...)

[x,fval,exitflag,output,lambda] = fseminf(...)

Описание. Аргументы и возвращаемые величины данной функции в основном аналогичны ранее рассмотренным. Добавлены лишь аргументы, отражающие специфику задачи:

• ntheta – число полубесконечных ограничений вида $K_i(\mathbf{x},w_i) \le 0$, где переменные w_i – некоторые задаваемые векторы;

• seminfcon – функция (имя функции), возвращающая значения векторов ограничений при заданном **x** и оформленная в виде соответствующего m-файла, например так (файл этого примера должен иметь имя myinfcon):

```
function [c,ceq,K1,K2,...,Kntheta,S] = myinfcon(x.S)
if isnan(S(1.1)), % Начальный интервал дискретизации
S = ... % S имеет ntheta строк и 2 столбца
end
w1 = ... % Набор дискретных значений аргумента w1
w2 = ... % Набор дискретных значений аргумента w2
...
ntheta = ... % Набор дискретных значений аргумента wntheta
K1 = ... % Первое полубесконечное ограничение, зависящее от x и wl
K2 = ... % Второе полубесконечное ограничение, зависящее от x и w2
...
% Последнее полубесконечное ограничение, зависящее от x и wntheta
Kntheta =
c = ... % Вычисление левой части нелинейного неравенства
ceq = ... % Вычисление левой части нелинейного равенства
```

Здесь S – рекомендуемый интервал дискретизации (для расчета левых частей полубесконечных ограничений), соответственно, для w_1, w_2, … w_{ntheta}; может быть опущен.

Пример. Рассмотрим задачу минимизации функции $f(\mathbf{x}) = (x_1 - 0{,}5)^2 + (x_2 - 0{,}5)^2 + (x_3 - 0{,}5)^2$ при наличии ограничений

$$K_1(\mathbf{x}, w_1) = \sin(w_1 x_1)\cos(w_1 x_2) - \frac{1}{1000}(w_1 - 50)^2 - \sin(w_1 x_3) - x_3 - 1 \le 0,$$

$$K_2(\mathbf{x}, w_2) = \sin(w_2 x_2)\cos(w_2 x_1) - \frac{1}{1000}(w_2 - 50)^2 - \sin(w_2 x_3) - x_3 - 1 \le 0,$$

где w_1 и w_2 принадлежат отрезку [1, 100].

Для решения задачи создадим два m-файла, первый (с именем myfun) – для вычислений целевой функции, второй (с именем mycon) – для вычислений левых частей ограничений:

```
function f = myfun(x,s)
f = sum((x-0.2).^2); % Целевая функция
function [c,ceq,K1,K2,s] = mycon(X,s)
if isnan(s(1,1)),
        s = [0.2 0; 0.2 0]; % Начальный интервал дискретизации
end
w1 = 1:s(1,1):100; % дискретные значения
w2 = 1:s(2,1):100;
```

```
% полубесконечные ограничения
K1 = sin(w1*X(1)).*cos(w1*X(2))-1/1000*(w1-50).^2-sin(w1*X(3))-X(3)-1;
K2 = sin(w2*X(2)).*cos(w2*X(1))-1/1000*(w2-50).^2-sin(w2*X(3))-X(3)-1;
% другие ограничения отсутствуют
 c = []; ceq=[];
plot(w1,K1,'-',w2,K2,':'),title('Semiinfinite conditions')
drawnow
```

Решение задачи иллюстрируется следующим образом:

```
>> % Стартовые значения
>> x0 = [0.5; 0.2; 0.3];
>> % Нахождение решения
>> [x,fval] = fseminf(@myfun,x0,2,@mycon)
x =
0.6535
0.2821
0.4013
```

На рисунке 3 приведен графический вид функций в правых частях полубесконечных ограничении, возвращаемый m-файлом mycon для точки минимума. Как видно из рисунка, ограничения удовлетворяются.

Функция linprog

Функция linprog обеспечивает решение задачи линейного программирования (см. таблицу 1).

Запись:

```
x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
 [x,fval,exitflag,output,lambda] = linprog(...)
```

Аргументы и возвращаемые величины здесь аналогичны рассмотренным ранее для других функций за одним исключением: здесь введен дополнительный аргумент f – вектор коэффициентов линейной целевой функции. Функция может использовать алгоритм большой размерности lipsol или алгоритм средней размерности (метод проекций).

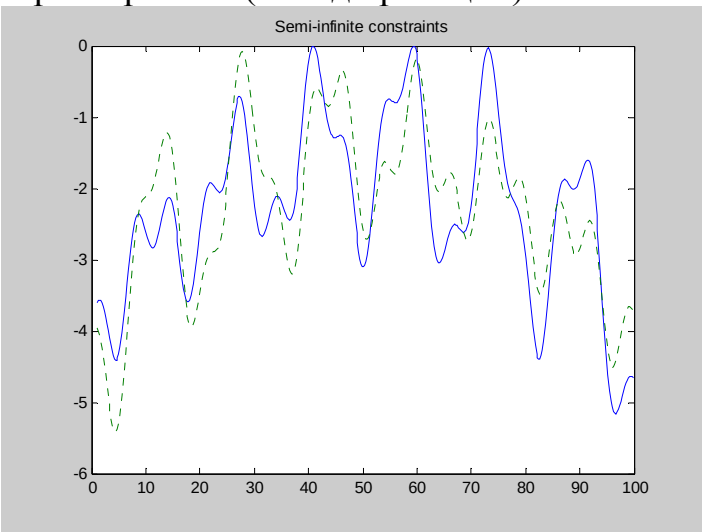

Рис. 6. Графический вид функций в правых частях полубесконечных ограничений

Пример. Требуется найти решение задачи линейного программирования, описывающейся соотношениями

$f(\mathbf{x}) = -5x_1 - 4x_2 - 6x_3$,

$3x_1 + 2x_2 + 4x_3 \le 42$,

$3x_1 + 2x_2 \le 42$,

$0 \le x_1$, $0 \le x_2$, $0 \le x_3$.

Решение задачи приведено ниже.

```
>> f = [-5; -4; -6]; % Вектор коэффициентов линейной целевой функции
>> % Матрица коэффициентов ограничений-неравенств
>> A = [1 -1 1; 3 2 4; 3 2 0];
>> b = [20; 42; 30]; % Вектор ограничений-неравенств
>> % Задание нижних границ переменных (нулей)
>> lb = zeros(3,1);
>> % Поиск решения
>> [x,fval,exitflag,output,lambda] = linprog(f,A,b,[], [], lb)
Optimization terminated successfully.
x =       0.0000
15.0000
3.0000
fval =-78.0000
exitflag =1
output =iterations: 6
cgiterations: 0
algorithm: 'lipsol'
lambda =ineqlin: [3x1 double]
eqlin: [0xl double]
upper: [3x1 double]
lower: [3x1 double]
```

Из возвращенной информации, в частности, следует:

- что оптимальное решение x = [0.0000 15.0000 3.0000];
- минимальное значение целевой функции равно –78.0000;
- вычисления завершились нахождением решения (exitflag > 0);
- всего было выполнено 6 итераций;
- был использован алгоритм lipsol

и т. п.

Функция quadprog

Функция quadprog возвращает решение задачи квадратичного программирования. Может использовать алгоритм как средней, так и большой размерности.

Запись:

x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options,p1,p2,...)

[x,fval,exitflag,output,lambda] = quadprog(...)

Аргументы (за исключением матрицы **H**) и возвращаемые величины аналогичны рассмотренным для функции fmincon.

Пример 1. Найдем решение задачи квадратичного программирования, имеющей описание

$$f(x) = \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 2x_1 - 6x_2$$

$$x_1 + x_2 \le 2$$
$$-x_1 + 2x_2 \le 2$$
$$2x_1 + x_2 \le 3$$
$$0 \le x_1,\ \ 0 \le x_2$$

В данном случае

$H = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix},\ f = \begin{bmatrix} -2 \\ -6 \end{bmatrix},\ x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

и решение задачи представляется в следующем виде:

```
>> H = [1 -1; -1 2];
>> f = [2; -6];
>> A = [1 1; -1 2; 2 1];
>> b = [2; 2; 3];
>> lb = zeros(2,1);
>> [x,fval,exitflag,output,lambda] = quadprog(H,f,A,b,[], [],lb)
Optimization terminated successfully.
x =       0.6667
1.3333
fval =-8.2222
exitflag =1
output =
iterations: 3
algorithm: 'medium-scale: active-set', firstorderopt: [], cgiterations: []
lambda =
lower: [2x1 double], upper: [2x1 double]
eqlin: [0xl double], ineqlin: [3x1 double]
```

Из возвращаемой информации следует, что оптимальное значение x = [0.6667 1.3333], минимальное значение целевой функции –8.2222, количество выполненных итераций –3, использован алгоритм средней размерности.

Пример 2. Для минимизации квадратичной формы, зависящей от большого числа переменных, следует использовать функцию quadprog. Рассмотрим подобную задачу при числе переменных 400 и симметричной трехдиагональной матрице **H**, сохраненной в файле qpbox1.mat.

Соответствующая программа и результаты вычислений приведены ниже.

```
>> load qpbox1 % загрузка матрицы H
```

```
>> % задание граничных значений
>> lb = zeros(400,1); lb(400) = -inf;
>> ub = 0.9*ones(400,1); ub(400) = inf;
>> f = zeros(400,1); f([1 400]) = -2;
>> xstart = 0.5*ones(400,1); % начальные значения
>> % поиск решения
>> [x,fval,exitflag,output] =quadprog(H,f,[],[],[],[],lb,ub,xstart);
>> exitflag
exitflag =1
>> output
output =
firstorderopt: 7.8435e-006
iterations: 20
cgiterations: 1809
algorithm: 'large-scale: reflective trust-region'
```

4. Многокритериальная оптимизация

В теории *многокритериальной оптимизации* (МКО) решаются задачи принятия решений одновременно по нескольким критериям. Задача МКО ставится следующим образом: требуется найти числа $x_1, x_2, \dots, x_n$, удовлетворяющие системе ограничений

$$g_i(x_1, x_2, \dots, x_m) \le b_i,\ i = \overline{1,m}, \quad (23)$$

для которых функции

$$F_k = f_k(x_1, x_2, \dots, x_n),\ k = \overline{1,K}, \quad (24)$$

достигают максимального значения.

Множество точек $X = (x_1, x_2, \dots, x_n)$, удовлетворяющих системе (23), образует *допустимую область* $D \subset R^n$. Элементы множества D называются *допустимыми решениями* или *альтернативами*, а числовые функции f_k, $k = \overline{1,K}$ – *целевыми функциями*, или *критериями*, заданными на множестве D. В формулировке задаче (23) - (24) присутствует K целевых функций. Эти функции отображают множество $D \subset R^n$ в множество $F \subset R^n$, которое называется *множеством достижимости*.

В векторной форме математическую модель МКО (23)-(24) можно записать следующим образом:

$$f(X) = \big(f_1(X), \dots, f_2(X)\big) \to max,\ X \in D \quad (25)$$

Здесь *f(X)* – вектор-функция аргумента $X \in D$.

В теории МКО понятие оптимальности получает различные толкования, и поэтому сама теория содержит три основных направления:

1. Разработка концепции оптимальности.

2.Доказательство существования решения, оптимального в соответствующем смысле.

3. Разработка методов нахождения оптимального решения.

Если функции $f_1, f_2, \dots, f_K$ достигают максимум в одной и той же точке $X^* \in D$, то говорят, что задача (25) имеет *идеальное решение*.

Случаи существования идеального решения в многокритериальной задаче крайне редки. Поэтому основная проблема при рассмотрении задачи (16) – формализация *принципа оптимальности*, т.е. определение того, в каком смысле «оптимальное» решение лучше других. В случае отсутствия «идеального решения» в задаче (25) *ищется компромиссное решение.*

Для всякой альтернативы $X \in D$ вектор из значений целевых функций $(f_1(X), f_2(X), \dots, f_K(X))$ является *векторной оценкой* альтернативы *X.* Векторная оценка альтернативы содержит полную информацию о ценности (полезности) этой альтернативы для главного конструктора системы, или, как принято говорить в системном анализе, лица, принимающего решение (ЛПР). Сравнение любых двух исходов заменяется сравнением их векторных оценок.

Пусть $X_1, X_2 \in D$. Если для всех критериев $f_1, f_2, \dots, f_K$ имеют место неравенства $f_k(X_2) \ge f_k(X_1),\ k = \overline{1,K}$, причем хотя бы одно неравенство

строгое, то говорят, что решение X_2 *предпочтительнее* решения X_1. Условие предпочтительности принято обозначать в виде $X_2 \succ X_1$.

Определение (оптимальность по Парето). В задаче МКО точка $X_0 \in D$ называется оптимальной по Парето, если не существует другой точки $X \in D$, которая была бы предпочтительнее, чем X_0.

Точки, оптимальные по Парето, образуют множество точек, оптимальных по Парето (множество неулучшаемых или эффективных точек) $D_p \subset D$.

Оптимальные решения многокритериальной задачи следует искать только среди элементов множества альтернатив D_p. В этой области ни один критерий не может быть улучшен без ухудшения хотя бы одного из других. Важным свойством множества Парето D_p является возможность выбрать из множества альтернатив D заведомо неудачные, уступающие другим по всем критериям. Обычно решение многокритериальной задачи должно начинаться с выделения множества D_p. При отсутствии дополнительной информации о системе предпочтений ЛПР должно принимать решение именно из множества Парето D_p.

В векторной оптимизации, кроме множества Парето, в общем случае нет общих правил, по которому варианту X_2 отдается предпочтение по сравнению с другим вариантом X_1.

Часто решение многокритериальной задачи состоит в построении множества Парето-оптимальных точек и дальнейшем выборе одной из них на основе какого-либо критерия.

Во всех случаях задача многокритериальной оптимизации сводиться к задаче с одним критерием. Существует много способов построения такого окончательного критерия, однако ни одному из них нельзя заранее отдать наибольшее предпочтение. Для каждой задачи этот выбор должен делаться ЛПР.

Заметим, что целевые функции отображают множество точек, оптимальных по Парето $D_p \subset D \subset R^n$ в множество $F_p \subset F \subset R^n$, которое называется *множеством Парето*.

4.1. Метод ε-ограничений

Некий определенный способ, который отчасти позволяет преодолеть проблему выпуклости метода взвешенных сумм, есть метод ε-ограничений. В этом случае осуществляется минимизация основной цели F_p и при представлении остальных целей в форме ограничений типа неравенств.

$$min_{x\epsilon\Omega}\, F_p(x) \qquad (26)$$

при выполнении условия

$$F_i(x) \le \varepsilon_i, i = \overline{1,m}, i \ne p \quad (27)$$

На рисунке 7 геометрическая интерпретация метода ε-ограничений, представлена двумерная интерпретация метода ε-ограничений для задачи с двумя целями.

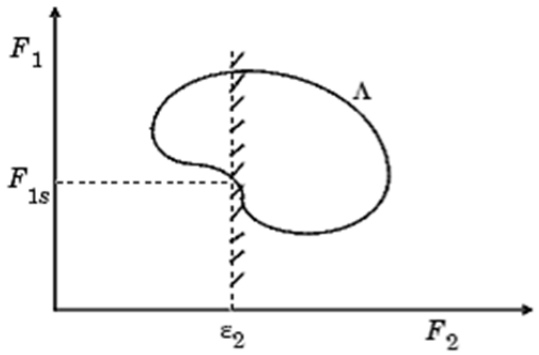

Рис. 7. Геометрическая интерпретация метода ε-ограничений

Подобный подход позволяет определить некое количество неулучшаемых решений для случая вогнутой границы, что, по существу, является недоступным в методе взвешенных сумм, например, в точке искомого решения $F_1 = F_{1s}, F_2 = \varepsilon_2$. Однако проблемой данного метода является подходящий выбор ε, который мог бы гарантировать допустимость некого решения. Следующий недостаток данного метода заключается в необходимости использования жестких ограничений, которые не всегда являются адекватными для точного построения задаваемых целей. Процедура оптимизации выполняется в соответствии с выбранными приоритетами и в пределах допустимости принятых границ. Основная трудность данного метода заключается в точной интерпретации подобной информации на ранних стадиях оптимизационного цикла.

4.2. Метод достижения цели

Описанный далее метод представляет собой метод достижения цели Гембики. Данный метод включает в себя выражение для множества намерений разработчика $F = \{F_1, F_2, \dots, F_m\}$, которое связано с множеством целей $F(x) = \{F_1(x), F_2(x), \dots, F_m(x)\}$. Такая формулировка задачи допускает, что цели могут быть недостижимыми, и что дает разработчику возможность относительно точно выразить исходные намерения. Относительная степень недостижимости поставленных намерений контролируется посредством вектора взвешенных коэффициентов $w = \{w_1, w_2, \dots, w_m\}$ и может быть представлена как стандартная задача оптимизации с помощью следующей формулировки

$$min_{\gamma \in \Re, x \in \Omega}\, \gamma, \quad (28)$$

при условии, что $F_i(x) - w_i\gamma \le F_i, i = \overline{1,m}$

Член $w_i\gamma$ вносит в данную задачу элемент ослабления, что, иначе говоря, обозначает жесткость заданного намерения. Весовой вектор w дает исследователю возможность достаточно точно выразить меру взаимосвязи между двумя целями. Например, установка весового вектора w как равного исходному намерению указывает на то, что достигнут тот же самый процент недостижимости цели F^*. Посредством установки в ноль отдельного весового коэффициента (т.е. $w_i = 0$) можно внести жесткие ограничения в поставленную задачу. Метод достижения цели обеспечивает подходящую интуитивную интерпретацию поставленной исследовательской задачи и которая, в свою очередь, является вполне разрешимой с помощью стандартных процедур оптимизации.

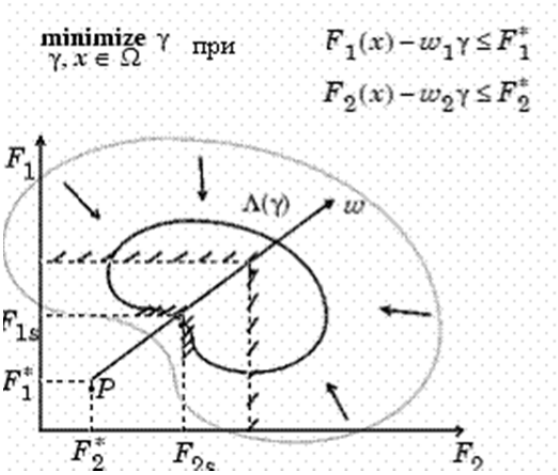

Рис. 8. Геометрическая интерпретация метода достижения цели

Задание компонентов намерений $\{F_1, F_2\}$ определяет точку намерений P. Весовой вектор определяет поиска от точки P в сторону пространства допустимых функций $\Lambda(\gamma)$. В процессе оптимизации происходит изменение величины γ, что вызывает изменение размера заданной допустимой области. Границы ограничений стягиваются к единственной в своем роде точке решения F_{1s}, F_{2s}.

4.3. Практическое решение многокритериальных задач

Функция fminimax

fminimax – функция для решения минимаксных задач (см. таблицу 1):

$\min\limits_{\mathbf{x}} \max\limits_{\{F\}_i} \{F_i(\mathbf{x})\}$ при наличии ограничений

$\mathbf{c}(\mathbf{x}) \le 0$, $\mathbf{ceq}(\mathbf{x}) = 0$,

$\mathbf{A}\,\mathbf{x} \le \mathbf{b}$, $\mathbf{Aeq}\,\mathbf{x} = \mathbf{beq}$,

$\mathbf{lb} \le \mathbf{x} \le \mathbf{ub}$.

Запись функции:

x = fminimax(fun,x0) – возвращает значение x, минимизирующее максимальное значение функций из заданного набора fun при стартовой точке поиска x0.

Другие представления аналогичны рассмотренным для функции fgoalattain:

x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,...)

[x,fval,maxfval,exitflag,output,lambda] = fminimax(...)

Аргументы функции идентичны рассмотренным для функции fgoalattain.

Пример. Пусть требуется решить минимаксную задачу для набора из 5 функций $[f_1(x), f_2(x), f_3(x), f_4(x), f_5(x)]$, где

$f_1(x) = 2x_1^2 + x_2^2 - 48x_1 - 48x_2 + 304,$

$f_2(x) = -x_1^2 - 3x_2^2,$

$f_3(x) = x_1 + 3x_2 - 18,$

$f_4(x) = -x_1 - x_2,$

$f_5(x) = x_1 + x_2 - 8.$

Для получения решения сначала представим данные функции в виде m-файла

```
function f = myfun(x)
f(1) = 2*x(1)^2+x(2)^2-48*x(1)-40*x(2)+304;
f(2) = -x(1)^2-3*x(2)^2;
f(3) = x(1)+3*x(2)-18;
f(4) = -x(1)-x(2);
f(5) = x(1)+x(2)-8;
```

Затем используем функцию fminimax:

```
>> x0 = [0.1; 0.1];  % Стартовое значение
>> [x,fval] = fminimax('myfun',x0)
```

Результаты вычислений:

Optimization terminated successfully:

Magnitude of directional derivative in search direction less than 2*options. TolFun and maximum constraint violation is less than options. TolCon

Active Constraints: 1 5

x =4.0000 4.0000; fval =0.0000 -64.0000 -2.0000 -8.0000 -0.0000

Решение – x = [4, 4].

Функция fgoalattain

Функция fgoalattain служит для решения задачи векторной оптимизации методом «достижения цели». Она записывается в следующем виде:

- x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub) – то же, что и предыдущая функция, но при наличии дополнительных граничных ограничений $\mathbf{lb} \le \mathbf{x} \le \mathbf{ub}$;
- x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub, nonlcon) – аналогично предыдущей функции, но при наличии дополнительных ограничений в форме нелинейных неравенств или равенств $\mathbf{c}(\mathbf{x}) \le 0$, $\mathbf{ceq}(\mathbf{x}) = 0$, при отсутствии граничных ограничений задаются lb = [] и ub = [];

Аргументы функции:

- fun – векторная функция векторного аргумента. Должна быть задана либо с помощью функции inline, например:

```
>> fun = inline('sin(x.*x)');
```

либо как m-файл, например:

```
function F = myfun(x)
F = ...
```

Если задано вычисление градиента (функцией options = optimset('GradObj','on')), то m-файл должен возвращать не только значение функции **F**, но и значения градиентов **G**:

```
function[F,G] = myfun(x)
F = ... % Вычисление векторной функции
G = ... % Вычисление градиента
```

- goal – вектор задаваемых целевых значений той же размерности, что и вектор fun;
- weight - вектор весов той же размерности, что и вектор целей, часто принимается равным abs(goal).
- nonlcon – функция, возвращающая значения функций-ограничений, а при необходимости (если задано options = optimset ('GradConstr' ,'on')) и их градиентов; должна быть оформлена в виде m-файла, например:

```
function[c,ceq] = mycon(x)
c = ... % Вычисление левых частей нелинейных неравенств
ceq = ... % Вычисление левых частей нелинейных равенств
function [c,ceq,GC,GCeq] = mycon(x)
c = ... % Вычисление левых частей нелинейных неравенств
ceq = ... % Вычисление левых частей нелинейных равенств
GC = ... % Градиенты неравенств
GCeq = ... % Градиенты равенств
```

- Options – опции (их можно изменять, используя функцию optimset):
  - DerivativeCheck – дает проверку соответствия производных, определенных пользователем, их вычисленным оценкам в виде первых разностей;
  - Diagnostics – вод диагностической информации о минимизируемой функции;
  - DiffMaxChange – максимальные значения изменений переменных при определении первых разностей;
  - DiffMinChange – минимальные значения изменений переменных при определении первых разностей;
  - Display – уровень отображения: 'off' – вывод информации отсутствует, 'iter' – вывод информации о поиске решения на каждой итерации, 'final' – вывод только итоговой информации;
  - GoalExactAchieve – определяет количество целей, которые должны быть достигнуты «точно»;

- GradConstr – использование градиентов для ограничений (опция имеет смысл в случае применения аргумента nonlcon, см. выше), возможные значения – 'off' и 'on';
- GradObj – использование градиента для целевой функции, определяемого пользователем (возможные значения – 'off' и 'on');
- MaxFunEvals – максимальное число вычислений функции;
- MaxIter – максимальное допустимое число итераций;
- MeritFunction – устанавливает вид функции оценки качества достижения цели (возможные значения 'multiobj' или 'singleobj');
- TolCon – допуск останова вычислений при нарушении ограничений;
- TolFun – допуск останова вычислений по величине изменений функции;
- TolX – допуск останова вычислений но величине изменений x;

- attainfactor – коэффициент достижения цели, усредненное значение несоответствий заданным целям, выраженное в долевом (процентном) виде. Если данный коэффициент отрицательный, цели были «перекрыты», если положительный – цели не достигнуты;
- exitflag – информация о характере завершения вычислений: если эта величина положительна, то вычисления завершились нахождением решения x, если она равна нулю, то останов произошел в результате выполнения предельного числа итераций, если данная величина отрицательна, то решение не найдено;
- lambda – множители Лагранжа, соответственно, для различных типов ограничений:
  - lambda.lower – для нижней границы lb;
  - lambda.upper – для верхней границы ub;
  - lambda.ineqlin – для линейных неравенств;
  - lambda.eqlin – для линейных равенств;
  - lambda.ineqnonlin – для нелинейных неравенств;
  - lambda.eqnonlin – для нелинейных равенств;
- output – информация о результатах оптимизации:
  - output.Iterations – число выполненных итераций;
  - output.funcCount – число вычислений функции;
  - output.algorithm – используемый алгоритм.

Пример 1. Рассмотрим следующий пример. Пусть некоторая замкнутая линейная динамическая система управления 3-го порядка описывается уравнениями

$x = (\mathbf{A} + \mathbf{BKC})x + \mathbf{B}u$,

$y = \mathbf{C}x$,

где матрицы

$$A = \begin{bmatrix} 0 & -2 & 10 \\ 0 & 1 & -2 \end{bmatrix}, B = \begin{bmatrix} -2 & 2 \\ 0 & 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

отражающие динамические свойства объекта управления, являются заданными, а матрица **К** регулятора – изменяемой.

Как известно, качество работы подобных систем (качество переходных процессов) определяется расположением на комплексной плоскости собственных чисел матрицы **А + ВКС**. Поставим задачу оптимизации таким образом: при задании диапазона возможных изменений элементов матрицы **К** от -4 до +4 подобрать эти элементы таким образом, чтобы указанные собственные числа равнялись величинам [–5, –3, –1].

Решение задачи, как и раньше, проведем поэтапно.

Этап 1. Создадим m-файл (с именем eigfun) для вычисления и упорядочивания по величине собственных чисел матрицы **А + ВКС**:

```
function F = eigfun(K,A,B,C) % Нахождение собственных чисел
F = sort(eig(A+B*K*C)); % Упорядочивание собственных чисел
```

Этап 2. Составление оптимизирующей программы:

```
>> A = [-0.5 0 0; 0 -2 10; 0 1 -2];
>> B = [1 0; -2 2; 0 1];
>> C = [1 0 0; 0 0 1];
>> K0 = [-1 -1; -1 -1]; % задание начальных условий
>> % задание вектора целей
>> goal = [-5 -3 -1];
>> weight = abs(goal) % задание вектора весовых коэффициентов
>> lb = -4*ones(size(K0)); % нижние границы элементов матрицы К
>> ub = 4*ones(size(K0)); % верхние границы элементов матрицы К
>> % установка опции вывода информации
>> options = optimset('Display','iter');
>> [K,fval,attainfactor] = fgoalattain(@eigfun,K0,...
   goal,weight,[],[],[],[],lb,ub,[],options,A,B,C)
  % Результаты вычислений:
```

Таблица 2.

Результат вычисления и упорядочивания по величине собственных чисел матрицы А + ВКС

Attainment			Directional		
Iter	F-count	factor	Step-size	derivative	Procedure
1	6	1.885	1	1.03	
2	13	1.061	1	-0.679	
3	20	0.4211	1	-0.523	Hessian modified
4	27	-0.06352	1	-0.053	Hessian modified twice
5	34	-0.1571	1	-0.133	
6	41	-	1	-	Hessia

		0.3489		0.00768	n modified
7	48	-0.3643	1	-4.25e-005	Hessian modified
8	55	-0.3645	1	-0.00303	Hessian modified twice
9	62	-0.3674	1	-0.0213	Hessian modified
10	69	-0.3806	1	0.00266	
11	76	-0.3862	1	-2.73e-005	Hessian modified twice
12	83	-0.3863	1	-1.2e-013	Hessian modified twice

Optimization terminated successfully:

Search direction less than 2*options.TolX and maximum constraint violation is less than options.TolCon

Active Constraints: 1 2 4 9 10

K = -4.0000 -0.2564

-4.0000 -4.0000

fval = -6.9313 -4.1588 -1.4099

attainfactor =-0.3863

Выведенная информация имеет следующий характер:

- Iter – номер итерации;
- F-count – количество вычислений функции;
- Attainment factor – коэффициент (уровень) достижения целей;
- Step-size – шаг поиска;
- Directional derivative – норма градиента;
- Procedure – выполненная процедура (Hessian modified – гессиан модифицирован, Hessian modified twice – гессиан модифицирован дважды);
- Optimization terminated successfully – оптимизация проведена успешно;
- конечное значение матрицы

K = -4.0000 -0.2564

-4.0000 -4.0000

- итоговые значения собственных чисел

fval = -6.9313 -4.1588 -1.4099

- итоговое значение коэффициента достижения целей attainfactor = -0.3863, что говорит о «перевыполнении» заданных целей в среднем на 38% (сравнение итоговых значений собственных чисел с заданными это подтверждает).

Полученные результаты не слишком близки к целевым. Поэтому для получения более приемлемого результата внесем коррекцию в программу оптимизации – зададим опцию точного достижения всех трех целей:

```
>> options = optimset('GoalsExactAchieve',3);
```

после чего повторим поиск оптимального решения:

```
>> [K,fval,attainfactor] = fgoalattain('eigfun', K0,goal, weight,[], [],[],[],lb,ub,[]
,options, A,B,C)
Optimization terminated successfully:
Magnitude of directional derivative in search direction less than
2*options.TolFun and maximum constraint violation is less than options.TolCon
Active Constraints: 9 10 12 14
K =     -1.5954      1.2040
        -0.4201     -2.9046
fval =  -5.0000
        -3.0000
        -1.0000
attainfactor = 0
```

Таким образом, результат является вполне удовлетворительным. Подробную демонстрацию приведенного примера можно получить, используя функцию goaldemo.

Литература

1. Аттетков А.В. Методы оптимизации: Учеб. для вузов / Аттетков А.В., Галкин С.В., Зарубин В.С; под ред. В.С. Зарубина, А.П. Крищенко [2-е изд.]. – МГТУ им. Н.Э. Баумана, 2003. – 440с.

2. Аоки М. Ведение в методы оптимизации. М.: Наука. 1977. 344с.

3. Табак Д., Куо Б. Оптимальное управление и математическое программирование. – М.: Наука, 1975. – 280с.

4. Банди Б. Методы оптимизации. Вводный курс. – М.: Радио и связь, 1988. – 128с.

5. Каханер Д., Моулер К., Нэш С. Численные методы и математическое обеспечение. – М.: Мир, 1998. – 575 с.

6. Васильков Ю. В., Василькова Н. Н. Компьютерные технологии вычислений в математическом моделировании. – М.: Финансы и статистика, 1999. – 256 с.

7. Дьяконов В., Круглов В. Математические пакеты расширения MATLAB. Специальный справочник. – СПб.: Питер, 2001. – 480 с. (Пакет Optimization Toolbox – с. 376-436.)

Марина Викторовна Маркина
Анна Владимировна Судакова

ПРАКТИКУМ ПО РЕШЕНИЮ ЗАДАЧ ОПТИМИЗАЦИИ В ПАКЕТЕ MATLAB

Учебно-методическое пособие

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского».
603950, Нижний Новгород, пр. Гагарина, 23.