Содержание

Введение

1. Описание задачи

2. Описание метода решения

3. Проектирование интерфейса

4. Структура программного модуля

5. Тестирование

Заключение

Список использованной литературы и программных средств

Приложение 1. Интерфейс приложения

Приложение 2. Листинг класса SimplexSolve

Введение

Линейное программирование – математическая дисциплина, посвященная теории и методам решения экстремальных задач на множествах n-мерного [векторного пространства](http://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE), задаваемых системами линейных уравнений и неравенств.

Линейное программирование является частным случаем выпуклого программирования, которое в свою очередь является частным случаем математического программирования. Термин «программирование» нужно понимать в смысле «планирования». Он был предложен в середине 1940-х годов [Джорджем Данцигом](http://ru.wikipedia.org/wiki/%D0%94%D0%B6%D0%BE%D1%80%D0%B4%D0%B6_%D0%94%D0%B0%D0%BD%D1%86%D0%B8%D0%B3), одним из основателей линейного программирования, ещё до того, как компьютеры были использованы для решения линейных задач оптимизации.

Работа посвящена наиболее распространенному методу решения задачи линейного программирования – симплекс-методу. Симплекс-метод является классическим и наиболее проработанным методом в линейном программировании.

1. Описание задачи

Задача линейного программирования (ЛП) возникает из необходимости оптимально использовать имеющиеся ресурсы. Это задачи, связанные с целеобразованием и анализом целей и функций; задачи разработки или совершенствования структур (производственных структур предприятий, организованных структур объединений); задачи проектирования (проектирование сложных робототехнических комплексов, гибких производственных систем).

В качестве конкретных примеров задач, которые относятся к области линейного программирования, можно назвать задачу об использовании сырья, задачу об использовании мощностей, задачу на составление оптимальной производственной программы.

Задача ЛП заключается в отыскании вектора , максимизирующего/минимизирующего линейную целевую функцию

(1)

при следующих линейных ограничениях

(2)

(3)

Запись задачи ЛП в виде (1)-(3) называется нормальной формой задачи.

Эту же задачу ЛП можно представить в векторно-матричной записи:

(4)

где  - вектор коэффициентов целевой функции,

 - вектор решения,

 - вектор свободных членов,

 - матрица коэффициентов.

Область  называется областью допустимых значений (ОДЗ) задач линейного программирования. Соотношения (2), (3) называются системами ограничений задачи ЛП. Так как , то можно ограничиться изучением задачи одного типа.

Решением задачи ЛП, или оптимальным планом, называется вектор, удовлетворяющий системе ограничений задачи и оптимизирующий целевую функцию.

Другая форма представления задачи ЛП – каноническая. Она имеет вид:



В канонической форме записи задач линейного программирования все переменные, входящие в систему ограничений, должны быть неотрицательными, а все ограничения должны быть представлены равенствами. Любую задачу линейного программирования можно свести к задаче линейного программирования в канонической форме. Для этого в общем случае нужно уметь сводить задачу максимизации к задаче минимизации; переходить от ограничений неравенств к ограничениям равенств и заменять переменные, которые не подчиняются условию неотрицательности.

2. Описание метода решения

Симплекс-метод является наиболее распространенным вычислительным методом, который может быть применен для решения любых задач ЛП как вручную, так и с помощью ЭВМ.

Этот метод позволяет переходить от одного допустимого решения к другому, причем так, что значения целевой функции непрерывно возрастают. В результате оптимальное решение находят за конечное число шагов. Алгоритм симплекс-метода позволяет также установить является ли задача ЛП разрешимой.

Рассмотрим задачу ЛП в канонической форме. Будем искать решение задачи (6), (7), (8).

(6)

(7)

(8)

1. Положим k = 1. Взяв переменные  за свободные и положив их равными нулю, а , переобозначив в  , - за базисные, находим первую крайнюю точку:

.

1. Заполним начальную допустимую симплекс-таблицу

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | … |  |  | … |  |  |
|  |  | … |  | 0 | … | 0 | 0 |
|  |  | … |  | 1 | … | 0 |  |
| … | … | … | … | … | … | … | … |
|  |  | … |  | 0 | … | 1 |  |

где  - вектор коэффициентов целевой функции,

 - вектор свободных членов,

 - матрица коэффициентов.

1. Если для k-той крайней точки все , то эта точка оптимальная, переход на пункт 7.

В остальных случаях переход к пункту 3.

1. Находим ведущий столбец . Правило выбора: выбираем столбец, в котором самый минимальный коэффициент  среди отрицательных:



1. Находим ведущую строку  по правилу:



Если все элементы ведущего столбца , то задача ЛП не является разрешимой, т.к. целевая функция не ограничена на множестве допустимых значений, переход на пункт 7.

Таким образом, ведущий элемент .

1. Выполняем один шаг метода Гаусса: выводим переменную с индексом  из числа базисных, а переменную с индексом  вводим в базис. Новые элементы ведущей строки находятся по формуле:



Новые значения элементов остальных строк матрицы:

,

Все элементы в ведущем столбце равны 0, тогда как сам ведущий элемент равен 1.

1. Получаем (k + 1) крайнюю точку . Полагая k = k + 1, перестраиваем симплекс-таблицу и переходим к пункту 2.
2. Конец решения.

3. Проектирование интерфейса

Разработанное приложение имеет простой однооконный интерфейс с набором всех необходимых инструментов для работы с программой.

Вверху окна стандартно располагается строка меню (JMenu), содержащая подменю (JSubMenu) Файл, Режим работы, Справка. В подменю Файл доступны следующие пункты меню (JMenuItem): Открыть файл, Выход. В подменю Режим работы с помощью Группы радиокнопок (JRadioButton Group) осуществляется взаимоисключающий выбор одного из двух режимов работы: автоматический, режим обучения. Из подменю Справка доступен вызов окна «О программе» (SimplexAboutBox).

Под строкой меню располагается панель инструментов, дублирующая функции, доступные из строки меню, но предоставляющая более удобное использование и быстрый доступ к ним пользователю. Она содержит кнопку (JButton) «Загрузить файл», а также список (JComboBox) для выбора режима работы.

Далее располагаются панели (JPanel), предоставляющие информацию о решаемой задаче, а именно:

* Панель параметров. Отображает количество свободных и базисных переменных и количество ограничений с помощью JLabel.
* Панель «Целевая функция» отображает вид целевой функции с помощью группы надписей (JLabel) и текстовых полей (JTextField).
* Панель «Решение» отображает вектор решения на текущем шаге выполнения задачи (для автоматического режима – оптимальное решение).

В центральной части окна расположена таблица (JTable), отображающая симплекс-таблицу на текущем шаге, и набор кнопок (JButton) для работы с таблицей:

В автоматическом режиме:

* Кнопка «решить». Осуществляет решение загруженной задачи или выдает сообщение о существующей ошибке (неверный входной файл, функция не ограничена на множестве допустимых решений).

В режиме обучения:

* «выбрать ведущий столбец». Вычисляет ведущий столбец и сравнивает результат с выбором пользователя. При несовпадении результатов выдает сообщение об ошибке «Ведущий столбец выбран неверно». Также вычисляет и заполняет вспомогательный столбец «Отношение», необходимый для выбора ведущей строки.
* «выбрать ведущую строку». Находит ведущую строку и сравнивает результат с выбором пользователя. При несовпадении результатов выдает сообщение об ошибке «Ведущая строка выбрана неверно».
* «перестроить симплексную таблицу». Осуществляет один шаг метода Гаусса для замены базисной переменной.

Внизу окна расположено поле для вывода многострочного текста (JTextArea), в котором отображается вспомогательная информация о текущем состоянии выполнения программы, а также о правилах выбора ведущих столбца и строки.

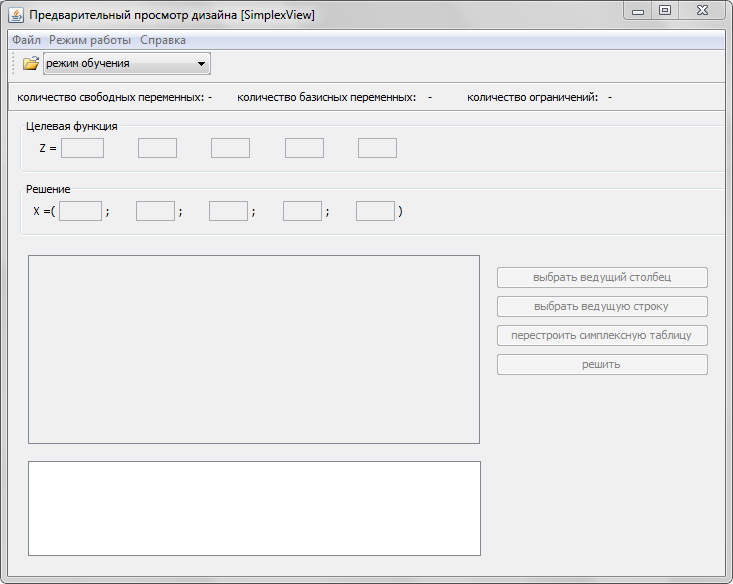


Рис. 1. Главное окно разработанного приложения.

Скриншоты интерфейса разработанного приложения при разных вариантах работы программы представлены в Приложении 1.

4. Структура программного модуля

В рамках поставленной задачи был разработан программный модуль, осуществляющий решение задачи линейного программирования на основе начального допустимого базисного решения.

Входными данными является текстовый файл, содержащий начальное допустимое базисное решение (на входные данные накладываются следующие ограничения: максимальное количество свободных переменных – 5, базисных – 8).

Выходными данными является полученный вектор решения, а также сообщения о состоянии выполнения программы.

Разработанный программный модуль предоставляет пользователю возможность выбора одного из двух режимов работы:

* автоматического,
* режима обучения.

В автоматическом режиме (Приложение 1, рис.2) решение задачи осуществляется в одно действие без участия пользователя. В режиме обучения (Приложение 1, рис.3) решение выполняется пошагово с привлечением участия пользователя на каждом шаге. Пользователь осуществляет выбор ведущего столбца, затем ведущей строки, после чего симплекс таблица перестраивается и итерация повторяется. Построение симплекс таблиц ведётся до тех пор, пока решение не станет оптимальным либо пока не будет получено сообщение об ошибке.

В программе предусмотрена обработка следующих исключительных ситуаций:

* загружен некорректный входной файл (Приложение 1, рис.4);
* целевая функция не ограничена на множестве допустимых решений (Приложение 1, рис.5).

Также реализована система подсказок, предоставляющая пользователю более лёгкую работу с программным средством.

В качестве среды разработки была выбрана NetBeans IDE, являющаяся средой разработки приложений на языке Java.

Структура созданного программного модуля представляет собой совокупность следующих классов:

* SimplexApp – главный класс приложения, осуществляющий запуск приложения, создание и отображение главного окна приложения.
* SimplexView – класс главного окна приложения, инициализирует все компоненты интерфейса, осуществляет их настройку, а также обработку событий, вызванных пользователем.
* SimplexAboutBox – класс, инициализирующий окно «О программе».
* SimplexSolve – класс, реализующий выполнение симплекс-метода.
* ReadFile – класс для обработки входного файла.
* TableView – реализует вывод симплекс-таблицы.
* Help – реализует вывод подсказок о ходе выполнения решения и о работе программы.

Класс SimplexSolve содержит следующие методы:

static void initSolution(int varCount) - создаёт вектор решения необходимой размерности.

static float[][] Solve(float[][] matrix) - осуществляет решение задачи в автоматическом режиме. Получая на вход начальную симплекс-таблицу, возвращает преобразованную симплекс-таблицу с полученным решением.

static boolean userChooseCol(float[][] matrix, JTable tableName) – выполняет выбор ведущего столбца и сравнивает полученный результат с выбором пользователя. Возвращает истину, если выбор был произведен верно, иначе ложь. В случае если результаты не совпали, выводит сообщение об ошибке.

static boolean userChooseRow(float[][] matrix, JTable tableName) – проводит проверку ограниченности целевой функции на множестве допустимых решений, выполняет выбор ведущей строки и сравнивает полученный результат с выбором пользователя. Возвращает истину, если выбор был осуществлён верно, иначе ложь. В случае если результаты не совпали, сообщает пользователю об ошибке.

static void userBuildNewTable(float[][] matrix, JTable tableName) – перестраивает симплексную таблицу и обновляет вектор решения.

static boolean checkSolved(float matrix[][]) – проверяет текущее решение на оптимальность. Возвращает истину, если решение оптимально, иначе ложь.

Класс ReadFile содержит следующие методы:

static float[][] read(String filename) throws IOException – осуществляет чтение входного файла. При отсутствии ошибок, возвращает начальную симплексную таблицу, иначе выдает сообщение об ошибке входных данных.

static String[] doVarCol(), static String[] doVarRow() – создают вспомогательные строку и столбец обозначения переменных для отображения симплекс-таблицы.

static int getVarCount() – возвращает количество свободных переменных.

static int getBvarCount() – возвращает количество базисных переменных.

static int getCondCount() – возвращает количество ограничений.

Класс TableView содержит следующие методы:

static void clearTable(JTable tableName) – очищает таблицу.

static void setNames (JTable tableName) – заполняет шапку таблицы.

static void fillTable(JTable tableName, float[][] matrix) – заполняет симплекс таблицу.

static void fillProportion(JTable tableName, float[] proportion, int tempCInd) – заполняет вспомогательный столбец отношения (в обучающем режиме).

Класс Help содержит метод

static String showHelp(int event) – осуществляет вывод подсказки в зависимости от произошедшего события.

Листинг класса SimplexSolve, содержащего алгоритм симплекс-метода, приведёт в Приложении 2.

5. Тестирование

Тест № 1.

Вход: Загружаем корректный файл с начальным допустимым базисным решением data1.txt. Выбираем автоматический режим работы. Нажимаем кнопку «решить».

Выход: В панели «Решение» отображается вектор решения. В таблицу выводится симплекс-таблица на последней итерации выполнения алгоритма. В поле подсказки отображается информация о том, что задача решена и полученное решение оптимально.

(Приложение 1, рис. 2)

Тест № 2.

Вход: Загружаем некорректный файл с начальным допустимым базисным решением data2.txt.

Выход: Выводится сообщение «Ошибка входных данных». В поле подсказки отображается дополнительная информация об ограничениях на входные данные.

(Приложение 1, рис.4)

Тест № 3.

Вход1: Загружаем корректный файл с начальным допустимым базисным решением data5.txt, целевая функция не ограничена на множестве допустимых решений.. Выбираем режим обучения. Выделяем неверный ведущий столбец. Нажимаем кнопку «выбрать ведущий столбец».

Выход1: Выводится сообщение «ведущий столбец выбран неверно». В поле подсказки отображается подсказка с правилом выбора ведущего столбца.

Вход2: Выделяем верный ведущий столбец. Нажимаем кнопку «выбрать ведущий столбец».

Выход2: В таблице выводится дополнительный столбец отношения.

Вход3: Выделяем ведущую строку. Нажимаем кнопку «выбрать ведущую строку».

Выход3: Выводится сообщение «функция не ограничена на множестве допустимых значений». Решение задачи прекращается. В поле подсказки выводится информация о том, что дальнейшее решение задачи невозможно, для решения новой задачи необходимо загрузить файл.

(Приложение 1, рис. 5)

Заключение

По итогам данной работы был разработан программный модуль, реализующий решение задач линейного программирования симплекс-методом на основе начального допустимого базисного решения.

Симплекс-метод является наиболее известным и широко применяемым на практике для решения общей задачи линейного программирования. Алгоритм метода прост в понимании и легок в реализации, при программировании алгоритма никаких сложностей принципиального характера не возникло. Однако, несмотря на то, что симплекс-метод является достаточно эффективным алгоритмом, показавшим хорошие результаты при решении прикладных задач ЛП, он является алгоритмом с экспоненциальной сложностью. Причина этого состоит в комбинаторном характере симплекс-метода, последовательно перебирающего вершины многогранника допустимых решений при поиске оптимального решения. Таким образом, данный метод эффективен на небольшом наборе входных данных, при увеличении же их сложность алгоритма будет возрастать скачкообразно.

В рамках поставленной задачи были выполнены все требования, реализованы все функциональные составляющие. Интерфейс разработанного программного продукта интуитивно понятен и легок в использовании. Модульность приложения предоставляет возможности для его дальнейшей доработки расширения и встраивания в вычислительные комплексы.

Список использованной литературы и программных средств

1. Исенбаева Е.Н. МЕТОДИЧЕСКИЕ УКАЗАНИЯ к проведению практических занятий по курсу "Системный анализ" на тему "Симплекс-метод решения задачи линейного программирования"/ Е.Н. Исенбаева. – Ижевск: ИжГТУ, 1999. – 14с.
2. Электронный ресурс: Симплекс метод: программная реализация симплекс-метода на языке Java - <http://www.mathelp.spb.ru/lp.htm>
3. Электронный ресурс: Wikipedia – Линейное программирование – http://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%BE%D0%B5\_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5
4. Электронный ресурс: Wikipedia – Задача линейного программирования <http://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5>
5. Среда разработки NetBeans IDE 6.9.1

Приложение 1

Интерфейс программы

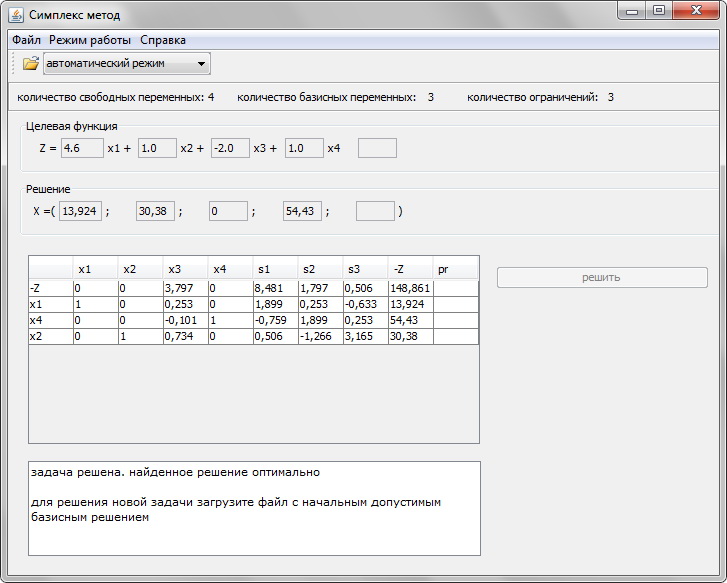


Рис. 2. Автоматический режим работы программы

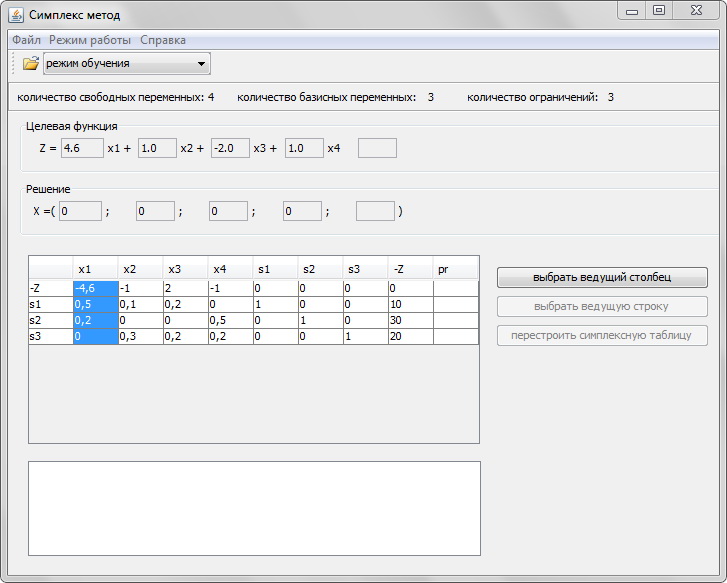


Рис. 3. Режим обучения

линейный программирование симплекс метод

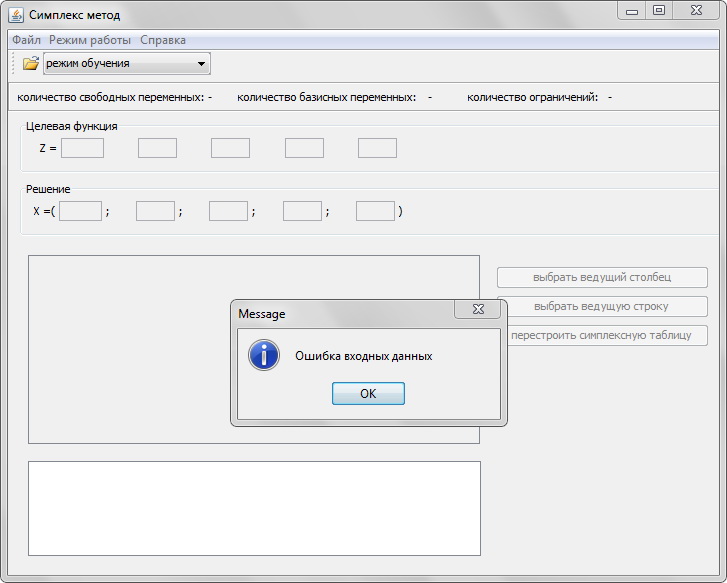


Рис. 4. Обработка события «Ошибка входных данных»

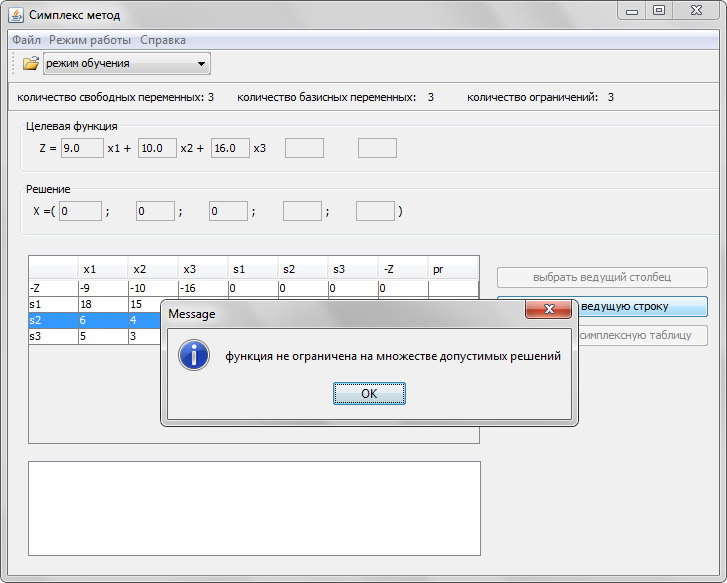


Рис. 5. Обработка события «Целевая функция не ограничена на множестве допустимых решений».

Приложение 2

Листинг класса SimplexSolve

package simplex;

import javax.swing.JOptionPane;

import javax.swing.JTable;

public class SimplexSolve {

static boolean solved = false;

static boolean lim = false;

static int tempCInd = 0;

static int minRInd = 0;

static int minCInd = 1;

static float[] solution;

//решение задачи в автоматическом режиме

static float[][] Solve(float[][] matrix){

M1: {

solved = true;

// проверяем решение на оптимальность

for (int i = 0; i <= ReadFile.colCount; i++){

if (matrix[i][0] < 0)

solved = false;

}

/ /пока решение не оптимально

while (!solved){

// находим ведущий столбец

float minR = matrix[0][0];

int minRInd = 0;

for (int i = 0; i <= ReadFile.colCount; i++){

if (matrix[i][0] < minR){

minR = matrix[i][0];

minRInd = i;

}

}

//проверяем, ограничена ли целевая функция на множестве доп. решений

lim = false;

for (int i = 0; i <= ReadFile.rowCount; i++){

if (matrix[minRInd][i] > 0)

lim = true;

}

//если функция не ограничена, выводим сообщение об ошибке, прерываем

//решение

if (!lim){

solved = true;

JOptionPane.showMessageDialog(null, "функция не ограничена на множестве допустимых решений");

break M1;

}

//находим ведущую строку

float minC = matrix[ReadFile.colCount][1]/matrix[minRInd][1];

int minCInd = 1;

for (int i = 1; i < tempCInd; i++){

if (matrix[ReadFile.colCount][i]/matrix[minRInd][i] < minC){

minC = matrix[ReadFile.colCount][i]/matrix[minRInd][i];

minCInd = i;

}

}

for (int i = tempCInd + 1; i <= ReadFile.rowCount; i++){

if (matrix[ReadFile.colCount][i]/matrix[minRInd][i] < minC){

minC = matrix[ReadFile.colCount][i]/matrix[minRInd][i];

minCInd = i;

}

}

//выводим из базиса базисную переменную [0][minCInd], вводим в базис

//переменную [minRInd][0]

ReadFile.varCol[minCInd-1] = ReadFile.varRow[minRInd] ;

//строим новую симплексную таблицу

//делим ведущую строку на ведущий элемент [minRInd][minCInd]

float temp = matrix[minRInd][minCInd];

System.out.print(">> " + temp + "\n");

System.out.print("\nведущая строка: ");

for (int i = 0; i <= ReadFile.colCount; i++){

matrix[i][minCInd] /= temp;

}

//получаем нули в ведущем столбце

for (int j = 0; j < minCInd; j++){

float minTemp = matrix[minRInd][j];

for (int i = 0; i <= ReadFile.colCount; i++){

matrix[i][j] += matrix[i][minCInd] \* -minTemp;

}

}

for (int j = minCInd+1; j <=ReadFile.rowCount; j++){

float minTemp = matrix[minRInd][j];

for (int i = 0; i <= ReadFile.colCount; i++){

matrix[i][j] += matrix[i][minCInd] \* -minTemp;

}

}

//обновляем вектор решения

for (int i = 0; i < ReadFile.bvarCount; i++){

for (int j = 0 ; j < ReadFile.varCount; j++){

int k = j + 1;

String tempS = "x" + k;

if (tempS.equals(ReadFile.varCol[i]))

solution[j] = matrix[ReadFile.colCount][i+1];

}

}

tempCInd = minCInd;

//рекурсивно вызываем процедуру, пока решение не будет оптимальным

Solve(matrix);

}

}

return matrix;

}

//создаем вектор решения

static void initSolution(int varCount){

solution = new float[varCount];

for (int i = 0; i < varCount; i++){

solution[i] = 0;

}

}

//выбор ведущего столбца в режиме обучения

static boolean userChooseCol(float[][] matrix, JTable tableName){

boolean err = false;

M1: {

//находим ведущий столбец

float minR = matrix[0][0];

minRInd = 0;

for (int i = 0; i <= ReadFile.colCount; i++){

if (matrix[i][0] < minR){

minR = matrix[i][0];

minRInd = i;

}

}

//проверяем выбор пользователя

while (minRInd != SimplexView.getSelectedCol() - 1){

JOptionPane.showMessageDialog(null, "ведущий столбец выбран

неверно");

err = true;

break M1;

}

int temp = minRInd;

float[] proportion = new float[ReadFile.rowCount];

//вычисляем вспомогательный столбец отношения

for (int i = 1; i <= ReadFile.rowCount; i++){

if ( i == tempCInd ){

proportion[i-1] = java.lang.Float.NaN;

}

else{

proportion[i-1] = matrix[ReadFile.colCount][i] /

matrix[temp][i];

}

}

TableView.fillProportion(tableName, proportion, tempCInd);

}

return err;

}

//выбор ведущей строки в режиме обучения

static boolean userChooseRow(float[][] matrix, JTable tableName){

lim = false;

boolean err = false;

M1:{

//проверяем, ограничена ли целевая функция на множестве доп. решений

for (int i = 0; i <= ReadFile.rowCount; i++){

if (matrix[minRInd][i] > 0)

lim = true;

}

if (!lim){

JOptionPane.showMessageDialog(null, "функция не ограничена на

множестве допустимых решений");

break M1;

}

//находим ведущую строку

float minC = matrix[ReadFile.colCount][1]/matrix[minRInd][1];

minCInd = 1;

for (int i = 1; i < tempCInd; i++){

if (matrix[ReadFile.colCount][i]/matrix[minRInd][i] < minC){

minC = matrix[ReadFile.colCount][i]/matrix[minRInd][i];

minCInd = i;

}

}

for (int i = tempCInd + 1; i <= ReadFile.rowCount; i++){

if (matrix[ReadFile.colCount][i]/matrix[minRInd][i] < minC){

minC = matrix[ReadFile.colCount][i]/matrix[minRInd][i];

minCInd = i;

}

}

//проверяем выбор пользователя

System.out.print("user: " + SimplexView.getSelectedRow() + "; min: "

+minCInd);

while (minCInd != SimplexView.getSelectedRow()){

err = true;

JOptionPane.showMessageDialog(null, "ведущая строка выбрана

неверно");

break M1;

}

}

return err;

}

//перестраивает симплексную таблицу

static void userBuildNewTable(float[][] matrix, JTable tableName){

//выводим из базиса базисную переменную [0][minCInd], вводим в базис

//переменную [minRInd][0]

ReadFile.varCol[minCInd-1] = ReadFile.varRow[minRInd] ;

//строим новую симплексную таблицу

//делим ведущую строку на ведущий элемент [minRInd][minCInd]

float temp = matrix[minRInd][minCInd];

for (int i = 0; i <= ReadFile.colCount; i++){

matrix[i][minCInd] /= temp;

}

//получаем нули в ведущем столбце

for (int j = 0; j < minCInd; j++){

float minTemp = matrix[minRInd][j];

for (int i = 0; i <= ReadFile.colCount; i++){

matrix[i][j] += matrix[i][minCInd] \* -minTemp;

}

}

for (int j = minCInd+1; j <=ReadFile.rowCount; j++){

float minTemp = matrix[minRInd][j];

for (int i = 0; i <= ReadFile.colCount; i++){

matrix[i][j] += matrix[i][minCInd] \* -minTemp;

}

}

for (int i = 0; i < ReadFile.bvarCount; i++){

for (int j = 0 ; j < ReadFile.varCount; j++){

int k = j + 1;

String tempS = "x" + k;

if (tempS.equals(ReadFile.varCol[i]))

solution[j] = matrix[ReadFile.colCount][i+1];

}

}

tempCInd = minCInd;

}

//проверяет, оптимально ли текущее решение

static boolean checkSolved(float matrix[][]){

solved = true;

for (int i = 0; i <= ReadFile.colCount; i++){

if (matrix[i][0] < 0)

solved = false;

}

if (solved){

JOptionPane.showMessageDialog(null, " задача решена ");

tempCInd = 0;

}

return solved;

}

}