ФЕДЕРАЛЬНОЕ АГЕНТСВО ПО ОБРАЗОВАНИЮ РОССИЙСКОЙ ФЕДЕРАЦИИ

УФИМСКИЙ ГОСУДАРСТВЕННЫЙ АВИАЦИОННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Общенаучный факультет

Кафедра ВВТиС

Отчет по лабораторным работам №4

**По дисциплине «Методы оптимизации»**

Группа ПМИ-440

Студент \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ Сайфутдинов Р.Ф.

(дата) (подпись) (Фамилия И.О.)

Проверил \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ Касаткин А. А.

(дата) (подпись) (Фамилия И.О.)

Уфа 2016

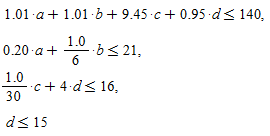
**Цель работы:** овладеть практическими навыками решения задач линейного программирования симплексным методом.

**Постановка задачи**

Решить задачу



где множество допустимых решений задается следующими условиями:



используя симплекс метод. Сравнить полученное значение с результатами решения задачи средствами Maple.

**Теоретическая часть**

Рассмотрим задачу поиска максимума функции

(4.1)

при следующих ограничениях

(4.2)

или в матричной форме

(4.3)

где .

– целевая функция,

– вектор стоимости,

– вектор ограничений,

– матрица условий.

Задача (4.3) называется канонической задачей линейного программирования.

В общей форме задача линейного программирования записывается в виде

(4.4)

Для перехода к задаче в канонической форме (4.3) необходимо перейти к задаче на максимум и ввести дополнительные координаты :

Точка выпуклого множества называется крайней, если не существует точек и числа , таких что . У многогранников крайние точки – вершины.

Задача (4.3) называется невырожденной, если любая крайняя точка множества допустимых решений (4.2) содержит ровно *m* положительных координат.

Пусть *x* – крайняя точка в невырожденной задаче (4.3) с первыми *m* (для определенности) положительными координатами. Тогда вектор *x* можно представить в виде – базисный вектор, – небазисный вектор. Аналогично матрицу A можно представить в виде .

*Алгоритм решения задач линейного программирования симплекс-методом*

1. Привести задачу к канонической форме
2. Отыскать крайнюю точку , множества допустимых элементов .
3. Построить симплексную таблицу для начальной крайней точки *x*.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | … |  |  | … |  | … |  | t |
| базис |  |  |  | … |  |  | … |  | … |  |  |
|  |  |  | 1 | … | 0 |  | … |  | … |  |  |
| … | … | … | … | … | … | … | … | … | … | … | … |
|  |  |  | 0 | … | 0 |  | … |  | … |  |  |
| … | … | … | 0 | … | 1 | … | … | … | … | … | … |
|  |  |  |  | … |  |  | … |  | … |  |  |
|  |  |  |  | … |  |  | … |  | … |  |  |
|  |  |  | 0 | … | 0 |  | … |  | … |  |  |

1. Исследовать симплексную таблицу.

а) Если вектор , то крайняя точка *x* – решение задачи.

б) Если для некоторого , то решение задачи (4.3) .

в) Пусть в строке имеются отрицательные числа, а соответствующие столбцы содержат положительные числа.

Предположим, что . Ясно, что . Столбец, соответствующий индексу называется разрешающим столбцом. Если достигается на нескольких значениях *j*, то в качестве разрешающего столбца выбираем столбец с любым таким индексом.

Обозначим . Эти значения ставим соответственно в последнем столбце симплексной таблицы.

Пусть . Строка вектора называется разрешающей. Если достигается на нескольких значениях *i*, то в качестве разрешающей строки выбираем любую такую строку. Элемент называется *разрешающим элементом* симплексной таблицы.

Далее из числа базисных векторов исключаем вектор , вместо него берем вектор . Значение функционала на новой крайней точке с новыми базисными векторами возрастет на величину .

1. Построить новую симплексную таблицу для нового базиса.

Способ построения новой таблицы по предыдущей (*правило прямоугольника*).

Далее пункт 4 и так далее, пока не придем к решению задачи.

1. *Метод Жордана-Гаусса* – сведение системы *m* уравнений с *n* неизвестными к каноническому виду при помощи элементарных операций над строками. При использовании первых *m* переменных такая система имеет вид

(4.5)

Переменные , входящие с единичными коэффициентами только в одно уравнение системы (4.5) и с нулевыми в остальные, называются базисными. В канонической системе каждому уравнению соответствует одна базисная переменная.

Базисным решением системы (4.5) называется решение, полученное при нулевых небазисных переменных.

Базисное решение называется допустимым базисным решением, если значения, входящих в него базисных переменных . Последнее условие эквивалентно условию .

Допустимое базисное решение является крайней точкой множества допустимых решений задачи (4.3).

*Двойственность задач линейного программирования*

Рассмотрим две задачи линейного программирования

(4.7)

(4.8)

Задача (4.8) является двойственной по отношению к задаче (4.7), и наоборот, задача (4.7) является двойственной по отношению к задаче (4.8) [3].

*Пример* [4]*.* Предприятие выпускает три вида продукции количества на трех различных типах установок.

Количество ресурса времени установки *j* для производства единицы продукции *i* определяетсявеличиной . Ресурс времени установок каждого типа ограничен величиной .

Прибыль от единицы каждого вида продукции . Необходимо максимизировать прибыль.

Двойственная задача будет иметь вид

Здесь – это оценка, соответствующая одной единице ограниченного ресурса по *i*-ой установке. Она равна величине, на которую могла бы увеличиться суммарная прибыль, если бы количество *i*-го ресурса увеличилось на единицу, и если это увеличение было бы использовано оптимально. Иными словами, – это количество прибыли, недополученной из-за нехватки единицы ограниченного ресурса .

**Решение задачи в пакете Maple**

**> **

**> **





**> **



**> **



**> **





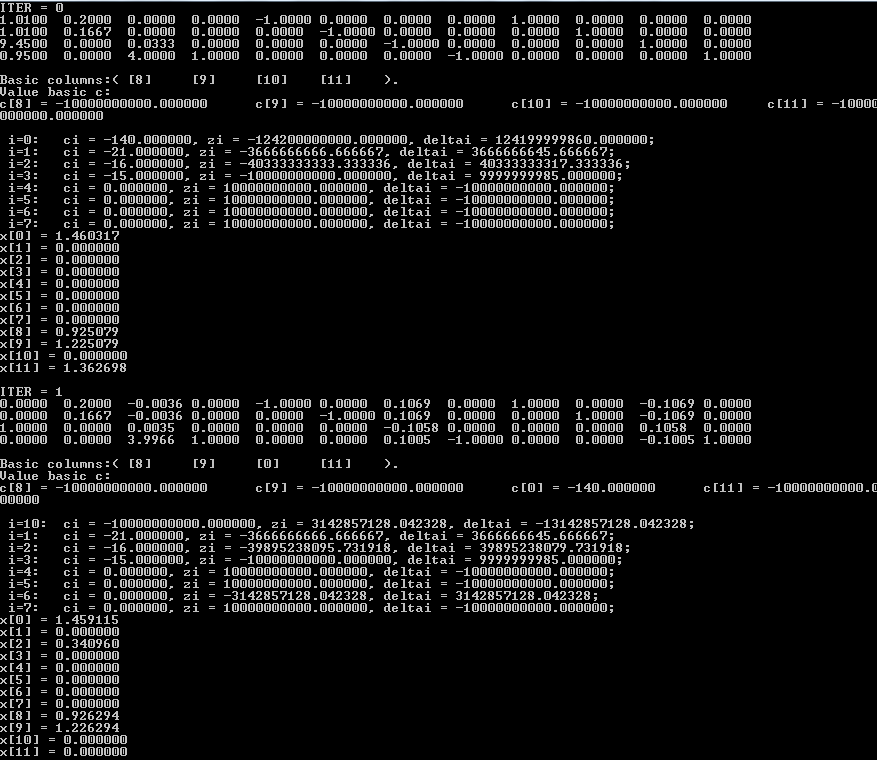
**> **



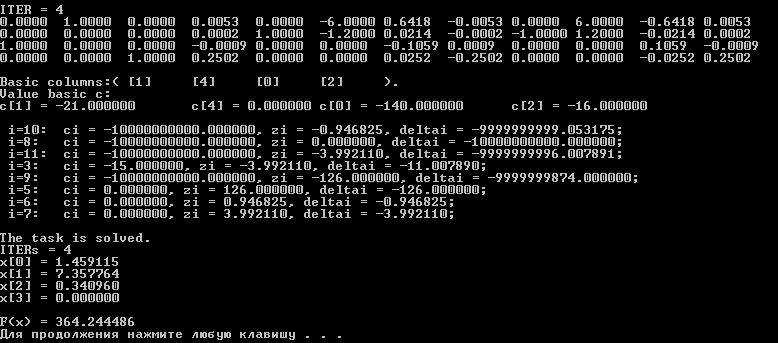
**> **



**Пример выполнения программы**

****

****

****

**Вывод:** в ходе данной лабораторной работы прошло знакомство с численными методами поиска условного экстремума действительной функции. Были изучены методы внешних штрафов, внутренних штрафов, а также комбинированный метод штрафов. Была проведен аналитический поиск минимума задачи условного экстремума, решение задачи средствами Maple, а также написана программа для численного нахождения условного минимума.

На основе полученных результатов можно сказать, что метод штрафов дает приближенное решение задачи условной минимизации.

**Листинг**

#include <stdio.h>

#include <stdint.h>

#include <inttypes.h>

#include <stdbool.h>

#include <math.h>

#include <stdlib.h>

#include <string.h>

#include <float.h>

#include <assert.h>

typedef enum Sign {

NOT\_GREATER\_THAN = 1,

EQUALITY = 0,

NOT\_LESS\_THAN = -1

} Sign\_t;

typedef enum Extreme {

MIN = -1,

MAX = 1

} Extreme\_t;

typedef struct Tableau {

uint32\_t M;

uint32\_t N;

double \*a;

double \*b;

double \*c;

Sign\_t \*sign;

Extreme\_t extr;

} Tableau\_t;

inline double f(const double \*c, const double \*x, const uint32\_t N);

void print\_a\_cb(const double \*mat, const double \*c, const uint32\_t \*col\_index\_b, const uint32\_t M, const uint32\_t N);

void print\_x(const double \*x, const uint32\_t N);

void copy\_c\_cond\_extr(double \*dst\_c, const double \*src\_c, const uint32\_t N, const Extreme\_t extreme);

int32\_t simplex(const Tableau\_t \*table, double \*x);

int32\_t main() {

double b[4] = { 140.0, 21.0, 16.0, 15.0 };

double c[4] = { 2.4, 2.7, 13.8, 2.75 };

Extreme\_t extr = MIN;

Tableau\_t table;

table.M = table.N = 4;

table.extr = extr;

if (extr == MAX) {

double mat[4 \* 4] = { 1.01, 1.01, 9.45, 0.95,

0.2, 1.0 / 6.0, 0.0, 0.0,

0.0, 0.0, 1.0 / 30.0, 4.0,

0.0, 0.0, 0.0, 1.0 };

Sign\_t sign[4] = { NOT\_GREATER\_THAN, NOT\_GREATER\_THAN, NOT\_GREATER\_THAN, NOT\_GREATER\_THAN };

table.a = mat;

table.b = b;

table.c = c;

table.sign = sign;

} else {

double mat[4 \* 4] = { 1.01, 0.2, 0.0, 0.0,

1.01, 1.0 / 6.0, 0.0, 0.0,

9.45, 0.0, 1.0 / 30.0, 0.0,

0.95, 0.0, 4.0, 1.0 };

Sign\_t sign[4] = { NOT\_LESS\_THAN, NOT\_LESS\_THAN, NOT\_LESS\_THAN, NOT\_LESS\_THAN };

table.a = mat;

table.b = c;

table.c = b;

table.sign = sign;

}

double \*x = (double \*)calloc(table.N, sizeof(double));

simplex(&table, x);

print\_x(x, table.M);

printf("F(x) = %f\n", f(table.c, x, table.N));

system("pause");

free(x);

return 0;

}

inline double f(const double \*c, const double \*x, const uint32\_t N) {

double res = 0.0;

for (uint32\_t i = 0; i < N; ++i) {

res += c[i] \* x[i];

}

return res;

}

void print\_a\_cb(const double \*mat, const double \*c, const uint32\_t \*col\_index\_b, const uint32\_t M, const uint32\_t N) {

const double \*mat\_ = mat;

for (uint32\_t i = 0; i < M; ++i) {

for (uint32\_t j = 0; j < N; ++j) {

printf("%.4f\t", \*mat\_++);

}

printf("\n");

}

printf("\nBasic columns:(\t");

for (uint32\_t i = 0; i < M; ++i) {

printf("[%" PRIu32 "]\t", col\_index\_b[i]);

}

printf(").\nValue basic c:\n");

for (uint32\_t i = 0; i < M; ++i) {

const uint32\_t kk = col\_index\_b[i];

printf("c[%" PRIu32 "] = %lf\t", kk, c[kk]);

}

printf("\n\n");

return;

}

void print\_x(const double \*x, const uint32\_t N) {

for (uint32\_t i = 0; i < N; ++i) {

printf("x[%" PRIu32 "] = %f\n", i, x[i]);

}

printf("\n");

return;

}

// if extreme = MIN then c\_i = -ci

void copy\_c\_cond\_extr(double \*dst\_c, const double \*src\_c, const uint32\_t N, const Extreme\_t extreme) {

double \*p = dst\_c;

const double \*q = src\_c;

for (uint32\_t i = 0; i < N; ++i) {

\*p++ = \*q++ \* extreme;

}

return;

}

int32\_t simplex(const Tableau\_t \*table, double \*x) {

static const double sufficiently\_large\_number = 1.0e+10;

const uint32\_t N = table->N;

const uint32\_t M = table->M;

const uint32\_t K = M; // in the number inequality

const uint32\_t N\_ext = N + K + M;

const uint32\_t offset\_b = N\_ext - M;

const double \*b = table->b;

for (uint32\_t i = 0; i < N; ++i) {

assert(b[i] >= 0);

}

double \*a\_ = (double \*)calloc(M \* N\_ext, sizeof(double));

double \*c\_ = (double \*)malloc(N\_ext \* sizeof(double));

double \*x\_ = (double \*)calloc(N\_ext, sizeof(double));

// col\_index - the room basic and not basic columns

// col\_index[0 .. N\_ext - M - 1] - not basic; col\_index[N\_ext - M .. N\_ext - 1] - basic

uint32\_t \*col\_index = (uint32\_t \*)malloc(N\_ext \* sizeof(uint32\_t));

uint32\_t \*col\_index\_nb = col\_index;

uint32\_t \*col\_index\_b = col\_index + offset\_b;

memcpy(x\_, x, N \* sizeof(double));

copy\_c\_cond\_extr(c\_, table->c, N, table->extr);

for (uint32\_t i = 0; i < M; ++i) {

memcpy(a\_ + i \* N\_ext, table->a + i \* N, N \* sizeof(double));

a\_[i \* N\_ext + i + N] = (double)(table->sign[i]);

a\_[i \* N\_ext + i + offset\_b] = 1.0;

col\_index[i] = i;

}

for (uint32\_t i = M; i < N\_ext; ++i) {

if (i >= N && i < offset\_b) {

c\_[i] = 0.0;

} else {

c\_[i] = -sufficiently\_large\_number;

}

col\_index[i] = i;

}

for (uint32\_t i = offset\_b; i < N\_ext; ++i) {

x\_[i] = b[i - offset\_b];

}

uint32\_t iter;

const uint32\_t max\_iter = N\_ext + 1;

for (iter = 0; iter < max\_iter; ++iter) {

printf("ITER = %" PRId32 "\n", iter);

print\_a\_cb(a\_, c\_, col\_index\_b, M, N\_ext);

uint32\_t index\_r = 0;

bool isExitDelta = true;

double max\_delta = 0.0;

for (uint32\_t i = 0; i < offset\_b; ++i) {

double z = 0.0;

const uint32\_t ii = col\_index\_nb[i];

for (uint32\_t k = 0; k < M; ++k) {

const uint32\_t kk = col\_index\_b[k];

z += c\_[kk] \* a\_[k \* N\_ext + ii];

}

const double delta\_i = c\_[ii] - z;

printf(" i=%" PRIu32 ":\tci = %lf, zi = %lf, deltai = %lf;\n", ii, c\_[ii], z, delta\_i);

if (delta\_i > max\_delta) {

isExitDelta = false;

max\_delta = delta\_i;

index\_r = i;

}

}

if (isExitDelta) {

printf("\nThe task is solved.\n");

break;

}

const uint32\_t col\_r = col\_index\_nb[index\_r];

uint32\_t row\_s = 0;

double min\_delta = DBL\_MAX;

for (uint32\_t i = 0; i < M; ++i) {

const double a\_ir = a\_[i \* N\_ext + col\_r];

if (a\_ir <= 0) continue;

const double tmp = x\_[col\_index\_b[i]] / a\_ir;

if (tmp < min\_delta) {

min\_delta = tmp;

row\_s = i;

} else if (tmp == min\_delta) {

row\_s = row\_s > i ? row\_s : i;

}

}

const uint32\_t col\_s = col\_index\_b[row\_s];

const double x\_r = min\_delta;

double \*a\_s = a\_ + row\_s \* N\_ext;

const double tmp = 1.0 / \*(a\_s + col\_r);

for (uint32\_t i = 0; i < N\_ext; ++i) {

\*(a\_s + i) \*= tmp;

}

for (uint32\_t i = 0; i < M; ++i) {

if (i == row\_s) continue;

double \*a\_i = a\_ + i \* N\_ext;

const double factor = \*(a\_i + col\_r); // a\_[s,r] = 1

x\_[col\_index\_b[i]] -= factor \* x\_r;

for (uint32\_t j = 0; j < N\_ext; ++j) {

\*(a\_i + j) -= \*(a\_s + j) \* factor;

}

}

col\_index\_b[row\_s] = col\_r;

col\_index\_nb[index\_r] = col\_s;

x\_[col\_s] = 0.0;

x\_[col\_r] = x\_r;

print\_x(x\_, N\_ext);

}

if (iter == max\_iter) {

printf("\nWarning: MAX ITER!\n");

}

printf("ITERs = %" PRIu32 "\n", iter);

memcpy(x, x\_, N \* sizeof(double));

free(a\_);

free(c\_);

free(x\_);

return 0;

}