# Задание

Написать программу, на языке С++ в среде VisualStudio “**Расчёт рациона с минимальной стоимостью”:**

Заданы n пищевых продуктов, содержащих m различных питательных веществ. Обозначим через aij содержание (долю) j-го питательного вещества в i-ом продукте, через bj — суточную потребность организма в j-ом питательном веществе, через ci — стоимость единицы i-го продукта. Составить суточный рацион питания минимальной стоимости, удовлетворяющий потребность во всех питательных веществах.

# Расчёт рациона минимальной стоимости

Задача расчёта рациона минимальной стоимости является одной из классических задач линейного программирования – нахождение точки минимума линейной функции f(x)=cx при условии ограничений ax>=b, где b>=0 и x>=0.

Для решения данной задачи был выбран симплекс-метод с введением дополнительных и искусственных переменных. Использовалась расширенная таблица плана симплекс-метода со строками F и M, и задача решалась в два этапа – сперва исключались из базиса искусственные переменные вычислениями по строке M, а затем применялся алгоритм нахождения оптимального плана по строке F.

Программа работает в 2-х режимах:

* с выводом результатов работы на экран (для демонстрации работоспособности);
* без вывода результатов работы на экран, но с определением времени, затрачиваемого на вычисления;

Листинг кода

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Вычисление точки минимума линейной функции f(x)=cx

Для неравенства вида ax>=b и ограничения b>=0,x>=0

\*\*a - матрица коэффициентов левой части неравенства

\*b - вектор значений правой части неравенства

\*c - коэффициенты линейной функции

m - количество неравенств (строк)

n - количество переменных (столбцов)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double \*ration(double \*\*a, double \*b, double \*c, int m, int n, int demo){

// Метод искусственного базиса

double \*\*plan; // Симплекс-таблица

int \*index; // Индексы переменных в базисе

int \*flags; // Флаги переменных в базисе

index = new int[m];

flags = new int[m+n];

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

index[i] = m+n+i; // индексы исскуственных переменных

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

flags[i] = 0; // флаги переменных

plan = new double \*[m+2];

for(int i = 0; i < m+2; i++)

plan[i] = new double[1+m+n];

#pragma omp parallel for

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

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

plan[i][j+1]=(j<n)?(a[i][j]):((j==(n+i))?-1:0);

// Формируем F строку матрицы

#pragma omp parallel for

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

plan[m][j+1] = (j<n)?(-c[j]):0;

}

// Формируем M строку матрицы

#pragma omp parallel for

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

plan[m+1][j+1] = (j<n)?0:-1;

if(j<n) for(int i = 0; i < m; i++) plan[m+1][j+1] += a[i][j];

}

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

plan[i][0]=b[i];

plan[m][0]=0;

plan[m+1][0]=0;

if(simplex(plan, index, flags, m, n, 1, demo)!=0) return NULL; // Исключение искусственных переменных

if(simplex(plan, index, flags, m, n, 0, demo)!=0) return NULL; // Нахождение решения

double \* result;

result = new double[n]; //результат

for(int i = 0; i < n; i++) result[i]=0;

for(int i = 0; i < m; i++) if(index[i]<n) result[index[i]]=plan[i][0];

for(int i = 0; i < m+2; i++) delete plan[i];

delete plan;

delete index;

delete flags;

return result;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Вычисление симлекс-метода по таблице плана

\*\*plan - таблица плана

\*index - индексы переменных в базисе

\*flags - флаги переменных в базисе

m - количество базисных переменных

n - количество переменных

extended = 0,1 - флаг использования расширенной таблицы

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int simplex(double \*\*plan, int \*index, int \*flags, int m, int n, int extended, int demo) {

while(1){

if(demo!=0) {

cout << "\tТекущий план plan["<<1+m+extended<<","<<1+m+n<<"]:" << endl;

for(int i = 0; i < 1+m+extended; i++) {

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

cout << plan[i][j] << "\t";

cout << endl;

}

cout << "\tПеременные в базисе index["<<m<<"]:" << endl;

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

cout << index[i] << "\t";

cout << endl;

}

// Проверяем текущий план на оптимальность

// Если все оценки в последней строке плана неположительные, то план оптимален

double maxByRow=0;

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

if(flags[j]==0 && maxByRow < plan[m+extended][j+1])

maxByRow = plan[m+extended][j+1];

if(maxByRow<=0) return 0; // Найден оптимальный план

// Находим решающий элемент матрицы плана

double maxByMod = 0;

int r=0;

int c=0;

for(int j = 0; j < m+n; j++) if(flags[j]==0 && plan[m+extended][j+1]>0) {

double minByCol=-1;

for(int i = 0; i < m; i++) if(plan[i][j+1]>0) {

minByCol = plan[i][0]/plan[i][j+1];

for(int i1 = i+1; i < m; i++)

if(plan[i1][j+1]>0 && minByCol > plan[i1][0]/plan[i1][j+1])

minByCol = plan[i1][0]/plan[i1][j+1];

break;

}

if(minByCol < 0) return -1; // Решение не найдено

if(maxByMod<minByCol\*plan[m+extended][j+1]) {

maxByMod=minByCol\*plan[m+extended][j+1];

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

if(plan[i][j+1]>0 && minByCol==plan[i][0]/plan[i][j+1]) {

r = i;

c = j+1;

}

}

}

if(demo!=0) cout << "\tРешающий элемент: (" << r << "," << c << ")" << endl;

// Выполняем алгоритм Гаусса-Жордана для заданного решающего элемента матрицы

gauss\_jordan(plan, r, c, 1+m+extended, 1+m+n);

// Переменную c-1 отмечаем как включённыю в базис вместо предыдущей

if(index[r]<m+n) flags[index[r]]=0;

index[r] = c-1;

flags[index[r]]=1;

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Алгоритм Гаусса-Жордана для заданного решающего элемента матрицы

\*\*data - матрица

r - номер строки решающего элемента

с - номер столбца решающего элемента

m - количество строк матрицы

n - количество столбцов матрицы

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void gauss\_jordan(double \*\*data, int r, int c, int m, int n) {

double d = data[r][c];

double epsilon=1e-7; // точность вычислений с плавающей точкой

#pragma omp parallel for

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

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

if(i!=r && j!=c) {

data[i][j] -= data[r][j]\*data[i][c]/d;

if(abs(data[i][j])<epsilon) data[i][j]=0;

}

#pragma omp parallel for

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

if(j!=c)

data[r][j] /= d;

#pragma omp parallel for

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

if(i!=r)

data[i][c] = 0;

data[r][c] = 1;

# }

# Контрольные примеры работы программы

Пример 1.

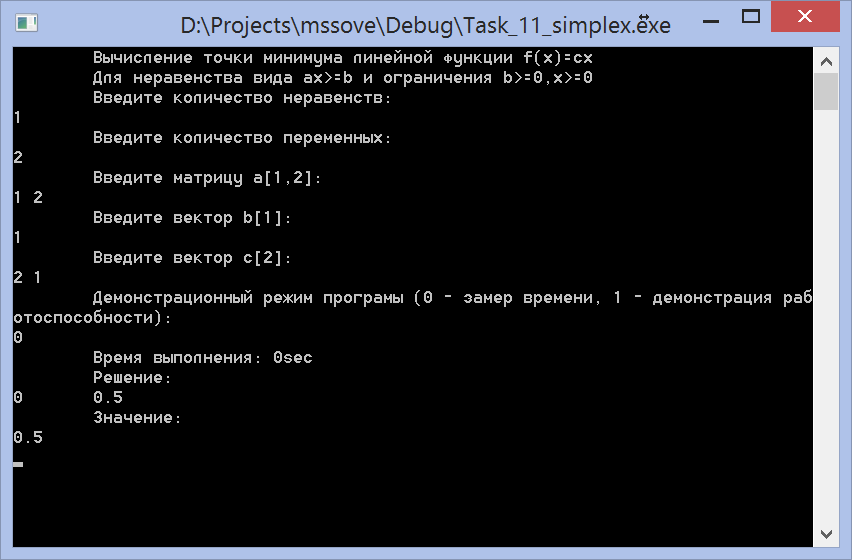


Рис.1..

Пример 2.

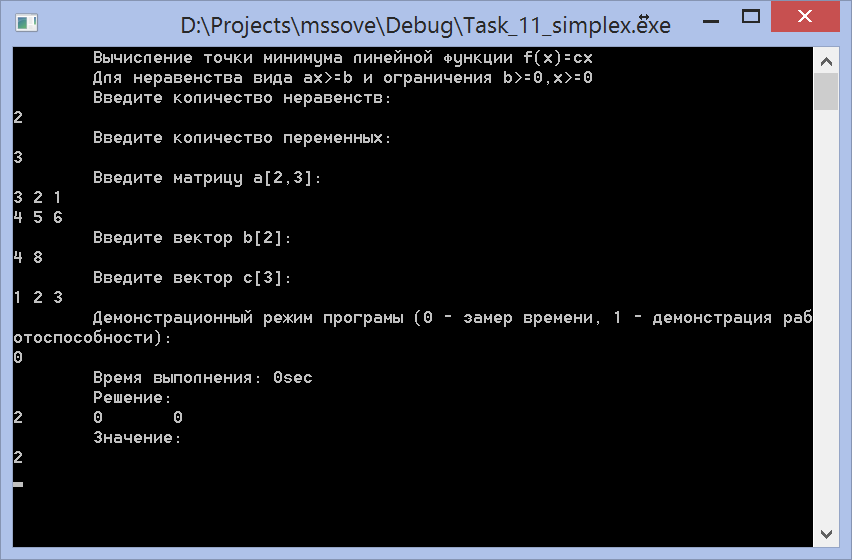


Рис.2..

# Заключение

В ходе данной лабораторной был изучен алгоритм симплекс-метода с введением дополнительных и искусственных переменных. Была сделана программная реализация данного алгоритма. Для данной программной реализации были проведены ряд тестов, показывающие правильность работы алгоритма.