МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «КубГУ»)**

**Факультет математики и компьютерных наук**

**Кафедра информационных образовательных технологий**

**ИНДИВИДУАЛЬНАЯ ЗАДАНИЕ**

**на тему «Реализация метода Кармаркара для решения задачи линейного программирования в языке C»**

Направление подготовки: **01.04.01 Математика**

Направленность (профиль): «**Преподавание математики/информатики»**

Работу выполнил

студент группы 201.1 Щёлоков Д. А.

(подпись, дата)

Работу принял

доц., канд. тех. наук Алексеев Е. Р.

(подпись, дата)

Краснодар

2025

**Постановка задачи линейного программирования (ЗЛП)**

#### ****Стандартная форма задачи ЛП:****

**Дано:**

1. **Целевая функция** (линейная):
2. **Ограничения** (линейные уравнения/неравенства):

**Требуется:**  
Найти значения переменных , при которых целевая функция Z достигает максимума (или минимума) с учетом ограничений.

Z

**Пример ЗЛП:**

**Дана целевая функция:**

**и система ограничений:**

Для решения ЗЛП необходимо избавиться от неравенств в системе. Для этого вводят дополнительные переменные, которые обращают неравенство в равенство, приводя ЗЛП к стандартной форме.

**Метод Кармаркара**

Для решения ЗЛП методом Кармаркара необходимо, чтобы система ограничений была представлена в «специальной форме»:

Здесь все ограничения представлены в виде однородных уравнения, кроме ограничения , которое определяет n-мерный правильный симплекс. Обоснованность алгоритма Кармаркара основана на выполнеии двух условий:

1. Вектор удовлетворяет ограничениям ;

2. min Z = 0;

Рассмотрим алгоритм преобразования ЗЛП в задачу со специальной формой, который будет использован в программной реализации для решения ЗЛП методом Кармаркара.

**Пусть дана следующая ЗЛП:**

1. С помощью переменной преобразуем ограничение в равенство: .

2. Введём новое ограничение , где U – достаточно большое положительное число, не удаляющее ни одной допустимой точки исходного пространства решений. Из равенства выше следует, что U можно взять равным 5.

3. Вводим ещё одну новую переменную и получаем . Такое условие необходимо, чтобы сделать ограничения системы однородными.

4. Умножим правую часть уравнения на , после приведения подобный членов получим однородное условие: .

5. Преобразуем равенство в уравнение, определяющее симплекс, для этого введём новые переменны . После замены переменных получаем следующую задачу:

6. Теперь необходимо, чтобы точка была центром симплекса и будет удовлетворять однородному уравнению. Для этого от левой части каждого однородного уравнения отнимем искуственную переменную с коэффициентом, равным алгебраической сумме всех коэффициентов левой части (в нашем случае 3 + 8 + 3 - 2 = 12). Эта искуственная переменная также прибавляется к уравенияю симплекса (), а также в виде штрафа появляется в выражении целевой функции. Преобразованная ЗЛП имеет вид:

где число M должно быть достаточно большим, чтобы привести переменную к нулевому значению.

Таким образом, любую ЗЛП можно привести к специальной форме для алгоритма Кармаркара.

**Этапы алгоритма Кармаркара**

Принципиальная идея Кармаркара заключается в том, что вычисление начинается с внутренней точки, соответствующей центру симплекса, в отличие от симплекс-метода, который проходит по вершинам симплекса.

1. После выбора внутренней точки в направлении *проекции градиента* определяется новая точка решения, которая должна быть строго внутренней (все координаты положительны). Это необходимое условие сходимости.

2. Чтобы новая точка решения была внутренней, построим вписанную в симплекс сферу. В n-мерном пространстве в правильный симплекс можно вписать сферу с максимальным радиусом . Тогда пересечение сферы радиуса и пространства решений, определяемого системой , будет содержать только внутренние точки пространства решений с положительными координатами.

3. Новая точка решения не обязаны быть центром симплекса. Поэтому, чтобы сделать процесс итерационным, необходимо перенести новую точку решения в центр симплекса. Для этого воспользуемся идеей *проективных преобразований*:

где - i-й элемент текущего решения, т. е. I-я координата текущей точки решения .

В матричном виде проективное преобразование можно переписать как:

,

где - диагональная матрица, у которой i-й диагональны элемент навен .

*Обратное преобразование*:

4. По определению . Отсюда следует, что значение должно быть положительным. Таким образом исходная ЗЛП преобразуется в следующую задачу:

5. Преобразованная задача имеет тот же тип, поэтому мы можем опять начать решение этой задачи с точки , являющейся центром симплекса. После каждой итерации метода можно нетрудно получить значения исходных X-переменных с помощью обратного преобразования.

6. Теперь покажем, как определить новую точку решения. На k-ой итерации задача имеет вид:

Новое значение Y вычисляется по следущей формуле:

,

где , - проекция градиента, которую можно вычислить так:

,

где , .

В качестве параметра Кармаркар предлагал использовать значение

**Реализация программы алгоритма Кармаркара на языке C**

Все преобразования выше реализованы на языке C с использованием динамических одномерный и двумерных массивов, выполнена реализация считывания матриц ограничений-неравенств и ограничений-равенств, векторов правых частей и целевой функции.

В качестве компилятора используется компилятор gcc версия 10.3.0 (tdm64-1).

Каждое громоздкое преобразование или действие, которое вызывается постоянно, выведено в отдельную функцию. Было решено реализовать следующие ***функции***:

* int empty\_file(const char \*filename) — функция проверки текстового файла на пустоту;
* double\* vector(int n) — функция задания одномерного динамического массива n-переменных типа double;
* double\*\* matrix(int n, int m) — функция задания двумерного динамического массива размера (nxm) типа double;
* void free\_matr(double \*\*A, int n) – функция очистки памяти от двумерного динамического массива;
* void ReadSizeMatr(char \*path, int \*n, int \*m) — функция определения размерности матрицы, заданной в файле;
* double\*\* ReadMatrix(char \*path, int n, int m) — функция считывания матрицы из текстового файла по его имени *path*;
* double\* ReadVector(char \*path, int n) – функция считывания вектора из текстового файла по его имени *path*;
* double \*MultV\_M(double \*x, int p, double \*\*A, int n, int m) – функция умножения вектора на матрицу;
* double \*\*MultM\_M(double \*\*A, int n1, int m1, double \*\*B, int n2, int m2) — функция умножения двух матриц;
* void Print\_Matr(double \*\*A, int n, int m) — функция выведения элементов матрицы в консоль;
* void Print\_Vect(double \*x, int n) — функция выведения элементов вектора в консоль;
* double \*\*A\_T(double \*\*A, int n, int m) — функция транспонирования матрицы;
* double \*C(char \*path, int n, int \*f) — функция считывания целевой функции из файла *path*;
* double NormVect(double \*x, int n) – функция нормы вектора ;
* double \*\*inverseMatrix(double \*\*A, int n) — функция нахождения обратной матрицы;
* double \*proj\_tranform(double \*c, double \*\*A, double \*\*D, double \*x, int n, int m, double r, double alf) — функция проективного преобразования, возвращает вектор нового решения;
* double \*Karmarkar\_metod(double \*c, double \*\*A, int n, int m, int maxiter, double eps, int \*k) — функция, принимающая вектор целевой функции, матрицу системы ограничений, представленных в специальной форме, maxiter – максимальное число итераций алгоритма, eps – точность, k – число пройденых итераций;
* void Karmarkar(char \*c\_path, char \*A\_path, char \*b\_path, char \*Aeq\_path, char \*beq\_path, int maxiter, double eps, int \*k) — функция преобразования исходных ограничений ЗЛП и целевой функции в специальную форму, для последующего решения ЗЛП методом Кармаркара. Принимает на вход пути к текстовым файлам, содержащим целевую функцию и ограничения, maxiter – максимальное число итераций алгоритма, eps – точность, k – число пройденых итераций.

**Листинг программы Karmarkar.c**

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <math.h>

#include <sys/stat.h>

int empty\_file(const char \*filename) {

struct stat st;

if (stat(filename, &st) == -1) {

//printf("Файла %s не существует/ошибка чтения\n",filename);

return -1; // Ошибка чтения файла

}

if (st.st\_size == 0){

//printf("Файл %s пуст!",filename);

return 1;

}

else{

//printf("Файл %s не пуст!",filename);

return 0;

}

}

double\* vector(int n){

return (double\*)malloc(n\*sizeof(double));

}

double\*\* matrix(int n, int m){

double \*\*A = (double\*\*)malloc(n\*sizeof(double\*));

for (int i=0;i<n;i++)

A[i] = (double\*)calloc(m,sizeof(double));

return A;

}

void free\_matr(double \*\*A, int n){

int i;

for (i=0;i<n;i++)

free(A[i]);

free(A);

}

void ReadSizeMatr(char \*path, int \*n, int \*m){

FILE \*file;

int nn,mm;

file = fopen(path,"r");

fscanf(file,"%d",&nn);

fscanf(file,"%d",&mm);

\*n = (int)nn;

\*m = (int)mm;

fclose(file);

}

double\*\* ReadMatrix(char \*path, int n, int m){

int i,j;

FILE \*file;

file = fopen(path,"r");

int nn,mm;

fscanf(file,"%d",&nn); //костыль для считывания с 3 элемента

fscanf(file,"%d",&mm);

double \*\*A = matrix(n,m);

double a;

for (i=0;i<n;i++)

for (j=0;j<m;j++){

fscanf(file,"%lf",&a);

A[i][j] = a;

}

fclose(file);

return A;

}

double\* ReadVector(char \*path, int n){

int i;

FILE \*file;

double \*x =vector(n);

double a;

file = fopen(path,"r");

for (i=0;i<n;i++){

fscanf(file,"%lf",&a);

x[i] = a;

}

fclose(file);

return x;

}

double \*MultV\_M(double \*x, int p, double \*\*A, int n, int m){

int j,k;

double \*Mult = vector(m);

for (j=0;j<p;j++)

Mult[j] = 0;

for (j=0;j<m;j++)

for (k=0;k<n;k++)

Mult[j] += x[k]\*A[k][j];

return Mult;

}

double \*\*MultM\_M(double \*\*A, int n1, int m1, double \*\*B, int n2, int m2){

int i, j, k;

double \*\*Mult = matrix(n1,m2);

for (i=0;i<n1;i++)

for (j=0;j<m2;j++){

Mult[i][j] = 0;

for (k=0;k<m1;k++)

Mult[i][j] += A[i][k]\*B[k][j];

}

return Mult;

}

void Print\_Matr(double \*\*A, int n, int m){

int i,j;

printf("\n");

for (i=0;i<n;i++){

printf("|");

for (j=0;j<m;j++)

printf(" %7.3lf ",A[i][j]);

printf("|\n");

}

}

void Print\_Vect(double \*x, int n){

int i;

printf(" {");

for (i=0;i<n;i++)

printf(" %7.3lf ",x[i]);

printf("}");

}

double \*\*A\_T(double \*\*A, int n, int m){

int i,j;

double \*\*A\_T = matrix(m,n);

for (i=0;i<n;i++)

for (j=0;j<m;j++)

A\_T[j][i] = A[i][j];

return A\_T;

}

double \*C(char \*path, int n, int \*f){

int i;

double \*c = vector(n);

if (!(empty\_file(path))){

char minmax;

double a;

FILE \*file;

file = fopen(path,"r");

fscanf(file,"%c",&minmax); //считывание из файла с целевой функцией первого символа: + или -

for (i=0;i<n;i++){

fscanf(file,"%lf",&a);

c[i] = a;

}

printf("\n\n\*Целевая функция:\*");

Print\_Vect(c,n);

if ((int)minmax == 43){

printf(" --> max\n"); //ANCII-код символа +: 43

for (i=0;i<n;i++)

c[i] = -c[i];

\*f = 1;

}

else{

printf(" --> min\n");

\*f = 0;

}

}

else{

printf("Файл целевой функции %c пуст!",\*path);

}

return c;

}

double NormVect(double \*x, int n){

int i;

double d=0;

for (i=0;i<n;i++)

d += x[i]\*x[i];

return sqrt(d);

}

double \*\*inverseMatrix(double \*\*A, int n) {

int i, j, k;

// Создаем расширенную матрицу [A|I]

double \*\*AI = (double \*\*)malloc(n \* sizeof(double \*));

for (i = 0; i < n; i++) {

AI[i] = (double \*)malloc(2 \* n \* sizeof(double));

// Копируем исходную матрицу

for (j = 0; j < n; j++) {

AI[i][j] = A[i][j];

}

// Добавляем единичную матрицу

for (j = n; j < 2 \* n; j++) {

AI[i][j] = (i == (j - n)) ? 1.0 : 0.0;

}

}

// Прямой ход метода Гаусса

for (i = 0; i < n; i++) {

// Если элемент на диагонали равен 0, находим строку для замены

if (AI[i][i] == 0.0) {

int swap\_row = -1;

for (k = i + 1; k < n; k++) {

if (AI[k][i] != 0.0) {

swap\_row = k;

break;

}

}

// Меняем строки местами

double \*temp = AI[i];

AI[i] = AI[swap\_row];

AI[swap\_row] = temp;

}

// Нормируем текущую строку

double M = AI[i][i];

for (j = i; j < 2 \* n; j++) {

AI[i][j] /= M;

}

// Вычитаем текущую строку из других строк

for (k = 0; k < n; k++) {

if (k != i && AI[k][i] != 0.0) {

double factor = AI[k][i];

for (j = i; j < 2 \* n; j++) {

AI[k][j] -= factor \* AI[i][j];

}

}

}

}

// Извлекаем обратную матрицу из правой части расширенной матрицы

double \*\*inverse = matrix(n,n);

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

inverse[i][j] = AI[i][j + n];

}

}

// Освобождаем память расширенной матрицы

free\_matr(AI,n);

return inverse;

}

double \*proj\_tranform(double \*c, double \*\*A, double \*\*D, double \*x, int n, int m, double r, double alf){ //проективное преобразование

double \*\*C = matrix(1,m);

double \*\*AD = matrix(n,m);

double \*\*P = matrix(n+1,m);

double \*\*PT = matrix(m,n+1);

double \*\*P\_PT = matrix(n+1,n+1);

double \*\*P\_PT\_1 = matrix(n+1,n+1);

double \*\*P1 = matrix(n+1,m);

double \*\*P2 = matrix(m,m);

double \*\*I = matrix(m,m);

double \*\*I\_P = matrix(m,m);

double \*\*cD = matrix(1,m);

double \*\*cDT = matrix(m,1);

double \*\*cp = matrix(m,1);

double \*cp\_n = vector(m);

double \*Y = vector(m);

double \*X = vector(m);

double \*DY = vector(m);

int i,j;

//Print\_Vect(c,m);

//Print\_Matr(A,n,m);

//Print\_Matr(D,m,m);

//Print\_Vect(x,m);

//printf("\n");

for (i=0;i<m;i++){

C[0][i] = c[i];

}

cD = MultM\_M(C,1,m,D,m,m);

//Print\_Matr(cD,1,m);

AD = MultM\_M(A,n,m,D,m,m);

//Print\_Matr(AD,n,m);

for (i=0;i<(n+1);i++)

for (j=0;j<m;j++){

if (i == n) P[i][j] = 1;

else P[i][j] = AD[i][j]; //Расширенная матрица с дополнительной единичной строкой

}

for (i=0;i<m;i++)

for (j=0;j<(n+1);j++)

PT[i][j] = P[j][i]; //

//Print\_Matr(P,n+1,m);

//Print\_Matr(PT,m,n+1);

P\_PT = MultM\_M(P,n+1,m,PT,m,n+1); //

P\_PT\_1 = inverseMatrix(P\_PT,n+1); //

//Print\_Matr(P\_PT,n+1,n+1);

//Print\_Matr(P\_PT\_1,n+1,n+1);

P1 = MultM\_M(P\_PT\_1,n+1,n+1,P,n+1,m); //

P2 = MultM\_M(PT,m,n+1,P1,n+1,m); //

//Print\_Matr(P2,m,m);

for (i=0;i<m;i++)

for (j=0;j<m;j++){

I[i][j] = (i == j)? 1: 0; //Единичная матрица

I\_P[i][j] = I[i][j]-P2[i][j]; //Проекционная матрица сохраняет выполнение равенства Ax = 0

} //поправляет значения переменных, чтобы точка не выходила на границы допустимой области

//Print\_Matr(I,m,m);

//Print\_Matr(I\_P,m,m);

for (i=0;i<m;i++)

cDT[i][0] = cD[0][i];

//Print\_Matr(cDT,m,1);

cp = MultM\_M(I\_P,m,m,cDT,m,1); //Проекция градиента, в направлении которого

//Print\_Matr(cp,m,1); //определяется новая точка решения

for (i=0;i<m;i++)

cp\_n[i] = cp[i][0]; //

//Print\_Vect(cp\_n,m);

double Norm = NormVect(cp\_n,m);

//printf("%f",Norm);

double \*x0 = vector(m);

double sumDY = 0;

for (i=0;i<m;i++){

x0[i] = 1.0/m;

Y[i] = x0[i] - r\*alf\*(1/Norm)\*cp\_n[i]; //

DY[i] = D[i][i]\*Y[i];

sumDY += DY[i];

}

//Print\_Vect(DY,m);

//printf("%f",sumDY);

for (i=0;i<m;i++)

X[i] = (DY[i])/sumDY;

/\*Освобождение памяти\*/

free\_matr(C,1);

free\_matr(AD,n);

free\_matr(P,n+1);

free\_matr(PT,m);

free\_matr(P\_PT,n+1);

free\_matr(P\_PT\_1,n+1);

free\_matr(P1,n+1);

free\_matr(P2,m);

free\_matr(I,m);

free\_matr(I\_P,m);

free\_matr(cD,1);

free\_matr(cDT,m);

free\_matr(cp,m);

free(cp\_n);

free(Y);

free(DY);

return X;

}

double \*Karmarkar\_metod(double \*c, double \*\*A, int n, int m, int maxiter, double eps, int \*k){

int i, j;

double r, z0 = 0, z = 0, alf;

double \*x0 = vector(m);

double \*X = vector(m);

double \*\*D = matrix(m,m);

double \*xx = vector(m);

//printf("%d ",\*k);

while (\*k <= maxiter){

//printf("%d ",\*k);

if (\*k == 0){

//printf("\n\*Вектора текущего приближения переменных на каждой итерации\*\n\nИтерация %d\n",\*k);

for (i=0;i<m;i++)

for (j=0;j<m;j++){

x0[j] = 1.0/m; //Начальная точка - центр симплекса x = (1/m,1/m,...,1/m)

D[i][j] = (i==j)? x0[j] : 0; //Диагональная матрица масштабирования

}

//printf("Матрица D:");

//Print\_Matr(D,m,m);

r = 1.0/sqrt(m\*(m-1)); //Радиус сферы, вписанной в симплекс

alf = (m-1)/(3.0\*m); //Параметр выбирается для сходимости решения

for (j=0;j<m;j++)

z0 += c[j]\*x0[j]; //начальное значение целевой функции

z = 1.1\*z0;

//printf("%f, %f, %f, %f",r,alf,z, z0);

for (i=0;i<m;i++)

X[i] = 0;

\*k += 1;

}

else{

for (i=0;i<m;i++)

xx[i] = X[i]-x0[i];

if ((fabs(z-z0)>eps) || (NormVect(xx,m)>eps)){

//printf("\n\nИтерация [%d]:",\*k);

z0 = z;

for (i=0;i<m;i++)

x0[i] = X[i];

X = proj\_tranform(c,A,D,x0,n,m,r,alf);

//Print\_Vect(X,m);

for (i=0;i<m;i++)

for (j=0;j<m;j++){

D[i][j] = (i==j)? X[j] : 0;

}

//printf("Матрица D:");

//Print\_Matr(D,m,m);

z = 0;

for (j=0;j<m;j++)

z += c[j]\*X[j]; //Новое значение функции на текущем шаге

//printf("\nz(x) = %f",z);

\*k += 1;

}

else break;

}

}

return X;

}

void Karmarkar(char \*c\_path, char \*A\_path, char \*b\_path, char \*Aeq\_path, char \*beq\_path, int maxiter, double eps, int \*k, double U){

int i,j;

int ni,mi;

int ne,me;

int n\_new,m\_new;

int f = 0; //флаг результата работы программы

if ((empty\_file(A\_path)) && (empty\_file(Aeq\_path))){

printf("Файлы %s и %s пустые!\n", A\_path, Aeq\_path);

f = -1; // Конец программы

}

else{

if (!(empty\_file(A\_path)) && (empty\_file(Aeq\_path))){

if (!(empty\_file(b\_path))){

printf("Дана следующая задача линейного программирования:");

ne = 0;

me = 0;

ReadSizeMatr(A\_path,&ni,&mi);

double \*\*A = matrix(ni,mi);

double \*b = vector(ni);

A = ReadMatrix(A\_path,ni,mi);

b = ReadVector(b\_path,ni);

printf("\n\n\*Ограничения-неравенства:\*");

Print\_Matr(A,ni,mi);

printf("\n\*Вектор правых частей:\*");

Print\_Vect(b,ni);

n\_new = ni + ne;

m\_new = ni + mi + 2;

double \*\*A\_new = matrix(n\_new,m\_new);

double \*b\_new = vector(n\_new);

double \*c = vector(max(mi,me));

double \*c\_new = vector(m\_new);

for (i=0;i<n\_new;i++){

for (j=0;j<m\_new;j++){

if ((i<ni)&&(j<mi))

A\_new[i][j] = A[i][j];

if ((i<ni)&&(mi<=j)&&(j<(m\_new-1)))

A\_new[i][j] = (i == (j-mi))? 1.0 : 0.0;

}

if (i<ni)

b\_new[i] = b[i];

}

int sign = 0;

c = C(c\_path,max(mi,me),&sign);

for (j=0;j<m\_new;j++)

c\_new[j] = (j <max(mi,me))? c[j] : 0;

Print\_Matr(A\_new,n\_new,m\_new);

Print\_Vect(c\_new,m\_new);

double max\_s = -1e10;

double \*s = vector(n\_new);

for (i=0;i<n\_new;i++){

s[i] = 0;

for (j=0;j<(m\_new-2);j++)

s[i] += A\_new[i][j];

if (s[i]>=max\_s)

max\_s = s[i];

}

double U = abs(max\_s)+1;

printf("\n%f\n",U);

for (i=0;i<n\_new;i++){

for (j=0;j<(m\_new-1);j++){

A\_new[i][j] = U\*A\_new[i][j]-b\_new[i];

}

}

for (j=0;j<m\_new;j++)

c\_new[j] = U\*c\_new[j];

Print\_Matr(A\_new,n\_new,m\_new);

Print\_Vect(c\_new,m\_new);

double \*sum\_i = vector(n\_new);

for (i=0;i<n\_new;i++){

sum\_i[i] = 0;

for (j=0;j<(m\_new-1);j++)

sum\_i[i] += A\_new[i][j];

A\_new[i][m\_new-1] = - sum\_i[i];

}

double sum\_j = 0;

for (j=0;j<(m\_new-1);j++)

sum\_j += c\_new[j];

c\_new[m\_new-1] = -(sum\_j+20);

Print\_Matr(A\_new,n\_new,m\_new);

Print\_Vect(c\_new,m\_new);

double \*y\_opt = vector(m\_new);

y\_opt = Karmarkar\_metod(c\_new,A\_new,n\_new,m\_new,maxiter,eps,k);

Print\_Vect(y\_opt,m\_new);

double \*x\_opt = vector(max(mi,me));

double z\_opt = 0;

for (i=0;i<max(mi,me);i++){

x\_opt[i] = U\*y\_opt[i];

z\_opt += x\_opt[i];

}

printf("\n\nНайденное оптимальное решение: x =");

Print\_Vect(x\_opt,max(mi,me));

printf("\nОртимальное значение целевой функции: z =%f",z\_opt);

f = 2;

}

else{

printf("Файл %s пуст!\n", b\_path);

f = -2;

}

}

if ((empty\_file(A\_path)) && !(empty\_file(Aeq\_path))){

if (!(empty\_file(beq\_path))){

printf("Дана следующая задача линейного программирования:\n");

ni = 0;

mi = 0;

ReadSizeMatr(Aeq\_path,&ne,&me);

double \*\*Aeq = matrix(ne,me);

double \*beq = vector(ni);

Aeq = ReadMatrix(Aeq\_path,ne,me);

beq = ReadVector(beq\_path,ne);

printf("\n\n\*Ограничения-равенства:\*");

Print\_Matr(Aeq,ne,me);

printf("\n\*Вектор правых частей:\*");

Print\_Vect(beq,ne);

f = 3;

}

else{

printf("Файл %s пуст!\n", beq\_path);

f = -3;

}

}

}

if (f == 0){

printf("Дана следующая задача линейного программирования:\n");

ReadSizeMatr(A\_path,&ni,&mi);

ReadSizeMatr(Aeq\_path,&ne,&me);

double \*\*A = matrix(ni,mi);

double \*b = vector(ni);

double \*\*Aeq = matrix(ne,me);

double \*beq = vector(ni);

A = ReadMatrix(A\_path,ni,mi);

b = ReadVector(b\_path,ni);

Aeq = ReadMatrix(Aeq\_path,ne,me);

beq = ReadVector(beq\_path,ne);

printf("\n\n\*Ограничения-неравенства:\*");

Print\_Matr(A,ni,mi);

printf("\n\*Вектор правых частей:\*");

Print\_Vect(b,ni);

printf("\n\n\*Ограничения-равенства:\*");

Print\_Matr(Aeq,ne,me);

printf("\n\*Вектор правых частей:\*");

Print\_Vect(beq,ne);

n\_new = ni + ne;

m\_new = ni + mi + 2;

//printf("\n Новая размерность A\_new: %d,%d\n",n\_new,m\_new);

double \*\*A\_new = matrix(n\_new,m\_new);

double \*b\_new = vector(n\_new);

double \*c = vector(max(mi,me));

double \*c\_new = vector(m\_new);

for (i=0;i<n\_new;i++){

for (j=0;j<m\_new;j++){

if ((i<ni)&&(j<mi))

A\_new[i][j] = A[i][j];

if ((i<ni)&&(mi<=j)&&(j<(m\_new-1)))

A\_new[i][j] = (i == (j-mi))? 1.0 : 0.0;

if ((ni<=i)&&(j<mi))

A\_new[i][j] = Aeq[i-ni][j];

}

if (i<ni)

b\_new[i] = b[i];

if ((ni<=i)&&(i<n\_new))

b\_new[i] = beq[i-ni];

}

int sign = 0;

c = C(c\_path,max(mi,me),&sign);

for (j=0;j<m\_new;j++)

c\_new[j] = (j <max(mi,me))? c[j] : 0;

//Print\_Matr(A\_new,n\_new,m\_new);

//Print\_Vect(c\_new,m\_new);

double max\_s = -1e10;

double \*s = vector(n\_new);

for (i=0;i<n\_new;i++){

s[i] = 0;

for (j=0;j<(m\_new-2);j++)

s[i] += A\_new[i][j];

if (b\_new[i]>=max\_s)

max\_s = b\_new[i];

}

for (i=0;i<n\_new;i++){

for (j=0;j<(m\_new-1);j++){

A\_new[i][j] = U\*A\_new[i][j]-b\_new[i];

}

}

for (j=0;j<m\_new;j++)

c\_new[j] = U\*c\_new[j];

//Print\_Matr(A\_new,n\_new,m\_new);

//Print\_Vect(c\_new,m\_new);

double \*sum\_i = vector(n\_new);

for (i=0;i<n\_new;i++){

sum\_i[i] = 0;

for (j=0;j<(m\_new-1);j++)

sum\_i[i] += A\_new[i][j];

A\_new[i][m\_new-1] = - sum\_i[i];

}

double sum\_j = 0;

for (j=0;j<(m\_new-1);j++)

sum\_j += c\_new[j];

c\_new[m\_new-1] = - sum\_j+1;

//Print\_Matr(A\_new,n\_new,m\_new);

//Print\_Vect(c\_new,m\_new);

double \*y\_opt = vector(m\_new);

y\_opt = Karmarkar\_metod(c\_new,A\_new,n\_new,m\_new,maxiter,eps,k);

//Print\_Vect(y\_opt,m\_new);

double \*x\_opt = vector(max(mi,me));

double z\_opt = 0;

for (i=0;i<max(mi,me);i++){

x\_opt[i] = U\*y\_opt[i];

z\_opt += c[i]\*x\_opt[i];

}

if (sign == 1)

z\_opt = - z\_opt;

printf("\n\nНайденное оптимальное решение: x =");

Print\_Vect(x\_opt,max(mi,me));

printf("\nОртимальное значение целевой функции: z = %f",z\_opt);

}

}

int main() {

SetConsoleOutputCP(CP\_UTF8);

/\*

Метод Кармаркара решает ЗЛП в виде:

z = c^t\*x ->min

|Ax=0

|sum(xi)=1

|xi>=0

Если ЗЛП задана не в специальной форме, то функция Karmarkar приводит заданную

систему ограничений к специальной форме и находит решение задачи оптимизации

\*/

int maxiter = 1000;

double eps = 1e-6;

int k = 0;

double U = 4; //Положительный коэффициент, подбираемы для каждой задачи отдельно

/\* Пути к файлам со значениями для задачи \*/

char c\_path[] = "c.txt";

char A\_path[] = "A.txt";

char b\_path[] = "b.txt";

char Aeq\_path[] = "Aeq.txt";

char beq\_path[] = "beq.txt";

//char lb\_path[] = "lb.txt";

//char ub\_path[] = "ub.txt";

Karmarkar(c\_path,A\_path,b\_path,Aeq\_path,beq\_path,maxiter,eps,&k,U);

getchar();

return 0;

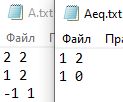
}

**Тестирование разработанной программы**

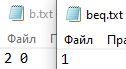
Пример 1. Пусть задана ЗЛП в следующем виде:

Запишем в файлы A.txt, b.txt, Aeq.txt, beq.txt, c.txt коэффициенты ограничений и целевой функции соответственной.

В файлах A.txt и Aeq.txt сначала идут размерности соответствующих матриц ограничений ( A.txt – неравенства, Aeq.txt – равенства ).



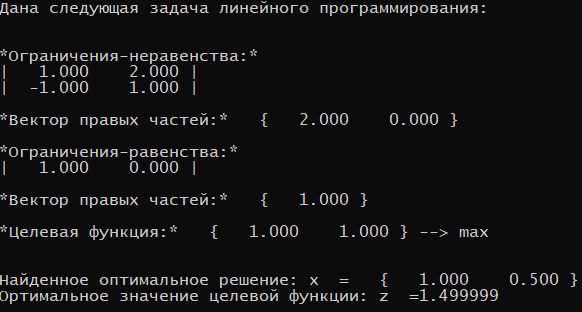
В файлах b.txt и beq.txt записаны правые части неравенств и уравнений.

****

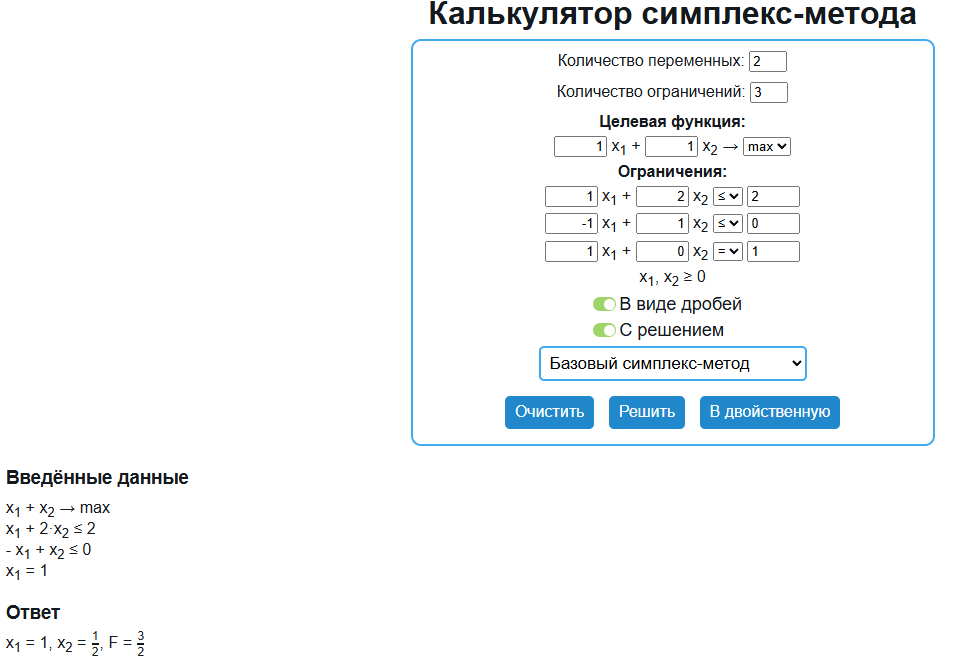
Вектор целевой функции представлен в файле c.txt следующим образом: сначала идёт знак + или - (максимум или минимум), затем коэффициенты целевой функции.



В результате работы программа выведет в консоль исходную задачу, оптимальное решение и оптимальное значение функции:

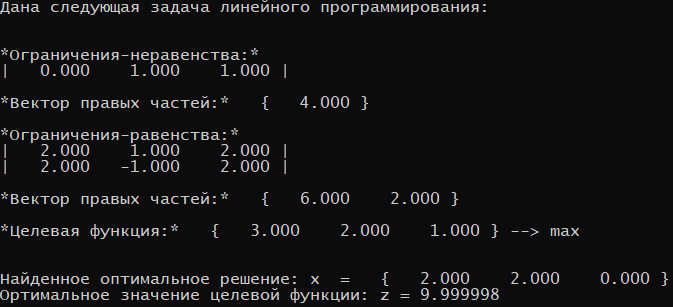


В качестве проверки воспользуемся сайтом <https://programforyou.ru/calculators/simplex-method>, чтобы вычислить эту же задачу Симплекс-методом. Оптимальная точка и оптимальное решение в разработанной программе и на сайте совпадают.

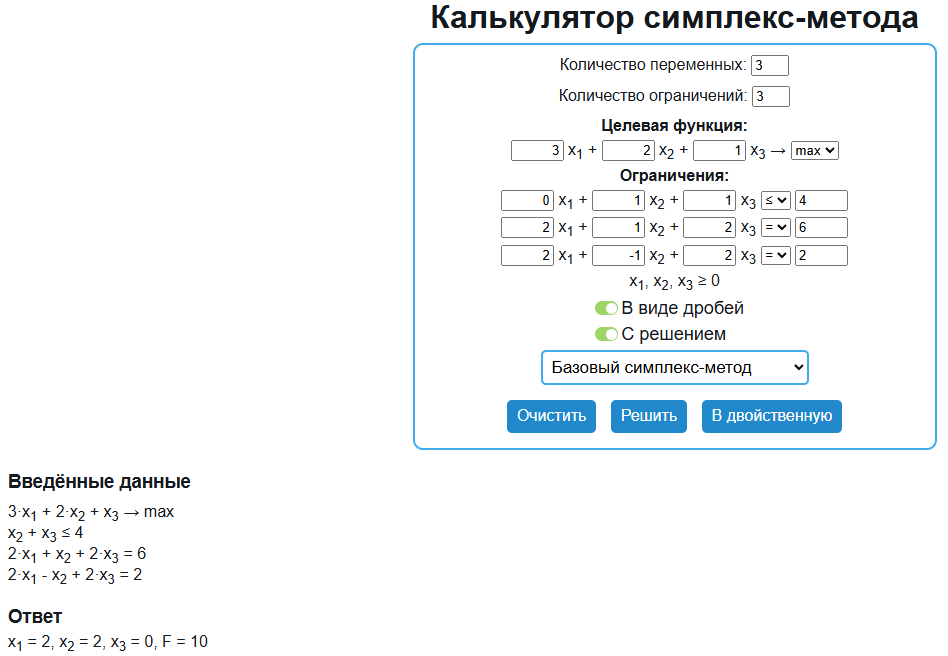


Пример 2. Пусть задана следующая ЗЛП:

После занесения коэффициентов в текстовые файлы, программа выводит отпимальное значени X и оптимальное значение целевой функции Z:

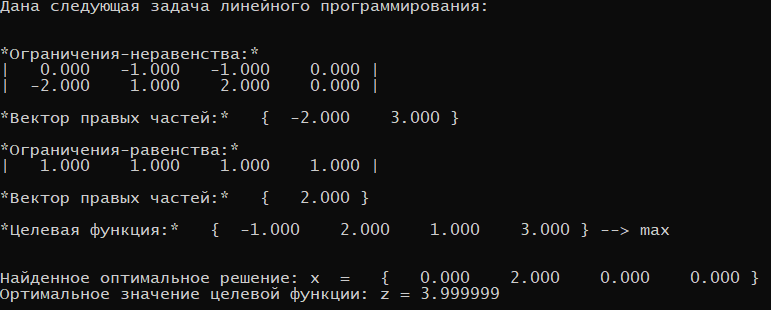


Проверка Симплекс-методом:

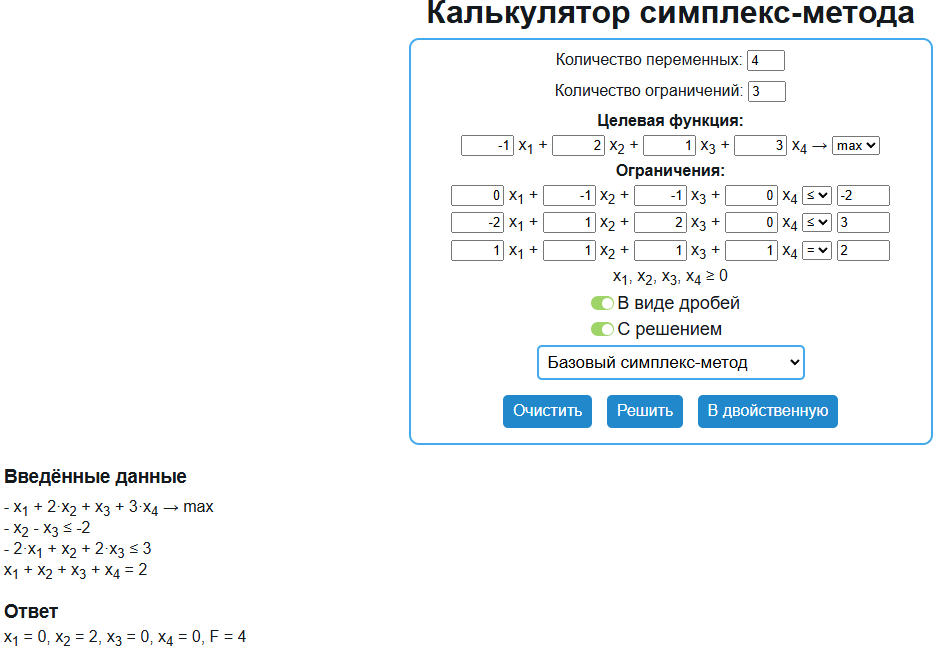


Пример 2. Пусть задана следующая ЗЛП:

Коэффициент U для данной системы ограничений равен 4. Далее представлен вывод оптимального решения и оптимального значения целевой функции:



Проверка Симплекс-методом:



**Итоги:** реализованный метод Кармаркара для решения ЗЛП успешно работает, если в ограничениях заданы неравенства и равенства. В данный момент подготовлена основа приведения ЗЛП к специальной форме, если заданы только ограничения-неравенства или только ограничения-равенства.