Санкт-Петербургский Политехнический Университет

Институт Компьютерных Наук и Технологий

**Кафедра «Информационных и управляющих систем»**

**Курсовой проект**

по дисциплине «Теория принятия решений»  
«Метод линейной суперизации в задачах линейного программирования»

Перевод статьи. Реализация алгоритма.

Выполнил

студент гр.13544/2 «\_\_\_\_\_\_\_\_\_\_» Е.О. Грицина

Руководитель «\_\_\_\_\_\_\_\_\_\_» Ю.Г. Ефремов

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_» \_\_\_\_\_\_\_\_\_\_2016 г.

Санкт-Петербург

2016

# Содержание

[Введение 4](#_Toc468954161)

[Описание проблемы 5](#_Toc468954162)

[Принцип суперизации. 5](#_Toc468954163)

[Линейная суперизация. 5](#_Toc468954164)

[Алгоритм линейной суперизации 7](#_Toc468954165)

[Программная реализация 9](#_Toc468954166)

[Технические требования 9](#_Toc468954167)

[Описание классов 9](#_Toc468954168)

[Описание пользовательского интерфейса 10](#_Toc468954169)

[Результаты тестирования 12](#_Toc468954170)

[Заключение 13](#_Toc468954171)

[Приложение 1. Исходный код программы 14](#_Toc468954172)

[Приложение 2. Перевод статьи 28](#_Toc468954173)

[Приложение 3. Оригинал статьи на английском языке 57](#_Toc468954183)

# Введение

Целью данного курсового проекта является изучение существующей проблемы в области методов оптимизации, описанной в статье на английском языке. В качестве основы для данной работы, была выбрана статья «Can Linear Superiorization Be Useful for Linear Optimization Problems?», написанная в 2016 году и размещенная от имени Yair Censor на интернет ресурсе <http://www.optimization-online.org/>.

Выбранная статья рассматривает возможность применения так называемого метода суперизации для решения задач линейной оптимизации. Она содержит довольно подробное описание сути метода, историю его появления и модификаций, теоретические выкладки и практические рекомендации по применению суперизации в рассматриваемых задачах.

Курсовой проект содержит краткое описание метода суперизации, описание алгоритма применения этого метода в задачах линейного программирования, реализацию этого алгоритма и пользовательского интерфейса. Также в приложении представлен текст разработанной программы, оригинал статьи и ее перевод на русский язык.

# Описание проблемы

В выбранной статье рассматривается метод линейной суперизации в качестве инструмента для решения задач линейного программирования. Метод линейной суперизации не предназначен для нахождения точки минимума функции, но он может быть применим для регулирования работы линейных алгоритмов, которые ищут точки с наименьшим значением целевой функции. Этот алгоритм особенно хорошо применим к задачам большой размерности. Он является хорошей альтернативой Simplex -методу линейного программирования, сравнение с которым приведено в рассматриваемой статье.

## Принцип суперизации.

К сожалению, в русском языке не существует перевода для слова «Superiorization», поэтому его значение понять довольно трудно.

Многие методы оптимизации не приспособлены для работы с ограничениями (неравенствами). Так, например в градиентных методах, условия ограничений обрабатываются на каждой итерации, что является дорогостоящей операцией. Постоянное решение этой нетривиальной задачи приводит к тому, что градиентные методы применимы на довольно ограниченных размерностях. Другие методы пытаются «встроить» ограничения в исходную функцию, и работать уже с новой целевой функцией.

В отличие от этих подходов, методы суперизации можно рассматривать как противоположную точку зрения. Вместо того чтобы адаптировать алгоритмы минимизации без ограничений для обработки ограничений, здесь адаптируются алгоритмы поиска поисках минимума целевой функции в ограниченных множествах. Это делается с сохранением исходного алгоритма, и не требует больших вычислений.

У метода суперизации существует два преимущества:

1. Вычислительное. Алгоритмы поиска в допустимом множестве менее требовательны к ресурсам. Таким образом, рассматриваемый метод позволяет эффективно модифицировать исходные алгоритмы с помощью недорогих дополнений, что хорошо работает на практике.
2. Прикладное. Во многих существующих реальных приложениях выбор целевой функции является зависимым от модели и метода сбора данных, что приводит к возникновению ограничений. И в таких случаях для получения «приемлемых результатов», часто бывает достаточно найти допустимую точку, которая уменьшает значение целевой функции (но не минимизирует ее), с чем прекрасно справляется метод суперизации.

## Линейная суперизация.

Рассмотрим пару (*М,* ***А***), где *М,* называемое целевым набором, является подмножеством заданного множества *Q* J-мерного евклидового пространства: *M ⊂ Q ⊆ RJ*. Пусть ***А*** *: Q → Q* будет неким алгоритмическим оператором, который определяет итерационный процесс, называемый базовым алгоритмом,

*x0 ∈ Q, xk+1 =* ***A****(xk), k = 1, 2, …*

задачей которого является найти точку в целевом множестве *М*. Мы, в дальнейшем, будем называть такую пару (*M,* ***A***) парой суперизации. Пусть *: Q ⊆ RJ→ R -* заданная действительная функция, называется базовой функцией.

Методология суперизации предназначена для ограниченных задач уменьшения функции следующего вида:

**Проблема.** Пусть *(М,* ***А****)* пара суперизации и пусть*: Q ⊆ RJ→ R* является базовой функцией. Необходимо найти точку *х\** в *М*, в которой значение функции меньше (не обязательно минимально), чем в точке, которая может быть достигнута путем применения базового алгоритма поиска в *М*.

Методология суперизации решает эту задачу путем исследования возмущений базового алгоритма, а затем с помощью предупреждения таких возмущений, "заставляет" базовый алгоритм, делать поправку шага в дополнение к своей первоначальной задаче уменьшения функции. Добавление таких возмущений в алгоритм называется "суперизация базового алгоритма".

Определим целевое множество *M* как*:*

где даны: I x J матрица действительных чисел и вектор .

Пусть целевой функцией для линейной суперизации будет:

где – скалярное произведение вектора *x* и данного вектора .

Следуя общим принципам методологии суперизации, далее в статье приводится алгоритм для решения задачи линейного программирования. На вход алгоритма подаются исходные данные проблемы A, b, и c. Так же выбранная пользователем начальная точка у ̅ и ядро 0 < a <1, с помощью которых алгоритм генерирует размеры шагов в соответствии с целым числом N (смотрите пункт 7 в п.3.3).

## Алгоритм линейной суперизации

1. **установим**

2. **установим**

3. **установим**

4. **пока** не выполняется правило остановки

5. **установим**

6. **установим**

7. **установим**

8. **пока** n<N

9. **установим**

10. **установим**

11. **установим**

12. **установим**

13. **установим**

14. **выход**

15. **установим**

16. **установим**

17. **установим**

18. **выход**

В качестве базовой функции ***A*** использован метод релаксацииЭгмона-Моцкина-Шенберга, который можно описать следующим образом:

Обозначим полупространства, представленные отдельными строкам в Hi,

,

где является i-й строкой матрицы A, а – i-я компонента вектора b. Ортогональная проекция произвольной точки на *Hi* имеет следующую форму:

**Инициализация:**  задается произвольно.

**Итерация:** Следующая итерация вектора ***x*** рассчитывается на основе текущей итерации *xk* следующим образом:

(6)

**Параметры релаксации:** Параметры λk выбраны такими, что , для всех , с некоторыми сколь угодно малыми .

**Контроль:** Контрольная последовательность зациклена на {1, 2…, I}.

Все подробности и детали реализации этих алгоритмов так же приведены в статье. После изучения этих особенностей мне удалось реализовать описанный метод в виде программного продукта.

# Программная реализация

Программная реализация описанного алгоритма представляет собой реализацию алгоритма поиска AMS точки в ограниченном множестве с добавлением супризации. Программа представляет собой приложение для операционной системы Android и может запускаться полностью локально на любом устройстве.

Структурно реализация состоит из следующих модулей:

1. Файлы описания разметки графического интерфейса.
2. Модуль графического интерфейса пользователя, который позволяет вводить свои условия для задачи линейной минимизации. А также предоставляет сравнительную таблицу и графики для алгоритмов LinSup и Simplex.
3. Контроллер вычислений - принимает на вход условия задачи, реализует алгоритм и вычисляет значения искомой точки.
4. Модуль вспомогательных математических функций. Реализует функции для работы с матрицами и векторами.

## Технические требования

Программная реализация выполнена в среде Intellij IDEA в виде Android Studio 2.1 на языке Java. К устройству предъявляются следующие требования:

1. ОС Android 4.4 и выше
2. 512 Mb оперативной памяти
3. Возможность устанавливать неподписанные приложения с карты памяти.

## Описание классов

**Класс MainActivity** – главный экран приложения. Осуществляет навигацию пользователя.

**Класс CustomProblemActivity** – класс, реализующий возможность ввести пользователем свою задачу линейной оптимизации. Метод solveClicked – создает экземпляр класса LinSupSolver, передает туда условия задачи и ждет результатов расчета. После чего отображает результаты в текстовом поле.

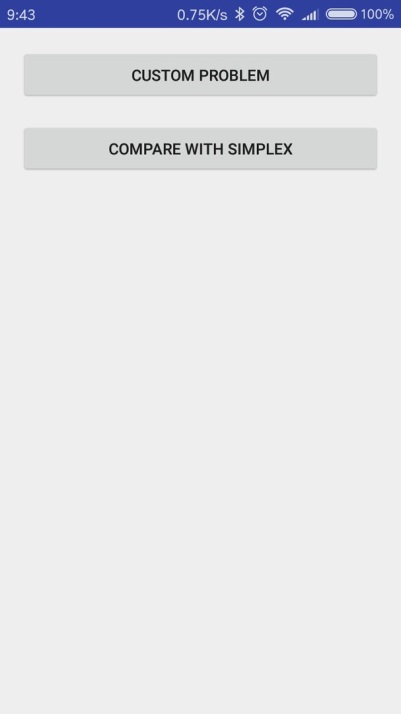
**Класс CompareActivity** - создает экземпляр класса LinSupSolver и обращается к специальному методу, который производит вычисления алгоритмов Simplex и LinSup для четырез различных разменостей. После чего отображает эти результаты в таблице на экране приложения. При нажатию на кнопку “Plot” по результатам в таблице строится график относительной ошибки метода LinSup для трех значений ядра.

**Класс LinSupUtils** – производит математические операции с векторами, такие как – скалярное произведение, сложение, вычитание, нахождение нормы вектора, получение случайного числа и так далее.

**Класс LinSupSolver** – содержит реализацию алгоритма AMS версии с суперизацией. Метод clear() – обнуляет все значение в алгоритме. Метод startLinSup() - реализует LinSup алгоритм, метода simplex() – получает решение задачи Simplex алгоритмом, методы init\_test(), solveForCompare(), initAndSolve() – используются для запуска разных задач оптимизации, методы getStoppingRule(), getProximityFun(), getPHi(), getNexbYbyA() – используются для вычислений в алгоритме LinSup.

## Описание пользовательского интерфейса

Начальный экран приложения, служит для навигации пользователя по разделам.

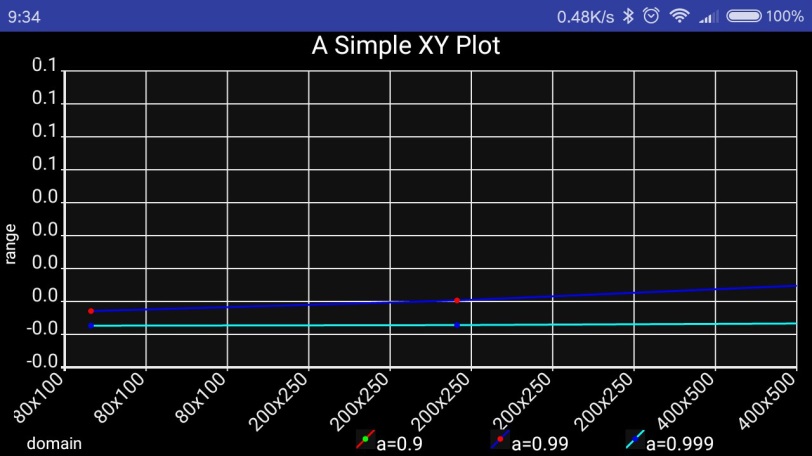


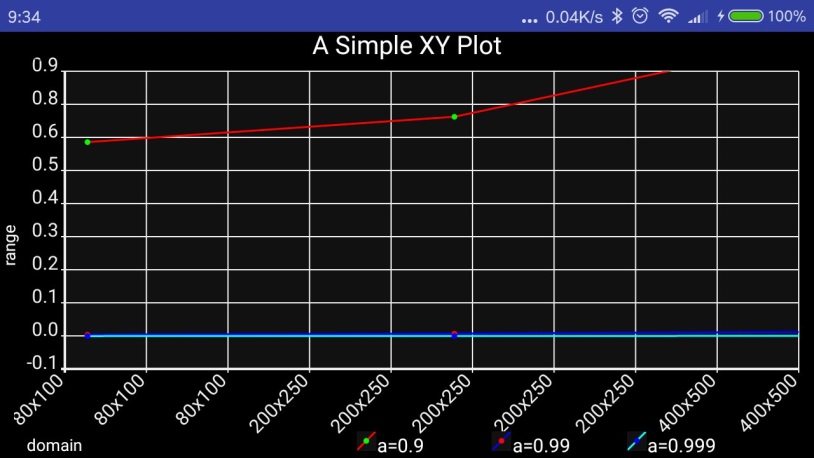
Экран ввода пользовательской задачи оптимизации. Позволяет задать размерность задачи IxJ, вектора B, C, матрицу A, вектора начальных решений Y0. При нажатии на кнопку Solve будет произведено решение задачи, а реалультат отобразится в поле Result.

Экран сравнения Simplex и LinSup алгоритма. Позволяет посмотреть результат автоматического решения сгенерированных проблем больших размерностей для алгоритма Simplex и для алгоритма LinSup с тремя значениями ядра "а". При нажатии на кнопку Plot, на экране будет показан график изменения относительной ошибки алгоритма LinSup, относительно алгоритма Simplex. Все вычисления происходят в аторматическом режиме, на этом экране от пользователя не требуется никаких действий.







# Результаты тестирования

В рассматриваемой статье реализация алгоритма линейной суперизации рассматривается с нескольких сторон. Мне наиболее интересным вариантом тестирования показалось сравнение работы алгоритма LinSup и Simplex.

Работа двух алгоритмов сравнилась на случайно сгенерированных проблемах разных размерностей. В случае с LinSup алгоритмом происходило три разных запуска с различными значениями ядра «а». Результаты относительно ошибки алгоритма LinSup и сравнение времени работы приведены в таблицах ниже.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Simplex** | **LinSup = 0.9** | **LinSup = 0.99** | **LinSup = 0.999** |
| **Размерность задачи** | Время  (с) | Время  (с) | Время  (с) | Время  (с) |
| 80x100 | 0.08 | 0.16 | 2.52 | 25.16 |
| 400x500 | 3.50 | 2.69 | 14.8 | 143.6 |
| 800x1000 | 26.68 | 12.2 | 92.3 | 815.78 |
| 2000x2500 | 800.3 | 40.8 | 564.9 | 5879.16 |
| 8000x10000 | 205698.68 | 706.3 | 8056.6 | **73086.45** |

Таблица 1. Сравнение скорости работы алгоритмов при разных размерностях задачи.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Simplex | LinSup = 0.9 | LinSup = 0.99 | LinSup = 0.999 |
| **Размерность задачи** | Значение функции | Значение функции | Значение функции | Значение функции |
| 80x100 | -145.16 | -35.6 | -130.5 | -145.36 |
| 400x500 | -894.5 | -98.35 | -901.26 | -893.3 |
| 800x1000 | -1568.5 | 464.6 | -1570.2 | -1568.01 |
| 2000x2500 | -3956.56 | 1658.6 | -3952.3 | -3849.8 |
| 8000x10000 | -19879.91 | 36559.6 | -17568.3 | -19816.23 |

Таблица 1. Сравнение значений целевой функции в точках, найденых алгоритмами.

Как видно их этих таблиц, LinSup алгоритм ведет себя гораздо лучше в задачах с большими размерностями. Так, при размерности в 8000х10000 LinSup со значением ядра 0,999 обеспечивает почти такую же точность вычислений, как и Simplex алгоритм, но требует для этого почти в три раза меньше времени.

# Заключение

В ходе выполнения данной курсовой работы мной был изучен метод суперизации для задач линейного программирования. Было разработано приложение, демонтирующее работу этого метода и позволяющее сравнить его с классическим Simplex алгоритмом. Мне удалось изучить научную статью на английском языке, перевести ее на русский и повторить некоторые результаты, которые удалось достигнуть автору статьи.

Результатом выполнения данной курсовой работы стало приложение для операционной системы Android, позволяющее производить расчет задач линейной оптимизации двумя способами и сравнивать результаты их работы.

# Приложение 1. Исходный код программы

**package** ru.grit.egor.linsup.utils;

**import** android.util.Log;  
  
**import** org.apache.commons.math3.exception.MaxCountExceededException;  
**import** org.apache.commons.math3.optimization.GoalType;  
**import** org.apache.commons.math3.optimization.PointValuePair;  
**import** org.apache.commons.math3.optimization.linear.LinearConstraint;  
**import** org.apache.commons.math3.optimization.linear.LinearObjectiveFunction;  
**import** org.apache.commons.math3.optimization.linear.Relationship;  
**import** org.apache.commons.math3.optimization.linear.SimplexSolver;  
**import** org.apache.commons.math3.optimization.linear.UnboundedSolutionException;  
  
**import** java.util.ArrayList;  
**import** java.util.Collection;  
**import** java.util.HashMap;  
  
**public class** LinSupSolver {  
  
 **final static** String ***TAG*** = LinSupSolver.**class**.getSimpleName();  
  
 ArrayList<ArrayList<Double>> **yk** = **new** ArrayList<>();  
  
 HashMap<Integer, Integer> **lk** = **new** HashMap<>();  
  
 ArrayList<ArrayList<Double>> **A** = **new** ArrayList<>();  
 ArrayList<Double> **B** = **new** ArrayList<>();  
 ArrayList<Double> **C** = **new** ArrayList<>();  
 ArrayList<Double> **y0** = **new** ArrayList<>();  
 **int N** = 10;  
 **int I** = 4;  
 **int J** = 4;  
 Double **a** = 0.999;  
 Double **lambda** = 2.99;  
 Double **e** = 1.0E-10;  
 Double **e2** = 1.0E-5;  
  
  
 PointValuePair **solution** = **null**;  
  
 **private** callback **mCallback**;  
  
 **public void** setmCallback(callback callback) {  
 **mCallback** = callback;  
 }  
  
 **public interface** callback {  
 **void** error(String error);  
 **void** simplexResult(String result);  
 **void** linSupResult(String result);  
 }  
  
  
 */\*@Override  
 protected void onCreate(@Nullable Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_start);  
  
 //init\_test2();  
 for (int iter = 0; iter < N; iter++) {  
 yk = new ArrayList<>();  
 lk = new HashMap<>();  
 A = new ArrayList<>();  
 B = new ArrayList<>();  
 C = new ArrayList<>();  
 y0 = new ArrayList<>();  
  
 Log.i(TAG, "-------------------------------------------------------------------");  
 Log.i(TAG, "Iteration " + iter);  
 init();  
 try {  
 try {  
 int i = simplex();  
 printSimplex(i);  
 } catch (MaxCountExceededException e ) {  
 Log.i(TAG, "Max Count Exceeded Exception");  
 }  
  
 startLinSup();  
 printLinSup();  
 } catch (UnboundedSolutionException e ) {  
 Log.i(TAG, "Unbounded Solution Exception");  
 }  
 }  
 Log.i(TAG, "-------------------------------------------------------------------");  
 }\*/* **void** clear() {  
 **yk** = **new** ArrayList<>();  
 **lk** = **new** HashMap<>();  
 **A** = **new** ArrayList<>();  
 **B** = **new** ArrayList<>();  
 **C** = **new** ArrayList<>();  
 **y0** = **new** ArrayList<>();  
 **solution** = **null**;  
 }  
  
  
 **public** ArrayList<Double> solveForCompare1() {  
 ArrayList<Double> result = **new** ArrayList<>();  
 clear();  
 **I** = 80;  
 **J** = 100;  
 initRandom();  
 **try** {  
 **int** i = simplex();  
 printSimplex(i);  
 result.add(**solution**.getValue());  
 } **catch** (Exception e) {  
 Log.*i*(***TAG***, e.toString());  
 }  
 clear();  
 **I** = 200;  
 **J** = 150;  
 initRandom();  
 **try** {  
 **int** i = simplex();  
 printSimplex(i);  
 result.add(**solution**.getValue());  
 } **catch** (Exception e) {  
 Log.*i*(***TAG***, e.toString());  
 }  
 clear();  
 **I** = 400;  
 **J** = 500;  
 initRandom();  
 **try** {  
 **int** i = simplex();  
 printSimplex(i);  
 result.add(**solution**.getValue());  
 } **catch** (Exception e) {  
 Log.*i*(***TAG***, e.toString());  
 }  
 */\*clear();  
 I = 800;  
 J = 1000;  
 initRandom();  
 try {  
 int i = simplex();  
 printSimplex(i);  
 result.add(solution.getValue());  
 } catch (Exception e) {  
 Log.i(TAG, e.toString());  
 }\*/* **return** result;  
 }  
  
  
 **public void** initAndSolve(**int** ii, **int** jj, ArrayList<Double> bb, ArrayList<Double> cc, ArrayList<Double> yy, ArrayList<ArrayList<Double>> aa) {  
  
 **yk** = **new** ArrayList<>();  
 **lk** = **new** HashMap<>();  
  
 **I** = ii;  
 **J** = jj;  
 **C** = cc;  
 **B** = bb;  
 **y0** = yy;  
 **A** = aa;  
 *//init\_test2();* **try** {  
 **try** {  
 **int** i = simplex();  
 printSimplex(i);  
 } **catch** (MaxCountExceededException e ) {  
 Log.*i*(***TAG***, **"Max Count Exceeded Exception"**);  
 **if** (**mCallback** != **null**) {  
 **mCallback**.error(**"Max Count Exceeded Exception"**);  
 }  
 }  
  
 startLinSup();  
 printLinSup();  
 } **catch** (UnboundedSolutionException e ) {  
 Log.*i*(***TAG***, **"Unbounded Solution Exception"**);  
 **if** (**mCallback** != **null**) {  
 **mCallback**.error(**"Unbounded Solution Exception"**);  
 }  
 }  
 }  
  
 **private void** init\_test1() {  
 **I** = 3;  
 **J** = 2;  
 **C**.addAll(LinSupUtils.*toList*(**new double**[] { 1, 1} ));  
 **B**.addAll(LinSupUtils.*toList*(**new double**[] { 16.0, 10.0, 8.0} ));  
  
 **for** (**int** i = 0; i < **I**; i++) {  
 **A**.add(**new** ArrayList<Double>());  
 }  
 **A**.get(0).addAll(LinSupUtils.*toList*(**new double**[] {1, 0} ));  
 **A**.get(1).addAll(LinSupUtils.*toList*(**new double**[] {0, 1 } ));  
 **A**.get(2).addAll(LinSupUtils.*toList*(**new double**[] {0.5, 0.4 } ));  
  
  
 *// LinSup init-s* **for** (**int** j = 0; j < **J**; j++) {  
 **y0**.add(10.0);  
 }  
 }  
  
 **private void** init\_test2() {  
 **I** = 5;  
 **J** = 5;  
 **C**.addAll(LinSupUtils.*toList*(**new double**[] { -0.6, 1.9, -1.2, 2.6, 0.1} ));  
 **B**.addAll(LinSupUtils.*toList*(**new double**[] { 10.5, 10.2, 10.5, 12.0, 11.6} ));  
  
 **for** (**int** i = 0; i < **I**; i++) {  
 **A**.add(**new** ArrayList<Double>());  
 }  
 **A**.get(0).addAll(LinSupUtils.*toList*(**new double**[] {0.5, -1.2, 0.8, 1.9, 1.1} ));  
 **A**.get(1).addAll(LinSupUtils.*toList*(**new double**[] {0.9, 0.2, -1.8, -0.7, 0.3} ));  
 **A**.get(2).addAll(LinSupUtils.*toList*(**new double**[] {2.0, -1.4, 0.5, 1.1, 1.9} ));  
 **A**.get(3).addAll(LinSupUtils.*toList*(**new double**[] {1.5, -0.3, -0.6, 2.0, 0.4} ));  
 **A**.get(4).addAll(LinSupUtils.*toList*(**new double**[] {-0.7, 1.9, 1.1, 1.0, 1.6} ));  
  
 *// LinSup init-s* **for** (**int** j = 0; j < **J**; j++) {  
 **y0**.add(10.0);  
 }  
 }  
  
 **private void** initRandom() {  
 **for** (**int** j = 0; j < **J**; j++) {  
 **y0**.add(10.0);  
 }  
 **for** (**int** j = 0; j < **J**; j++) {  
 **C**.add(LinSupUtils.*getRandom*(-2.0, 3.0));  
 }  
 **for** (**int** i = 0; i < **I**; i++) {  
 **A**.add(**new** ArrayList<Double>());  
 **for** (**int** j = 0; j < **J**; j++) {  
 **A**.get(i).add(LinSupUtils.*getRandom*(-1.0,2.0));  
 }  
 }  
  
 **for** (**int** i = 0; i < **I**; i++) {  
 **B**.add(LinSupUtils.*getRandom*(5.0,15.0));  
 }  
 }  
  
 **private boolean stop** = **false**;  
 **private void** startLinSup() {  
 **stop** = **false**;  
 **int** k = 0; *// k <- 0* **yk**.add(**new** ArrayList<Double>()); *// yk(0) <- y0* **yk**.get(k).addAll(**y0**);  
  
 **lk**.put(-1, 0); *// lk(-1) <- 0* HashMap<Integer, HashMap<Integer, ArrayList<Double>>> ykn = **new** HashMap<>();  
  
 **while** (getProximityFun(**yk**.get(k)) > **e** ||  
 (k==0 || getStoppingRule(**yk**.get(k-1),**yk**.get(k)) > **e2**)) {  
  
 **int** n = 0; *// n <- 0* **int** l = LinSupUtils.*getRandom*(k, **lk**.get(k-1)); *// l <- rand(k,lk(k-1)* ykn.put(k, **new** HashMap<Integer, ArrayList<Double>>()); *// ykn <- yk* ykn.get(k).put(n, **new** ArrayList<Double>());  
 ykn.get(k).get(n).addAll(**yk**.get(k));  
  
 **while** (n < **N**) {  
  
 Double betta = Math.*pow*(**a**, l); *// Bkn <- nl* ArrayList<Double> z = LinSupUtils.*getZ*(ykn.get(k).get(n), betta, **C**); *// z <- ykn - betta\* c / ||c||2* n = n + 1; *// n <- n+1* ykn.get(k).put(n, **new** ArrayList<Double>());  
 ykn.get(k).get(n).addAll(z); *// ykn <- z* l = l + 1; *// l <- l+1* }  
  
  
 **lk**.put(k, l); *// lk <- l* **yk**.add(**new** ArrayList<Double>()); *// y(k+1) <- A (ykn(k,N))* **yk**.get(k+1).addAll(getNextYbyA(ykn.get(k).get(**N**)));  
  
 k = k+1; *// k <- k+1* }  
  
 }  
  
 **private double** getStoppingRule(ArrayList<Double> prev, ArrayList<Double> next) {  
 ArrayList<Double> temp = LinSupUtils.*vectorsDifference*(next, prev);  
 **double** result = LinSupUtils.*secondVectorNorm*(temp) / LinSupUtils.*secondVectorNorm*(next);  
 *//Log.d(TAG, "getStoppingRule " + result);* **return** result;  
 }  
  
 **private double** getProximityFun(ArrayList<Double> x) {  
 **double** sum1 = 0.0;  
  
 **for** (**int** i = 0; i < **I**; i++) {  
 **double** val = Math.*max*((LinSupUtils.*innerProduction*(**A**.get(i),x) - **B**.get(i)), 0);  
 val = val\*val;  
 **double** val2 = 0.0;  
 **for** (**int** j = 0; j < **J**; j++) {  
 val2 += Math.*pow*(**A**.get(i).get(j), 2);  
 }  
  
 sum1 += val/val2;  
 }  
 sum1 = sum1 / (2 \* **I**);  
  
 **double** sum2 = 0.0;  
 **for** (**int** j = 0; j < **J**; j++) {  
 **double** val = -x.get(j);  
 val = Math.*max*(val, 0);  
 val = Math.*pow*(val, 2);  
 sum2 += val;  
 }  
 sum2 = sum2 / (2 \* **J**);  
  
 *//Log.d(TAG, "getProximityFun = " + (sum1+sum2));* **return** sum1+sum2;  
 }  
  
 **private** ArrayList<Double> getPHi(ArrayList<Double> z, **int** i) {  
 ArrayList<Double> result = **new** ArrayList<>();  
 result.addAll(z);  
  
 **if** (LinSupUtils.*innerProduction*(**A**.get(i),z) > **B**.get(i)) {  
 Double temp = (LinSupUtils.*innerProduction*(**A**.get(i),z) - **B**.get(i));  
 temp = (temp) / LinSupUtils.*secondVectorNorm*(**A**.get(i));  
 result = LinSupUtils.*vectorsDifference*(result, LinSupUtils.*production*(**A**.get(i), temp));  
 }  
 **return** result;  
 }  
  
  
 **private** ArrayList<Double> getNextYbyA(ArrayList<Double> prev) {  
 ArrayList<Double> result = **new** ArrayList<>();  
 **for** (**int** j = 0; j < **J**; j++) {  
 result.add(0.0);  
 }  
  
 **for** (**int** i = 0; i < **I**; i++) {  
 result = LinSupUtils.*vectorsSum*(result, LinSupUtils.*vectorsDifference*(getPHi(prev, i), prev));  
 }  
 result = LinSupUtils.*production*(result, (**lambda**/**I**));  
 result = LinSupUtils.*vectorsSum*(prev, result);  
  
  
  
 **if** (getProximityFun(result) < **e**) {  
 **stop** = **true**;  
 }  
  
 **for** (**int** a = 0; a < result.size(); a++) {  
 **if** (result.get(a) < 0.0) {  
 result.set(a,0.0);  
 }  
 }  
 *//getProximityFun(result);* **return** result}  
  
  
 **private int** simplex() {  
  
 *// describe the optimization problem* LinearObjectiveFunction f = **new** LinearObjectiveFunction(LinSupUtils.*toArray*(**C**), 0);  
 Collection<LinearConstraint> constraints = **new** ArrayList();  
 **for** (**int** i = 0; i < **I**; i++) {  
 constraints.add(**new** LinearConstraint(LinSupUtils.*toArray*(**A**.get(i)), Relationship.***LEQ***, **B**.get(i)));  
 }  
  
 *// create and run the solver* SimplexSolver solver = **new** SimplexSolver();  
 solver.setMaxIterations(10000);  
 **solution** = solver.optimize(f, constraints, GoalType.***MINIMIZE***, **true**);  
 **return** solver.getIterations();  
 }  
  
  
 **private void** printSimplex(**int** iters) {  
 **if** (**solution** != **null**) {  
 String log = **"Simplex MINIMIZE"**;  
 Log.*i*(***TAG***, **"----------- Simplex function MINIMIZE ---------"**);  
 *//Log.i(TAG, "STEPS: " + iters);* Log.*i*(***TAG***, **"f(x) = "** + **solution**.getValue());  
 log += **"\n"** + **"f(x) = "** + **solution**.getValue();  
 **for** (**int** j = 0; j < **J**; j++) {  
 Log.*i*(***TAG***, **"x("**+j+**") = "** + **solution**.getPoint()[j]);  
 log += **"\n"** + **"x("**+j+**") = "** + **solution**.getPoint()[j];  
 }  
 Log.*i*(***TAG***, **"----------- -------------------------- ---------"**);  
 log+=**"\n--------------------"**;  
 **if** (**mCallback** != **null**) {  
 **mCallback**.simplexResult(log);  
 }  
 }  
 }  
  
 **private void** printLinSup() {  
 Log.*i*(***TAG***, **"----------- LinSup function MINIMIZE ---------"**);  
 String log = **"LinSup MINIMIZE"**;  
 *//Log.i(TAG, "STEPS: " + (yk.size() - 1));* Log.*i*(***TAG***, **"f(x) = "** + LinSupUtils.*innerProduction*(**C**, **yk**.get(**yk**.size() - 1)));  
 log += **"\n"** + **"f(x) = "** + LinSupUtils.*innerProduction*(**C**, **yk**.get(**yk**.size() - 1));  
 **for** (**int** j = 0; j < **J**; j++) {  
 Log.*i*(***TAG***, **"x("** + j + **") = "** + **yk**.get(**yk**.size() - 1).get(j));  
 log+= **"\n"** + **"x("** + j + **") = "** + **yk**.get(**yk**.size() - 1).get(j);  
 }  
 Log.*i*(***TAG***, **"----------- -------------------------- ---------"**);  
 log+=**"\n--------------------"**;  
 **if** (**mCallback** != **null**) {  
 **mCallback**.simplexResult(log);  
 }  
  
 }  
  
}

**package** ru.grit.egor.linsup.utils;  
  
**import** android.util.Log;  
  
**import** java.util.ArrayList;  
**import** java.util.Random;  
  
**public class** LinSupUtils {  
  
 **final static** String ***TAG*** = LinSupUtils.**class**.getSimpleName();  
  
 **public static double** innerProduction(ArrayList<Double> vec1, ArrayList<Double> vec2) {  
 Double prod = 0.0;  
 **if** (vec1.size() != vec2.size() || vec1.size() == 0) {  
 Log.*e*(***TAG***, **"innerProduction : bad vectors size!"**);  
 **return** prod;  
 }  
 **for** (**int** i = 0; i < vec1.size(); i++) {  
 prod += vec1.get(i)\*vec2.get(i);  
 }  
 *//Log.e(TAG, "innerProduction : prod = " + prod);* **return** prod;  
 }  
  
 **public static double** secondVectorNorm(ArrayList<Double> vector) {  
 Double norm = 0.0;  
 **if** (vector.size() == 0) {  
 Log.*e*(***TAG***, **"secondVectorNorm : bad vector size"**);  
 **return** norm;  
 }  
  
 **for** (Double a : vector) {  
 norm += a\*a;  
 }  
  
 *//Log.e(TAG, "secondVectorNorm : norm = " + norm);* **return** Math.*sqrt*(norm);  
 }  
  
 **public static** ArrayList<Double> production(ArrayList<Double> vec, Double x) {  
 **if** (vec.size() == 0) {  
 Log.*e*(***TAG***, **"production : bad vector size!"**);  
 **return** vec;  
 }  
  
 ArrayList<Double> result = **new** ArrayList<Double>();  
  
 **for** (Double elem : vec) {  
 result.add(elem\*x);  
 }  
  
 **return** result;  
 }  
  
 **public static** ArrayList<Double> vectorsDifference(ArrayList<Double> vec1, ArrayList<Double> vec2) {  
  
 **if** (vec1.size() != vec2.size() || vec1.size() == 0) {  
 Log.*e*(***TAG***, **"vectorsDifference : bad vectors size!"**);  
 **return** vec1;  
 }  
  
 ArrayList<Double> result = **new** ArrayList<Double>();  
  
 **for** (**int** i = 0; i < vec1.size(); i++) {  
 result.add(vec1.get(i) - vec2.get(i));  
 }  
  
 **return** result;  
 }  
  
 **public static** ArrayList<Double> vectorsSum(ArrayList<Double> vec1, ArrayList<Double> vec2) {  
  
 **if** (vec1.size() != vec2.size() || vec1.size() == 0) {  
 Log.*e*(***TAG***, **"vectorsSum : bad vectors size!"**);  
 **return** vec1;  
 }  
  
 ArrayList<Double> result = **new** ArrayList<Double>();  
  
 **for** (**int** i = 0; i < vec1.size(); i++) {  
 result.add(vec1.get(i) + vec2.get(i));  
 }  
  
 **return** result;  
 }  
  
 **public static int** getRandom(**int** low, **int** high) {  
 **if** (high <= low) {  
 Log.*e*(***TAG***, **"getRandom : bad values!"**);  
 **return** 0;  
 }  
 **if** (high == 0 && low == 0) {  
 Log.*i*(***TAG***, **"getRandom : zero values!"**);  
 **return** 0;  
 }  
 Random r = **new** Random();  
 **int** result = r.nextInt(high-low) + low;  
 **return** result;  
 }  
  
 **public static double** getRandom(**double** low, **double** high) {  
 **if** (high < low) {  
 Log.*e*(***TAG***, **"getRandom : bad values!"**);  
 }  
 **if** (high == 0.0 && low == 0.0) {  
 Log.*i*(***TAG***, **"getRandom : zero values!"**);  
 **return** 0;  
 }  
 Random r = **new** Random();  
 **double** result = low + (high - low) \* r.nextDouble();  
 **return** result;  
 }  
  
 **public static** ArrayList<Double> getZ (ArrayList<Double> y, Double betta, ArrayList<Double> c) {  
 **if** (y.size() != c.size() || y.size() == 0) {  
 Log.*e*(***TAG***, **"getZ : bad vectors size!"**);  
 **return** y;  
  
 }  
  
 Double k = betta \* (1.0 / *secondVectorNorm*(c));  
 ArrayList<Double> temp = *production*(c, k);  
  
 ArrayList<Double> z = *vectorsDifference*(y,temp);  
 **return** z;  
 }  
  
 **public static double**[] toArray(ArrayList<Double> list) {  
 **double**[] array = **new double**[list.size()];  
 **for** (**int** i = 0; i < list.size(); i++) {  
 array[i] = (**double**) list.get(i);  
 }  
 **return** array;  
 }  
  
 **public static** ArrayList<Double> toList(**double**[] array) {  
 ArrayList<Double> list = **new** ArrayList<>();  
 **for** (**double** i : array) {  
 list.add(i);  
 }  
 **return** list;  
 }  
  
  
}

**package** ru.grit.egor.linsup.activity;  
  
**import** android.graphics.Color;  
**import** android.os.Bundle;  
**import** android.support.annotation.Nullable;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.view.View;  
**import** android.widget.LinearLayout;  
**import** android.widget.ProgressBar;  
**import** android.widget.TextView;  
  
**import** com.androidplot.xy.BoundaryMode;  
**import** com.androidplot.xy.LineAndPointFormatter;  
**import** com.androidplot.xy.PanZoom;  
**import** com.androidplot.xy.SimpleXYSeries;  
**import** com.androidplot.xy.StepMode;  
**import** com.androidplot.xy.XYGraphWidget;  
**import** com.androidplot.xy.XYPlot;  
**import** com.androidplot.xy.XYSeries;  
  
**import** java.text.FieldPosition;  
**import** java.text.Format;  
**import** java.text.ParsePosition;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.regex.Matcher;  
  
**import** ru.grit.egor.linsup.R;  
**import** ru.grit.egor.linsup.utils.LinSupSolver;  
**import** ru.grit.egor.linsup.utils.LinSupUtils;  
  
**public class** CompareActivity **extends** AppCompatActivity {  
  
 **private** XYPlot **plot**;  
 **private** ProgressBar **progressBar**;  
 **private** LinearLayout **table**;  
 ArrayList<Double> **simplex** = **new** ArrayList<>();  
 ArrayList<Double> **a09** = **new** ArrayList<>();  
 ArrayList<Double> **a099** = **new** ArrayList<>();  
 ArrayList<Double> **a0999** = **new** ArrayList<>();  
  
 @Override  
 **public void** onBackPressed() {  
 **if** (**plot**.getVisibility() == View.***VISIBLE***) {  
 **table**.setVisibility(View.***VISIBLE***);  
 **plot**.setVisibility(View.***GONE***);  
 } **else** {  
 **super**.onBackPressed();  
 }  
 }  
  
 @Override  
 **protected void** onCreate(@Nullable Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_compare***);  
  
 **plot** = (XYPlot) findViewById(R.id.***plot***);  
 **progressBar** = (ProgressBar) findViewById(R.id.***progress***);  
 **table** = (LinearLayout) findViewById(R.id.***table***);  
 findViewById(R.id.***button3***).setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **table**.setVisibility(View.***GONE***);  
 **plot**.setVisibility(View.***VISIBLE***);  
 showPlot();  
 }  
 });  
  
 **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 LinSupSolver solver = **new** LinSupSolver();  
 **simplex** = solver.solveForCompare1();  
runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **progressBar**.setVisibility(View.***GONE***);  
 TextView sim\_tv = (TextView) findViewById(R.id.***simplex***);  
 TextView lin1 = (TextView) findViewById(R.id.***linsup\_1***);  
 TextView lin2 = (TextView) findViewById(R.id.***linsup\_2***);  
 TextView lin3 = (TextView) findViewById(R.id.***linsup\_3***);  
 **for** (**int** i = 0; i < **simplex**.size(); i++) {  
 sim\_tv.setText(sim\_tv.getText()+**"\n"**+**simplex**.get(i));  
 lin1.setText(lin1.getText()+**"\n"**+**a09**.get(i));  
 lin2.setText(lin2.getText()+**"\n"**+**a099**.get(i));  
 lin3.setText(lin3.getText()+**"\n"**+**a0999**.get(i));  
 }  
 }  
 });  
 }  
 }).start();  
 }  
  
  
 **private void** showPlot() {  
  
 **plot**.setVisibility(View.***VISIBLE***);  
 **plot**.setRangeBoundaries(0.0001, 1.0, BoundaryMode.***GROW***);  
 **plot**.getGraph().setLinesPerDomainLabel(1);  
 *//PanZoom.attach(plot);  
 //PanZoom.attach(plot, PanZoom.Pan.HORIZONTAL, PanZoom.Zoom.STRETCH\_HORIZONTAL);* PanZoom.*attach*(**plot**, PanZoom.Pan.***BOTH***, PanZoom.Zoom.***STRETCH\_VERTICAL***);  
 *// plot.setRange* **final** String[] domainLabels = {**"80x100"**, **"200x250"**, **"400x500"***/\*, "800x1000"\*/*};  
  
 ArrayList<Double> re1 = **new** ArrayList<>();  
 ArrayList<Double> re2 = **new** ArrayList<>();  
 ArrayList<Double> re3 = **new** ArrayList<>();  
 **for** (**int** i = 0; i < **simplex**.size(); i++) {  
 re1.add(Math.*abs*(**a09**.get(i) - **simplex**.get(i))/ Math.*abs*(**simplex**.get(i)));  
 re2.add(Math.*abs*(**a099**.get(i) - **simplex**.get(i))/ Math.*abs*(**simplex**.get(i)));  
 re3.add(Math.*abs*(**a0999**.get(i) - **simplex**.get(i))/ Math.*abs*(**simplex**.get(i)));  
 }  
  
 *//Number[] series3Numbers = {0.0002, 0.0005, 0.0008, 0.0013};* XYSeries series1 = **new** SimpleXYSeries(  
 re1, SimpleXYSeries.ArrayFormat.***Y\_VALS\_ONLY***, **"a=0.9"**);  
 XYSeries series2 = **new** SimpleXYSeries(  
 re2, SimpleXYSeries.ArrayFormat.***Y\_VALS\_ONLY***, **"a=0.99"**);  
 XYSeries series3 = **new** SimpleXYSeries(  
 re3, SimpleXYSeries.ArrayFormat.***Y\_VALS\_ONLY***, **"a=0.999"**);  
  
 LineAndPointFormatter series1Format =  
 **new** LineAndPointFormatter(Color.***RED***, Color.***GREEN***, **null**, **null**);  
  
 LineAndPointFormatter series2Format =  
 **new** LineAndPointFormatter(Color.***BLUE***, Color.***RED***, **null**, **null**);  
  
 LineAndPointFormatter series3Format =  
 **new** LineAndPointFormatter(Color.***CYAN***, Color.***BLUE***, **null**, **null**);  
  
  
 **plot**.addSeries(series1, series1Format);  
 **plot**.addSeries(series2, series2Format);  
 **plot**.addSeries(series3, series3Format);  
  
 **plot**.getGraph().getLineLabelStyle(XYGraphWidget.Edge.***BOTTOM***).setFormat(**new** Format() {  
 @Override  
 **public** StringBuffer format(Object obj, StringBuffer toAppendTo, FieldPosition pos) {  
 **int** i = Math.*round*(((Number) obj).floatValue());  
 **return** toAppendTo.append(domainLabels[i]);  
 }  
 @Override  
 **public** Object parseObject(String source, ParsePosition pos) {  
 **return null**;  
 }  
 });  
 }  
}

**package** ru.grit.egor.linsup.activity;  
  
**import** android.content.Context;  
**import** android.os.Bundle;  
**import** android.support.annotation.Nullable;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.text.Editable;  
**import** android.text.TextWatcher;  
**import** android.util.Log;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.LinearLayout;  
**import** android.widget.TextView;  
  
**import** java.util.ArrayList;  
  
**import** ru.grit.egor.linsup.R;  
**import** ru.grit.egor.linsup.utils.LinSupSolver;  
  
**public class** CustomProblemActivity **extends** AppCompatActivity {  
  
 **static final** String ***TAG*** = CustomProblemActivity.**class**.getSimpleName();  
  
 Context **mContext**;  
 TextView **text\_I**;  
 TextView **text\_J**;  
 TextView **text\_C**;  
 TextView **text\_B**;  
 TextView **text\_Y**;  
 LinearLayout **matrix\_A**;  
 TextView **text\_result**;  
 Button **solve**;  
  
 @Override  
 **protected void** onCreate(@Nullable Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_custom\_problem***);  
 **mContext** = **this**;  
 **text\_I** = (TextView) findViewById(R.id.***i***);  
 **text\_J** = (TextView) findViewById(R.id.***j***);  
 **text\_C** = (TextView) findViewById(R.id.***c***);  
 **text\_B** = (TextView) findViewById(R.id.***b***);  
 **text\_Y** = (TextView) findViewById(R.id.***y0***);  
 **text\_result** = (TextView) findViewById(R.id.***result***);  
 **matrix\_A** = (LinearLayout) findViewById(R.id.***matrix\_a***);  
 **solve** = (Button) findViewById(R.id.***solve***);  
 **solve**.setOnClickListener(**solveClicked**);  
  
 **text\_I**.addTextChangedListener(**textWatcher**);  
 **text\_J**.addTextChangedListener(**textWatcher**);  
 }  
  
 View.OnClickListener **solveClicked** = **new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **try** {  
 **text\_result**.setText(**""**);  
 **final int** i = Integer.*parseInt*(**text\_I**.getText().toString());  
 **final int** j = Integer.*parseInt*(**text\_J**.getText().toString());  
 **final** ArrayList<Double> c = getDoubleArray(**text\_C**.getText().toString());  
 **final** ArrayList<Double> b = getDoubleArray(**text\_B**.getText().toString());  
 **final** ArrayList<Double> y0 = getDoubleArray(**text\_Y**.getText().toString());  
 **if** (c == **null** || b == **null** || y0 == **null**) {  
 **return**;  
 }  
  
 **final** ArrayList<ArrayList<Double>> A = **new** ArrayList<>();  
 **for** (**int** count = 0; count < **matrix\_A**.getChildCount(); count++) {  
 ArrayList<Double> temp = getDoubleArray(((EditText) **matrix\_A**.getChildAt(count)).getText().toString());  
 **if** (temp != **null**) {  
 A.add(temp);  
 } **else** {  
 **return**;  
 }  
 }  
  
 Log.*d*(***TAG***, **"all done! start solve"**);  
  
 **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 LinSupSolver solver = **new** LinSupSolver();  
 solver.setmCallback(**new** LinSupSolver.callback() {  
 @Override  
 **public void** error(**final** String error) {  
 runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **text\_result**.setText(error);  
 }  
 });  
 }  
  
 @Override  
 **public void** simplexResult(**final** String result) {  
 runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **text\_result**.setText(**text\_result**.getText().toString()+**"\n"**+result);  
 }  
 });  
 }  
  
 @Override  
 **public void** linSupResult(**final** String result) {  
 runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **text\_result**.setText(**text\_result**.getText().toString()+**"\n"**+result);  
 }  
 });  
 }  
 });  
 solver.initAndSolve(i,j,b,c,y0,A);  
 }  
 }).start();  
  
 } **catch** (Exception e) {  
 Log.*d*(***TAG***, **"solveClicked parse exception"**);  
 }  
  
 }  
 };  
  
 **private** ArrayList<Double> getDoubleArray(String text) {  
 **try** {  
 ArrayList<Double> result = **new** ArrayList<>();  
 text.replaceAll(**" "**, **""**);  
 String[] array = text.split(**","**);  
 **for** (**int** i = 0; i < array.**length**; i++) {  
 result.add(Double.*parseDouble*(array[i]));  
 }  
 **return** result;  
 } **catch** (Exception e) {  
 Log.*d*(***TAG***, **"getDoubleArray parse exception"**);  
 }  
 **return null**;  
 }  
  
 TextWatcher **textWatcher** = **new** TextWatcher() {  
 @Override  
 **public void** beforeTextChanged(CharSequence s, **int** start, **int** count, **int** after) {  
  
 }  
  
 @Override  
 **public void** onTextChanged(CharSequence s, **int** start, **int** before, **int** count) {  
  
 }  
  
 @Override  
 **public void** afterTextChanged(Editable s) {  
 **try** {  
 **int** size\_i = Integer.*parseInt*(**text\_I**.getText().toString());  
 **int** size\_j = Integer.*parseInt*(**text\_J**.getText().toString());  
 String text = **""**;  
 **if** (size\_j > 0) {  
 text += **"0"**;  
 **for** (**int** i = 1; i < size\_j; i++) {  
 text += **", 0"**;  
 }  
 }  
 **matrix\_A**.removeAllViews();  
 **for** (**int** i = 0 ; i < size\_i; i++) {  
 EditText view = **new** EditText(**mContext**);  
 view.setText(text);  
 **matrix\_A**.addView(view);  
 }  
 } **catch** (NumberFormatException e) {  
 Log.*d*(***TAG***, **"afterTextChanged parse exception"**);  
 }  
 }  
 };  
}

# Приложение 2. Перевод статьи

Может ли линейная суперизация быть полезной в задачах линейной оптимизации?

Yair Censor

Department of Mathematics

University of Haifa

Mt. Carmel, Haifa 3498838, Israel

([yair@math.haifa.ac.il](mailto:yair@math.haifa.ac.il))

March 17, 2016. Revised: July 19, 2016. Revised: September 27, 2016.

# Реферат

Линейная суперизация рассматривает задачи линейного программирования, но вместо того, чтобы пытаться решить их с помощью линейных методов оптимизации, этот метод использует возмущения (колебания) устойчивости алгоритмов поиска для нахождения меньших (не обязательно минимальных) значений целевой функции. В данной работе исследуются два вопроса. (1) Может ли LinSup обеспечить нахождения допустимой точки, в которой значение целевой функции ниже, чем в точке, полученной при тех же условиях исходным алгоритмом, но без применения суперизации? (2) Какова «стоимость» LinSup алгоритма по сравнению с Simplex-алгоритмом решения задач ЛП? Основываясь на результатах наших вычислительных экспериментов, мы можем ответить на эти вопросы, как "да" и "вполне приемлемая", соответственно.

**Ключевые слова**: Суперизация, ограниченное возмущение устойчивости, линейная суперизация, линейное программирование, Simplex алгоритм, алгоритмы поиска, алгоритмический оператор, алгоритм Агмона-Моцкина-Шенберга, линейные неравенства, линейная оптимизация.

# Введение

В данной работе мы рассматриваем метод линейной суперизации в качестве инструмента для решения задач линейного программирования. Метод линейной суперизации не гарантирует нахождение точки минимума линейной задачи оптимизации, но он регулирует работу линейных алгоритмов, которые ищут точки с наименьшим значением целевой функции. Этот алгоритм применим не только к простым задачам поиска минимума функции, но к задачам больших размерностей. Он является хорошей альтернативой Simplex -методу линейного программирования, с которым мы и будем сравнивать его здесь.

Эта работа опирается на более ранние теоретические работы по суперизации, которые включены в список используемых источников, в частности, [18]. В качестве инструментов для доказательств используются наши экспериментальные расчеты. Несмотря на это, мы не обсуждаем вычислительные проблемы сами по себе, а используем вычисления в качестве инструментов для практической проверки, в стиле "доказательство концепции" (proof of concept).

**Что такое суперизация?** Многие методы оптимизации с ограничениями основаны на методах безусловной оптимизации, которые просто приспособлены для работы с ограничениями. Так, например, в градиентных методах безусловное уменьшение внутреннего шага "управляет" процессом, но происходит в допустимом множестве после каждого шага минимизации. Подобная проекция ограничений внутри метода приводит к необходимости решения нетривиальной задачи оптимизации и к необходимости решения ее в каждой итерации, что препятствует работе методов градиента и ограничивает их эффективность только на допустимых множествах. Барьерные или штрафные методы аналогичным образом основаны на безусловной оптимизации, но с различными "дополнениями", которые гарантируют, что ограничения выполнятся. Методы регуляризации встраивают ограничения в целевую функцию и продолжают работать как методы решения без ограничения уже для новой целевой функции.

В отличие от этих подходов, методы суперизации можно рассматривать как противоположную точку зрения. Вместо того чтобы адаптировать алгоритмы минимизации без ограничений для обработки ограничений, здесь адаптируются алгоритмы поиска для осуществления поисках минимума целевой функции. Это делается с сохранением исходного алгоритма, и не требует больших вычислений

**Польза подхода.** Полезность такого подхода зависит от двух особенностей: (1) **Вычислительной**: исходный алгоритм решения логической задачи менее требователен, чем поиск точки ограниченной минимизации в допустимом множестве. Таким образом, рассматриваемый метод позволяет эффективно модифицировать исходные алгоритмы с помощью недорогих дополнений, что хорошо работает на практике. (2) **Прикладной**: в некоторых существующих реальных приложениях выбор целевой функции является экзогенным от моделирования и сбора данных, что приводит к возникновению ограничений. В таких ситуациях, подобный выбор целевой функции часто приводит к тому, что нет никакой необходимости искать точку минимума в этих ограничениях. Для получения «приемлемых результатов», часто бывает достаточно найти допустимую точку, которая уменьшает значение целевой функции (но не минимизирует ее). В некоторых научных приложениях, подобные операции с целевыми функциями хоть и являются затратными, но играют центральную роль в модели, однако, в других случаях приведенные выше рассуждения могут быть полезны.

**Текущее исследование.** Текущее состояние работ по суперизации можно оценить из материалов на странице в сети Интернет [10]. В частности, [26] и [9]. Недавние исследования включают в себя множество отчетов, начиная от новых приложений в промышленной рентгеновской компьютерной томографии [39] и новые математические результаты на основе суперизации, таких как обоснование строгой монотонности фейеровских методов с использованием суперизации проекционных методов [18]. Подробное описание предыдущей работы, посвященной суперизации можно найти в [15, раздел 3].

**Линейная суперизация.** Линейная суперизация рассматривает задачи линейного программирования, но вместо того, чтобы пытаться решить их с помощью линейных методов оптимизации, этот метод использует возмущения (колебания) устойчивых алгоритмов поиска для нахождения меньших (не обязательно минимальных) значений целевой функции. В данной работе исследуются два вопроса. (1) Может ли LinSup обеспечить нахождения допустимой точки, в которой значение целевой функции ниже, чем в точке, полученной при тех же условиях исходным алгоритмом, но без применения суперизации? (2) Какова «стоимость» LinSup алгоритма по сравнению с Simplex-алгоритмом решения задач ЛП? Основываясь на результатах наших вычислительных экспериментов, мы можем ответить на эти вопросы, как "да" и "вполне приемлемая", соответственно.

Интересным и перспективным аспектом современных экспериментов является зависимость результатов от размеров тестовой задачи. Мы обнаружили, что преимущества LinSup становятся монотонно более выраженным с увеличением размерности проблемы. Мы рассмотрели задачи с 8000 линейных неравенств и векторами по 10000 компонентов, и заметили тенденцию, которая также сохраняется и за пределами этих размерностей. Таким образом, LinSup вполне может стать полезным инструментом для решения задач больших размеров. Следует признать, что наша предварительная работа, представленная здесь, основывается на случайно сгенерированных проблемах, и что они не являются типичными проблемами для задачи линейного программирования, которые решаются на протяжении многих лет.

В Разделе 4 мы покажем, что LinSup находит высокую допустимую точку, т.е. допустимую точку с более низким значением целевой функции. В Разделе 5 мы исследуем численное поведение LinSup в сравнении с классическим Simplex алгоритмом линейной оптимизации. Общие принципы суперизации рассматриваются в Разделе 2, а алгоритм LinSup представлен в Разделе 3. Результаты наших экспериментов были получены с помощью программы MATLAB [31] и представлены в Разделах 4 и 5. Мы делаем заключительные замечания в Разделе 6 и перечисляем различные вопросы для проведения дальнейших исследований по LinSup. Приложение (раздел 7) кратко описывает историю технических изменений и модификаций в суперизации базовых алгоритмов, опубликованные в литературе за последние несколько лет с момента создания.

# Методология суперизации

Рассмотрим пару (*М,* ***А***), где *М,* называемое целевым набором, является заданным подмножеством заданного подмножества *Q* J-мерного евклидового пространства: *M ⊂ Q ⊆ RJ*. Пусть ***А*** *: Q → Q* будет неким алгоритмическим оператором, который определяет итерационный процесс, называемый базовым алгоритмом,

*x0 ∈ Q, xk+1 =* ***A****(xk), k = 1, 2, …* (1)

задачей которого является найти точку в целевом множестве *М*. Мы, в дальнейшем, будем называть такую пару (*M,* ***A***) как пару суперизации. Пусть *: Q ⊆ RJ→ R -* заданная действительная функция, называется базовой функцией.

Методология суперизации предназначена для ограниченных задач уменьшения функции следующего вида:

**Проблема 1. Проблема уменьшения ограниченной функции**. Пусть *(М,* ***А****)* пара суперизации и пусть*: Q ⊆ RJ→ R* является базовой функцией. Необходимо найти точку *х\** в *М*, в которой значение функции меньше (обязательно минимально), чем в точке, которая может быть достигнута путем применения базового алгоритма поиска М.

Методология суперизации решает эту задачу путем исследования возмущений базового алгоритма, а затем с помощью предупреждения таких возмущений, "заставляет" базовый алгоритм, делать поправку шага в дополнение к своей первоначальной задаче уменьшение функции. Добавление таких возмущений в алгоритм называется "суперизация базового алгоритма".

Если базовый алгоритм является вычислительно эффективным и полезным при нахождении точки *М*, и если его возмущение упругое и может быть просто и недорого вычислено, то преимущество такого способа заключается в том, что мы можем решить задачу условной минимизации целевой функции, регулируя итерации в соответствии с возмущениями, по стоимости минимизации целевой функции без ограничений. Методология суперизации автоматически генерирует версию базового алгоритма с суперизацией. Вектор *х\**, полученный путем применения версии базового алгоритма с суперизацией, не обязательно должны минимизировать в *М*. Для получения более подробной информации о видах возмущений устойчивости, которые могут быть использованы, обратитесь, например, к [9, определения 4 и 9] или [13, 15, 27].

Приведенные выше определения и терминология основаны на том, что мы подразумеваем под выражением "Найти точку *х\** в *М*" в Проблеме 1. В слабой суперизации "найти точку в *М*" подразумевается, как получить бесконечную последовательность , которая сходится к точке х\*∈ *М*, при этом *М* должно быть непустым. При сильной суперизации "найти точку *М*" понимается, как найти точку х\*, которая ɛ-совместима с *M*, для некоторого положительного ɛ. То есть точку, функция в которой близка, или отличается от *М* на значение меньше или равно ɛ. Таким образом, непустоту *М* учитывать не нужно. Эти понятия были определены в работе [9].

На ум приходят два важных частных случая пар суперизации (*М,* ***А***), хотя и другие случаи также возможны.

**Случай 2**. Целевое множество *М* является множеством решений выпуклой задачи следующего вида: Найти вектор где C*i* ⊆ RJ - замкнутые выпуклые подмножества, таким образом, : В этом случае алгоритмический оператор и базовый алгоритм (1) может быть применен к широкому спектру алгоритмов поиска значений функции, примеры [3, 4, 8, 11, 12, 19].

**Случай 3.** Целевое множество *М* является множеством решений следующей задачи минимизации с ограничениями: минимизировать целевую функцию f в допустимой области , таким образом, что . В этом случае алгоритмический оператор и базовый алгоритм (1) может быть применимы к любому из широкого разнообразия алгоритмов минимизации с ограничениями, распространенных в литературе.

В данной работе мы рассматриваем линейную суперизацию, которая концентрируется на особой ситуации **Случая 2**, в которой все ограничения устанавливает *Ci* также, как и целевая линейная функция . Суперизация работает и с другими целевыми функциями, такими как функции полной вариации (TV), например, в [13, 15, 27]. Суперизация в **Случае 3**, где *М* представляет собой множество решений максимального правдоподобия задачи оптимизации, появляется в работах [23, 28, 30].

# Линейная суперизация

* 1. **Проблема и алгоритм**

Определим целевое множество *M* как*:*

(2)

где даны: I x J матрица действительных чисел и вектор .

В качестве базового алгоритма мы выбрали проекционные метод поиска. Проекции на наборы используются во многих методах в теории оптимизации, но здесь проекционные методы относятся к итерационным алгоритмам, которые используют проекции на наборы, опираясь на общий принцип - группы, как правило, замкнутых и выпуклых, наборов легче спроецировать на определенные наборы, чем на другие наборы (пересечений, наборов изображений при некоторой трансформации и т.д.), которые получены из отдельных наборов.

Проекционные методы могут иметь различные алгоритмические структуры, такие как блок-итерационные проекции (BIP), см, например, [22, 24] и ссылки в них, или усреднения строк по проекции (SAP), см, например, [17] и ссылки в них. Некоторые из них особенно хорошо подходят для параллельных вычислений, и демонстрируют хорошие показатели сходимости при приемлемых начальных моделях поведения. Этот класс алгоритмов достаточно успешно прогрессировал в последнее время, и с успехом применяется для многих научных, технологических и математических задач. Смотрите, например, обзор [3] 1996 года, или из последнего времени [11], ссылки [8] [12].

Важное замечание. Выпуклая проблема, упомянутая в **Случае 2**, может быть переведена в безусловную минимизацию с помощью некоторой функции близости, которая оценивает изменение точек алгоритма. Например, суммы евклидовых норм расстояний до множеств CFP как функция близости применяются в методах наискорейшего спуска, методах одновременной проекции CFP типа CIMMINO. Тем не менее, нет никакой функции близости, которая работала бы в методе последовательных проекций на CFP типа Качмажа см [2]. Таким образом, исследование осуществимости алгоритмов поиска для CFP происходит независимо от метода минимизации, смотрите ссылки, упомянутые выше. На протяжении многих лет исследователи пытались использовать методы прогнозирования для алгоритмов выпуклой задачи Л.П., например, см. книгу Chinneckís [20]. Мини-обзор отношений между алгоритмами линейного программирования в [34, § 1] проливает больше света на это. Наши результаты позволяют нам задаться вопросом, может ли LinSup служить причиной для этого.

Пусть целевой функцией для линейной суперизации будет

(3)

где – скалярное произведение вектора *x* и данного вектора .

Следуя общим принципам методологии суперизации, которые были представлены для любых целевых функций в предыдущих публикациях, смотрите, например, недавние обзоры [26] и [9], приведем следующий линейный алгоритм суперизация. На вход алгоритма подаются исходные данные проблемы *A, b*, и *c* из (2) и (3), соответственно. Так же выбранная пользователем начальная точка и ядро 0 < *a* <1 (смотри пункт 1 в п.3.3), с помощью которых алгоритм генерирует размеры шагов , в соответствии с целым числом *N* (смотрите пункт 7 в п.3.3). Все величины в алгоритме, которые еще не были определены или объяснены - подробно описаны в п.3.3 ниже.

**Алгоритм 4. Линейная Суперизация (LinSup)**

1. **установим**

2. **установим**

3. **установим**

4. **пока** не выполняется правило остановки

5. **установим**

6. **установим**

7. **установим**

8. **пока** n<N

9. **установим**

10. **установим**

11. **установим**

12. **установим**

13. **установим**

14. **выход**

15. **установим**

16. **установим**

17. **установим**

18. **выход**

* 1. **Алгоритм Эгмона-Моцкина-Шенберга как базовый алгоритм**

Мы использовали проекционный метод Эгмона-Моцкина-Шенберга (AMS) [1,32], см. [19, Алгоритм 5.4.2], в качестве базового алгоритма уменьшения функции ***A*** на шаге 16 Алгоритма 4. Обозначим полупространства, представляющие отдельные строкам в (2) Hi,

, (4)

где является i-й строкой матрицы A, а – i-я компонента вектора b в (2). Ортогональная проекция произвольной точки на *Hi* имеет следующую форму:

(5)

**Алгоритм 5. Метод релаксации Эгмона, Моцкина и Шенберга.**

**Инициализация:**  задается произвольно.

**Итерация:** Следующая итерация вектора ***x*** рассчитывается на основе текущей итерации *xk* следующим образом:

(6)

**Параметры релаксации:** Параметры λk выбраны такими, что , для всех , с некоторыми сколь угодно малыми .

**Контроль:** Контрольная последовательность зациклена на {1, 2…, I}.

Этот циклический алгоритм поиска AMS выполняется циклически по неравенствам в (2). Для того чтобы справиться с ограничениями неотрицательности в (2) мы просто берем текущий вектор итерации, после того, как прошли полный цикл AMS метода через все *I* неравенств, и устанавливаем его отрицательные компоненты в нуль при сохранении остальных неизменными.

Краеугольным камнем методологии суперизации в целом, а также для линейного случая обсуждаемого здесь, является возмущение устойчивости базового алгоритма, который используется в ней. Алгоритм AMS, как известно, имеет ограниченное возмущение сопротивляемости, об этом можно узнать из различных опубликованных ранее результатов, см. например [16, теорема 12], [33].

* 1. **Детали реализации и объяснения**

Детали реализации нашей реализации LinSup Алгоритма 4 приведены в следующих секциях:

1. **Размер шагов возмущений**. Размеры шагов в Алгоритме 4 должны быть такими, что , чтобы гарантировать сходимость суммирующей последовательности , смотрите, например [18]. Для этого в Алгоритме 4 предполагается, что мы имеем в распоряжении суммирующую последовательность положительных вещественных чисел, сформированных в виде , где . Одновременно с формированием итерационной последовательности , подпоследовательность используется для генерации размеров шагов на шаге 9 Алгоритма 4. Число называется ядром последовательности .
2. **Контроль уменьшения размеров шагов целевой функции.** Если во время выполнения Алгоритма 4 размер шагов уменьшается слишком быстро, тогда слишком быстро уменьшается значение целевой функции, которая тесно связана с базовым алгоритмом. Этот тонкий баланс можно регулировать с помощью выбора показателя и обновления значения , степени которого определяют размеры на шаге 9. В нашей работе мы придерживаемся стратегии обновления индекса *l*, которая была предложена и реализована в работах [38, стр 38 и Таблица 7.1 на странице 49] и в [29], соответственно. Вместо того, чтобы последовательно увеличивать *l* сохраняя его значение с последней итерации *N*, в нашей стратегии мы устанавливаем *l* в начале каждой новой итерации (шаги 5 и 6) в виде случайного числа между текущим индексом итерации *k* и значением *l* c последней итерации, т.е . Эта стратегия была названа в различных докладах как *"ATL2",* и убедившись в ее полезности для нашей работы мы использовали ее во всех наших экспериментах. С другой стороны, значение , степень которого определяет размеры шагов , было экспериментально определено и подтверждено практическими результатами в разделе экспериментов с LinSup далее в работе. Очевидно, что в алгоритме Simplex нет такой тонкой балансировки, хотя подобное было на ранних стадиях его развития (стратегии поворота, список кандидатов и т.д.). Необходима дальнейшая работа, чтобы сделать LinSup более устойчивыми к выбору параметров.
3. **Нет сравнения значений целевой функции.** Под влиянием результатов в [18], мы полностью убрали из алгоритма проверку процесса принятия решений, которая сравнивала значение целевой функции в точке *z* на шаге 10 при возмущениях внутреннего цикла со значением целевой функции в точке . Эта проверка присутствовала в процессе принятие решения во многих предыдущих формулировках базовых алгоритма с суперизацией, смотрите, например, шаг (ХIV) в "версия алгоритма ***Р*** с суперизацией" в [27, стр 5537]. Так как мы имеем возможность обосновать наши экспериментальные результаты без этой проверки и с помощью математических обоснований в [18], мы решили отказаться и продолжить работу без нее. Смотрите также Приложение в разделе 7 ниже.
4. **Функция близости.** Для измерения точности вычисления (или уровня согласия) точки относительно целевого множества M мы использовали следующую функцию близости:

(7)

в которой знак + обозначает, для любого действительного числа d, Функция близости инвариантно масштабируемая, поскольку она точно измеряет взвешенную (с равными весами) сумму полуквадратов расстояний точки *х* по всем линейным ограничениям в неравенствах. Эти расстояния (см. шаг (5)) являются геометрическими объектами, которые нечувствительны к масштабированию.

1. **Начальные точки алгоритмов**. В наших экспериментальных исследованиях начальные точки всегда выбирались таким образом, чтобы не выполнялось условие для целевого набора точек . В противном случае, существует опасность того, что алгоритм не будет сходиться из-за шагов 16 в алгоритме AMS. Это было достигнуто следующим образом. Первый был случайным образом выбраны в интервале [0; 1] автоматически или пользователем, близость этого выбора к множеству M вычисляется функцией (7). Если близость равна нулю, то мы выбираем новый и повторяем это приращение до тех пор, пока мы не найдем точку с ненулевой близостью, чтобы выбрать в качестве точки инициализации.
2. **Генерация тестовых проблем**. Мы создали целевые наборы *M* различных размеров и линейной целевой функцией и запустили на них Алгоритм 4 с суперизацией и без нее, как в нашей экспериментальной работе, описанной ниже. Каждая проблема размером *I x J* была создана путем определения матрицы *А*, элементы которой выбирались случайным образом в интервале [1, 2] для всех экспериментов. Вектор *с* был выбран случайным образом в интервале [2, 3]. Для того, чтобы гарантировать сходимость (непустоту) целевого множества М, мы определили b как , где **1** **-** единичный вектор, что гарантирует нам . Во всех экспериментах мы использовали проблемы следующих размерностей *I х J*: *80 x 100, 200 x 250, 400 x 500, 800 x 1 000, 2 000 x 2 500, 4 000 x 5 000, 8 000 x 10 000.*
3. **Число N шагов возмущений.** Это число N шагов возмущений, которые выполняются перед каждым применением оператора ***А*** (на этапе 16) влияет на производительность алгоритма LinSup. Оно влияет на баланс между количеством вычислений и ресурсами, выделенными для шагов алгоритма уменьшения целевой функции. Слишком большое значения *N* приводит к большим затратам ресурсов в Алгоритме 4 на возмущения, которые дают уменьшение целевой функции. Для того чтобы найти подходящее значение *N* для нашей работы мы создали 10 задач для каждой из следующих размерностей 80 x 100, 200 x 250, 400 x 500, 800 x 1 000, 2 000 x 2 500, и 3 проблемы размерностей 4 000 x 5 000. Мы применили Алгоритм 4 к каждой задаче с числом *N* выбранным в диапазоне от *N = 5* до *N = 100*. Все остальные параметры, кроме *N* оставались неизменными во всех опытах (в частности, ядро , релаксирующий параметр в алгоритме AMS на шаге 16 , начальная точка c соответствующими размерностями для всех задач). Правило остановки для этих экспериментов было следующим: если функция близости (7) упала ниже значения , то алгоритм останавливался. Мы записали для всех запусков относительные ошибки между значениями линейной целевой функцией , полученными с помощью LinSup после остановки алгоритма, и значениями линейной целевой функцией , полученными Simplex методом, когда MATLAB сообщил, что решение было достигнуто:

(8)

Таблица на Рисунке 1 содержит усредненные значения этих относительных ошибок по всем проблемам одного и того же размера. Эти данные приведены на Рисунке 2. На основании этих выводов мы решили использовать во всех наших последующих вычислительных экспериментах значение *N = 30* из-за наблюдения того, что, для всех размеров проблем, снижение относительной погрешности RE было слишком мало за пределами этого значения N.

1. **Параметры релаксации в алгоритме AMS**. В наших экспериментах мы установили все параметры релаксации в алгоритме поиска AMS (в виде алгоритмического оператора ***А***) и использовали на шаге 16 Алгоритма 4 . При реализации только алгоритма AMS в литературе часто показывают, что параметры релаксации оказывают значительное влияние на поведение алгоритма, см, например, [25, подразделы 11.2 и 11.5]. Тем не менее, в этой работе, при встраивании алгоритма AMS в алгоритм LinSup мы наблюдали, что параметры релаксации в алгоритме AMS имеют слабое влияние на общее поведение и, следовательно, они были установлены, на этом этапе работы, как .
2. **Обработка ограничений неотрицательности**. Как уже упоминалось выше, ограничение неотрицательности в (2) обрабатываются путем обработки вектора решений на текущей итерации AMS по всем рядам I неравенств (2) и установление всех его отрицательных компонент в нуль при сохранении остальных без изменений.

# Эксперимент 1: Линейная суперизация при нахождении максимально допустимой точки

Можно использовать любой из большого разнообразия проекционных методов для обработки линейных ограниченных неравенств, который осуществляет поиск, но мы решили выбирать в качестве основного алгоритма ***А*** знаменитый алгоритм проекционного циклического поиска Агмона-Моцкина-Шенберга (AMS) [1, 32], известный в литературе как Техника Алгебраический Реконструкции (ART) для неравенств [25, подраздел 11.2], а также [19, алгоритм 5.4.2].

Наша цель в задаче 1, экспериментально подтвердить или отвергнуть следующее утверждение:

**Утверждения 6.** Рассмотрим два запуска LinSup Алгоритма 4 для одного и того же целевого множества *M*, как и в (2), один с суперизацикй и другой без нее. «Без суперизации» означает, что шаги 5-15 в алгоритме 4 будут пропущены, а на шаге 16 просто принимается , который представляет собой простой запуск базового алгоритма ***A*** без каких-либо возмущений. Определим также, что все остальные параметры идентичны для обоих запусков, такие как начальные точки и все параметры, связанные с применением базового алгоритма ***A*** на шаге 16, а также правила остановки. При таких условиях запуск «с суперизацией» дает (останавливается на) точку , в которой значение функции меньше, чем значение в точке , в которой остановился запуск «без суперизации».

Чтобы доказать это утверждение, мы создали тестовые проблемы, как описано в пункте 6 в Разделе 3.3. На каждой такой задачи мы запустили LinSup без суперизации и с суперизацией и обнаружили, что во всех наших экспериментах Утверждение 6 верно. Мы провели все эксперименты с ядром , параметрами релаксации в алгоритме AMS на шаге 16 , и с начальной точкой соответствующего размера для всех проблем. Правила остановки для этих экспериментов: когда функция близости (7) опускается ниже значения . Число N шагов возмущений, которые выполняются перед каждым применением оператора поиска ***А*** (на шаге 16) было, как это было решено в пункте 7 в разделе 3.3, . Время выполнения в секундах, как видно в таблице на Рисунке 3, соответственно, показывает, что алгоритм с суперизацией требует больше времени, чем исходный алгоритм поиска. Все значения в этой таблице являются усредненными по 10 различным проблемам для каждой размерности, кроме последней (8 000 x 10 000), для которой мы сделали только один проход. Правые колонки в столбцах таблицы доказывают верность нашего Утверждения 6. Эти данные приведены на рисунке 4, и можно четко отметить, что эта тенденция сохраняется и укрепляется с увеличением размерности проблемы.

Сформированные данные, описанные в пункте 6 Раздела 3.3, показывают, что значение целевой функции на самом деле зависит еще и от размера *J* вектора *х*. Это наблюдается в таблице на Рисунке 3: при увеличении размерности в 10 раз, соответствующие значения целевой функции с суперизацией и без нее также увеличивается примерно в 10 раз. С этой точки зрения, относительная разница между значениями целевой функции с суперизацией и без нее формируется для различных размеров проблемы.

# Эксперимент 2: Линейная суперизация в сравнении с линейной оптимизацией Simplex методом.

Для сравнения производительности LinSup с Simplex алгоритмом линейной оптимизации мы использовали MATLAB [31]. Мы создали тестовые проблемы, как описано в пункте 6 раздела 3.3. Так как мы хотели сравнить LinSup с результатами и временем работы алгоритма Simplex, мы сначала реализовали Simplex алгоритм в MATLAB и запустили его для каждой тестовой задачи, чтобы убедиться в том, что "условие выхода" срабатывает, когда "функция сходится к решению *х*". Если тестовая задача оказалась нерешаемой с помощью алгоритма Simplex, мы отказались от нее в пользу другой тестовой задачи, для которых Simplex находит решение.

После того, как тестовая задача была решена с помощью Simplex метода, мы рассчитывали функцию близости *Pr (х)* (7), которая в целом белая небольшая. Это значение близости затем использовали в качестве правила остановки для алгоритма LinSup, который работал над той же самой проблемой. Когда алгоритм LinSup достигал этой близости, алгоритм останавливался, и решение считалось найденным.

Принудительно заставляя LinSup работать, пока он не пришел к такой же близости как решение, полученное с помощью алгоритма Simplex, мы записали и сравнили значения целевой функции и времени выполнения для обоих алгоритмов. На основе опыта, накопленного в многочисленных экспериментах и опытах, мы приняли решение зафиксировать все параметры, за исключением одного и привести здесь результаты выполнении LinSup и Simplex алгоритма MATLAB для нескольких значений этого параметра и для различных размеров задачи. Как было сказано выше, число *N* шагов возмущений, которые выполняются перед каждым применением оператора поиска ***А*** были выбраны, как *N = 30* для всех экспериментов. В качестве оператора поиска ***А*** (на этапе 16) был выбран алгоритм AMS из раздела 3.2 с фиксированными параметрами релаксации , как в пункте 8 раздела 3.3. Все остальные детали реализации были описаны в разделе 3.3. Мы исследовали влияние различных вариантов выбора ядра **α** (пункт 1, раздел 3.3) на всех запусках.

Все данные, представленные в последующих таблицах и графиках, усредняются по 10 различным и независимым друг от друга сгенерированным проблемам размера от 80 х 100 до 2000 x 2500, 5 различных и независимым друг от друга сгенерированным проблемам размера 4000 x 5000 и одной задачи размера 8000 x 10000.

В дополнение к относительной погрешности RE (8) мы записали здесь также отношение времени:

**5.1 Результаты, и что они нам говорят**

В таблице на Рисунке 5 показаны значения целевой функции для Simplex алгоритма вместе со значениями целевой функции на выходе LinSup при запуске на 3х различных значениях ядра . Также показаны значения относительных погрешностей RE (8). При больших значениях ядра его влияние уменьшается медленнее, оставляя больше места для возмущений целевой функции, что влияет на результат работы LinSup. При увеличении размерности проблемы относительная погрешность RE также возрастает.

В таблице на Рисунке 6 показано время выполнения в секундах для Simplex алгоритма и для LinSup алгоритма для 3-х различных значений ядра . Также показано отношение времени TR (9). При этом наблюдается, что LinSup работает быстре по сравнению с алгоритмом Simplex.

Рисунки 7-11 основаны на данных, содержащихся в Таблицах 5-6. Графики зависимости относительных погрешностей RE от различных размерностей проблемы для 3 различных значений ядра , основанны на данных таблицы на Рисунке 5, приводятся на Рисунке 7. Для каждого относительные ошибки увеличивается с увеличением размерности проблемы. Для всех размерностей проблемы относительные погрешности уменьшаются с ростом значения . Для всех размерностей проблемы относительная погрешность имеет наименьшее значение при наибольшем значения .

Графики зависимости отношения времени TR от размерности проблемы для LinSup по 3 различным ядра , основанные на данных из таблицы на Рисунке 6, приведены на Рисунке 8. Для каждого отношение времени уменьшается с увеличением размерности проблемы. Для всех размерностей проблемы значение соотношения времени уменьшается при уменьшении значения . Это приводит к проблеме при выбора ядра . Для лучшей (меньшей) относительной погрешности значение должно быть больше, а для лучшего соотношения (меньшего) времени оно должно быть меньше. Мы увидим некоторые компромиссы далее на следующих рисунках и таблицах.

На Рисунке 9 показаны значения целевой функции в зависимости от размерности проблемы для 3-х различных значений ядра . Наибольшее значение приводит к тому, что алгоритм LinSup производит большие шаги при снижении функции. Таким образом, это позволяет получить значения целевой функции, которые ближе всего к результатам, полученным из симплексного алгоритма. Рисунок 10 кратко показывает движение, совмещая рисунки 7 и 8. Этот рисунок наглядно показывает компромисс между уменьшением значения целевой функции и скоростью работы алгоритма LinSup.

Зависимость времени выполнения в тысячах секунд от размерности проблемы для алгоритмов Simplex и LinSup для 3 значений ядра изображены на Рисунке 11. Обратите внимание на резкое увеличение времени работы Simplex алгоритма (пунктирная линия) для задач большой размерности. LinSup отличается более умеренным ростом времени выполнения по сравнению с Simplex алгоритмом.

Результаты нашей работы показывают, что существует внутренний "конфликт" при выборе параметров, определяющих неустойчивое равновесие между усилиями, которые алгоритм LinSup прикладывает для работы алгоритма поиска, и снижением значений функции с возмущениями. Но поведение этих результатов на проблемах с большими размерностями позволяет надеяться, что при дальнейшем увеличении размерности LinSup алгоритмы получат больше возможностей и станут даже конкурентами для линейных алгоритмов минимизации. Заметим, что для задачи в последней строке таблицы на Рисунках 5 и 6 при , LinSup останавливается на значении целевой функции, которое довольно близко к полученному алгоритмом Simplex, но, при этом, алгоритм LinSup достигает этого значения всего за треть времени, которое потребовалось Simplex алгоритму.

**5.2 Прерываем работу Simplex алгоритма**

LinSup не предназначен для решения задач линейного программирования, а, как описано в Разделе 2, всего лишь позволяет найти допустимую точку с низким (не обязательно минимальным) значением линейной целевой функции. Тем не менее, с точки зрения задачи линейного программирования результат LinSup можно считать "достаточно близким решением задачи ЛП". В этой связи возникает вопрос, как это соотносится с не оптимально останавливающимся алгоритмом Simplex. Для того, чтобы получить предварительный обзор этой проблемы, мы породили задачу ЛП размерности 8000 x 10000 и запустили Simplex и LinSup алгоритмы работать по ней. LinSup остановился, когда его итерации не показали никаких существенных изменений (то есть, когда ), он был запущен с двумя различными значениями . Simplex алгоритму не разрешалось работать до нахождения оптимального решения, и он остановился через то же время, которое потребовалось алгоритму LinSup. Значения после остановки позволяют сравнить LinSup с не оптимально остановленным Simplex алгоритмом по одной и той же проблеме. Функция близости и количество вычислений значение целевой линейной функции Simplex алгоритмом считались после каждой итерации.

Эти результаты показаны на Рисунках 12 и 13. Несмотря на то, что они изучены далеко не полностью, эти результаты показывают, что если Simplex алгоритм был бы остановлен, скажем, после 5000 секунд, оба запуска LinSup дали бы более низкие значения целевой линейной функции, что показано на рисунке 12. В этот момент времени Simplex алгоритм показывал на выходе лучшие возможности, то есть более низкое значение функции близости. Тем не менее, в более поздний момент времени, например, через 20000 секунд, результат работы обоих LinSup алгоритмов будет иметь более низкое значение близости, чем Simplex, что показано на Рисунке 13, а один запуск с более высоким значением ядра будет иметь даже более низкое значение целевой линейной функции.

Такие результаты намекают на возможные преимущества LinSup для задач ЛП с большими размерностями. Если смотреть на результат работы LinSup как на "достаточно хорошее приближение решения задачи ЛП", то LinSup не только сходится к такому решению быстрее, чем может сходиться Simplex метод, работающий с высокой точностью, но и быстрее, чем не оптимально остановленный Simplex алгоритм. Следует признать, что эти и другие эксперименты, представленные здесь, требуют дальнейшего изучения, см. Раздел 6.

# 6. Выводы

Линейная суперизация (LinSup) не является, насколько мы знаем на данный момент, методом минимизации. Нахождение точки ограниченного минимума с помощью этого метода не может быть гарантировано. Однако этот метод позволяет направить технические алгоритмы поиска в направлении точек с меньшими значениями функции (не обязательно минимальным) линейной целевой функции. Подобные применение LinSup в алгоритмах поиска являются вычислительно-эффективным, они использует проекции на выпуклые замкнутые множества ограничений, особенно успешно для линейного случая. Возмущения, применяемые для уменьшения значений целевой линейной функции не требуется никаких усилий, кроме как использования ***-с*** для корректировки направления спуска. Таким образом, предыдущая работа по методологии суперизации (см ссылки, упомянутые во введении и в приложении) наряду с доказательством концепции данной экспериментальной работы, позволяют предположить, что LinSup потенциально является жизнеспособным вариантом для решения задач ЛП больших размерностей.

Наши результаты показывают, что LinSup действительно находит приемлемую допустимую точку. И, при этом, увеличение времени работы на функциях проблемных размерностей в методологии LinSup является более умеренным, чем у алгоритма Simplex. Это позволяет нам сформулировать следующую гипотезу.

***Гипотеза 7.*** *Существуют такие размеры проблем, над которыми LinSup будет работать лучше, чем линейные алгоритмы минимизации. Возможно это потребует использования сторонних методов поиска внутри LinSup, которые поддаются распараллеливанию, таких как блок-итерационных проекций (BIP) или проекций усреднения строк (SAP), указанных в разделе 3.1 выше.*

Возникает множество вопросов для дальнейших исследований, основанных на текущей работе. Вот список некоторых потенциально интересных направлений:

(1) Расширить вычислительные эксперименты для больших размерностей проблем и для по-разному порожденных проблем.

(2) Протестировать LinSup на более широком классе тестовых задач, чем те, которые используются в данной работе, например, на задачах линейного программирования из NETLIB (http://www.netlib.org/) или других хранилищ.

(3) Исследовать LinSup с дополнительными проекционными методами поиска, которые поддаются распараллеливанию, такими как методы блок-итерационной проекции (BIP) или проекций усреднения строк (SAP).

(4) Исследовать влияние различных параметров на поведение LinSup, повторяя эксперименты с различными значениями: числа шагов возмущений N, которые выполняются перед каждым применением оператора осуществляющего поиск ищущий ***А***, параметров релаксации для встроенного базовом алгоритме поиска, ядра , с помощью которого генерируются размеры шагов .

(5) Углубленный математический анализ алгоритма LinSup.

(6) Повторить приведенные выше сравнения для дополнительных алгоритмов линейной оптимизации, такие как "interior-point" или "active-set" в пакете MATLAB или других.

(7) Исследовать противоречивый случай, в котором целевое множество М (2) пусто и заменяется, например, множеством ближайших точек для всех ограничений в соответствии с некоторой функцией близости. Алгоритмы линейного программирования могут не работать в таких условия, но LinSup вполне может дать полезный результат.

(8) Изучить работу LinSup для разреженных линейных ограничений, для которых некоторые проекционные методы уже доказали свою эффективность в качестве алгоритмов поиска.

# 7. Приложение: развитие алгоритма суперизация

Алгоритмическая структура версии базового алгоритма с суперизацией претерпела изменения и модификации в течение последних нескольких лет с момента его создания. Все эти изменения сохраняют основную базовую методологию алгоритма, и будет полезно бегло ознакомиться с ними здесь. Суперизация впервые появляется в работе [5], хотя там и не использовались слова «суперизация» и «возмущение устойчивости». Эта работа основана на некоторых более ранних теоретических работах [6, 7]. Псевдокод на правой боковой колонке страницы 543 в [5] представляет собой первый алгоритм суперизация. Размеры шагов там (строка 9) просто уменьшаются вдвое, и присутствует всего одна стадия восстановления функции (строка 3) для каждого цикла алгоритма поиска ***P*** (в строке 6). Там присутствует два этапа процесса принятия решений (в строках 4 и 7). Этот алгоритм был использован в дипломной работе Скотта Пенфолдом [35] (см также [36]) и в работе [37]. Алгоритмы 2 и 3 (TVS1- DROP и TVS2-DROP, соответственно) в [37] относятся к различным вариациям встраивания алгоритма поиска «DROP» [14], и не отличаются от методов суперизация. В той же работе [37] дорогостоящий шаг принятия решений (строка 12) в обоих алгоритмах был удален без побочных эффектов, что позволило существенно сэкономить время.

В " версия алгоритма ***P*** с суперихацией" на странице 6 в [13] оба этапа принятия решения из более ранних версий по-прежнему появляются, но на этот раз в одной строке (строка ХIII). Однако, в последующей работе [27], дорогостоящий этап принятия решений (12 строка в [37]) больше не появляется, и, кроме того, отрицательный субградиент (отрицательный градиент, если функция дифференцируема) заменяется на любое направление, в котором функция «не увеличивается», что и есть суперизация. Это последнее изменение не требуется суперизации с полной вариацией (ТВ), но быть использовано для других функций . Еще один новый компонент в [27], который является очень полезным - это способность производить во внутреннем цикле (от строки VII до строки ХVII) *N* шагов снижения функции (*N* задается пользователем) на каждой итерации алгоритма поиска, который обозначается там как (в строке ХVIII).

Интересное сравнительное исследование появляется в [15]. Алгоритм там называется "Версия алгоритма с суперизацией" и появляется на страницах 737-738, где реализуемый алгоритм поиска обозначается (на шаге 18).

В работе [18, алгоритм 4.1] был сделан еще один шаг вперед. (1) число *N* (из [27]) стало отличаться от одной итерации к другой, таким образом, теперь *N* заменяется на , где k *-* индекс итерации. (2) отброшена вторая проверка принятия решений, которая была на строке 14 в [15], даже в более ранних алгоритмах суперизации. Замена *N* на *Nk* математически верно, но, насколько нам известно, до сих пор не подтверждено кем-либо экспериментально.

Две важных недавних работы по реализации алгоритма суперизации появляются в [29] и [38]. Одно из важных дополнительных изменения, которые мы использовали в нашей работе (см. пункт 2 в разделе 3.3 выше), - это управление размерами шагов возмущений в Алгоритме 4 с помощью специальной стратегии обновления индекса *l* на каждой итерации.

**Выражение признательности**. Мы с благодарностью отмечаем некоторые предварительные обсуждения с Ran Davidi и John Chinneck. Благодарим двух анонимных рецензентов за их конструктивные замечания, которые помогли нам улучшить эту работу. Программы в MATLAB искусно выполнены с большим энтузиазмом и преданностью Yehuda Zur, за ему спасибо. Эта работа была поддержана Исследовательским грантом №2013003 от e United States-Israel Binational Science Foundation (BSF) и решением №1P20183640-01A1 от National Institutes of Health (NIH)..

**Комментарий**. Окончательный вариант данной работы доступен по адрессу: http://math.haifa.ac.il/yair/censorrecent-pubs.html. Другие документы по суперизация приведены ниже имеют свои тезисы и коды DOI, они размещены на: http://math.haifa.ac.il/yair/bibсуперизация-censor.html.

# Список использованных источников

[1] S. Agmon, The relaxation method for linear inequalities, Canadian Journal of Mathematics 6 (1954), 382-392.

[2] J.-B. Baillon, P.L. Combettes and R. Cominetti, There is no variational characterization of the cycles in the method of periodic projections, Journal of Functional Analysis 262 (2012), 400-408.

[3] H.H. Bauschke and J.M. Borwein, On projection algorithms for solving convex feasibility problems, SIAM Review 38 (1996), 367-426.

[4] H.H. Bauschke and V.R. Koch, Projection methods: Swiss army knives for solving feasibility and best approximation problems with half-spaces, Contemporary Mathematics 636 (2015), 1-40.

[5] D. Butnariu, R. Davidi, G.T. Herman, and I.G. Kazantsev, Stable convergence behavior under summable perturbations of a class of projection methods for convex feasibility and optimization problems, IEEE Journal of Selected Topics in Signal Processing 1 (2007), 540-547.

[6] D. Butnariu, S. Reich and A.J. Zaslavski, Convergence to Fixed points of inexact orbits of Bregman-monotone and of nonexpansive operators in Banach spaces, in: H.F. Nathansky, B.G. de Buen, K. Goebel, W.A. Kirk, and B. Sims, Fixed Point Theory and its Applications, (Conference Proceedings, Guanajuato, Mexico, 2005), Yokahama Publishers, Yokahama, Japan, pp. 11-32, 2006.

[7] D. Butnariu, S. Reich and A.J. Zaslavski, Stable convergence theorems for infinite products and powers of nonexpansive mappings, Numerical Functional Analysis and Optimization 29 (2008), 304-323.

[8] A. Cegielski, Iterative Methods for Fixed Point Problems in Hilbert Spaces, Lecture Notes in mathematics 2057, Springer-Verlag, Berlin, Heidelberg, Germany, 2012.

[9] Y. Censor, Weak and strong суперизация: Between feasibility-seeking and minimization, Analele Stiintifice ale Universitatii Ovidius Constanta-Seria Matematica 23 (2015), 41-54.

[10] Y. Censor, “Суперизация and Perturbation Resilience of Algorithms: A Bibliography compiled and continuously updated by Yair Censor”, an Internet page: [http://math.haifa.ac.il/yair/bib-суперизацияcensor.html](http://math.haifa.ac.il/yair/bib-superiorizationcensor.html).

[11] Y. Censor and A. Cegielski, Projection methods: an annotated bibliography of books and reviews, Optimization 64 (2015), 2343-2358. DOI:10.1080/02331934.2014.957701.

[12] Y. Censor, W. Chen, P.L. Combettes, R. Davidi and G.T. Herman, On the effectiveness of projection methods for convex feasibility problems with linear inequality constraints, Computational Optimization and Applications 51 (2012), 1065-1088.

[13] Y. Censor, R. Davidi and G.T. Herman, Perturbation resilience and суперизация of iterative algorithms, Inverse Problems 26 (2010), 065008 (12pp).

[14] Y. Censor, T. Elfving, G.T. Herman and T. Nikazad, On diagonallyrelaxed orthogonal projection methods, SIAM Journal on Scientific Computing 30 (2008), 473-504.

[15] Y. Censor, R. Davidi, G.T. Herman, R.W. Schulte and L. Tetruashvili, Projected subgradient minimization versus суперизация, Journal of Optimization Theory and Applications 160 (2014), 730-747.

[16] Y. Censor and A.J. Zaslavski, Convergence and perturbation resilience of dynamic string-averaging projection methods, Computational Optimization and Applications 54 (2013), 65-76.

[17] Y. Censor and A.J. Zaslavski, String-averaging projected subgradient methods for constrained minimization, Optimization Methods & Software 29 (2014), 658-670.

[18] Y. Censor and A.J. Zaslavski, Strict Fejer monotonicity by суперизация of feasibility-seeking projection methods, Journal of Optimization Theory and Applications 165 (2015), 172-187.

[19] Y. Censor and S.A. Zenios, Parallel Optimization: Theory, Algorithms, and Applications, Oxford University Press, New York, NY, USA, 1997.

[20] J.W. Chinneck, Feasibility and Infeasibility in Optimization, Springer Science+Business Media, LLC, New York, NY, USA, 2008.

[21] P.L. Combettes, On the numerical robustness of the parallel projection method in signal synthesis, IEEE Signal Processing Letters 8 (2001), 45-47.

[22] R. Davidi, G.T. Herman, and Y. Censor, Perturbation-resilient blockiterative projection methods with application to image reconstruction from projections, International Transactions in Operational Research 16 (2009), 505-524.

[23] E. GarduÒo, and G.T. Herman, Суперизация of the ML-EM algorithm, IEEE Transactions on Nuclear Science 61 (2014), 162-172.

[24] D. Gordon and R. Gordon, Component-averaged row projections: A robust, block-parallel scheme for sparse linear systems, SIAM Journal on Scientific Computing 27 (2005), 1092-1117.

[25] G.T. Herman, Fundamentals of Computerized Tomography: Image Reconstruction from Projections, Springer-Verlag, London, UK, 2nd Edition, 2009.

[26] G.T. Herman, Суперизация for image analysis, in: Combinatorial Image Analysis, Lecture Notes in Computer Science Vol. 8466, Springer, 2014, pp. 1-7.

[27] G.T. Herman, E. Garduno, R. Davidi and Y. Censor, Суперизация: An optimization heuristic for medical physics, Medical Physics 39 (2012), 5532-5546.

[28] W. Jin, Y. Censor and M. Jiang, A heuristic суперизация-like approach to bioluminescence, International Federation for Medical and Biological Engineering (IFMBE) Proceedings 39 (2013), 1026-1029.

[29] O. Langthaler, Incorporation of the Суперизация Methodology into Biomedical Imaging Software, Marshall Plan Scholarship Report, Salzburg University of Applied Sciences, Salzburg, Austria, and The Graduate Center of the City University of New York, NY, USA, September 2014, (76 pages). http://www.marshallplan.at/images/papers\_scholarship/2014/Salzburg \_University\_of\_Applied\_Sciences\_LangthalerOliver\_2014.pdf.

[30] S. Luo and T. Zhou, Суперизация of EM algorithm and its application in single-photon emission computed tomography (SPECT), Inverse Problems and Imaging 8 (2014), 223-246.

[31] MATLAB R , A high-level language and interactive environment system by MathWorks, <http://www.mathworks.com/products/matlab/>.

[32] T.S. Motzkin and I.J. Schoenberg, The relaxation method for linear inequalities, Canadian Journal of Mathematics 6 (1954), 393ñ404.

[33] T. Nikazad, R. Davidi and G.T. Herman, Accelerated perturbationresilient block-iterative projection methods with application to image reconstruction, Inverse Problems 28 (2012), 035005 (19pp).

[34] E.A. Nurminski, Single-projection procedure for linear optimization, Journal of Global Optimization, accepted for publication (2015). DOI: 10.1007/s10898-015-0337-9.

[35] S.N. Penfold, Image Reconstruction and Monte Carlo Simulations in the Development of Proton Computed Tomography for Applications in Proton Radiation Therapy, PhD Thesis, University of Wollongong, Wollongong NSW 2522, Australia, 2010.

[36] S.N. Penfold, A Prototype Proton Computed Tomography System: Image Reconstruction and Monte Carlo Simulations, Lambert Academic Publishing (LAP), Germany, 2012.

[37] S.N. Penfold, R.W. Schulte, Y. Censor and A.B. Rosenfeld, Total variation суперизация schemes in proton computed tomography image reconstruction, Medical Physics 37 (2010), 5887-5895.

[38] B. Prommegger, Verification and Evaluation of Superiorized Algorithms Used in Biomedical Imaging: Comparison of Iterative Algorithms With and Without Суперизация for Image Reconstruction from Projections, Marshall Plan Scholarship Report, Salzburg University of Applied Sciences, Salzburg, Austria, and The Graduate Center of the City University of New York, NY, USA, October 2014, (84 pages). http://www.marshallplan.at/images/papers\_scholarship/2014/Salzburg \_University\_of\_Applied\_Sciences\_PrommeggerBernhard\_2014.pdf.

[39] M.J. Schrapp and G.T. Herman, Data fusion in X-ray computed tomography using a суперизация approach, Review of Scientific Instruments 85 (2014), 053701 (9pp).

[40] H.A. Simon, Rational choice and the structure of the environment, Psychological Review 63 (1956), 129-138.

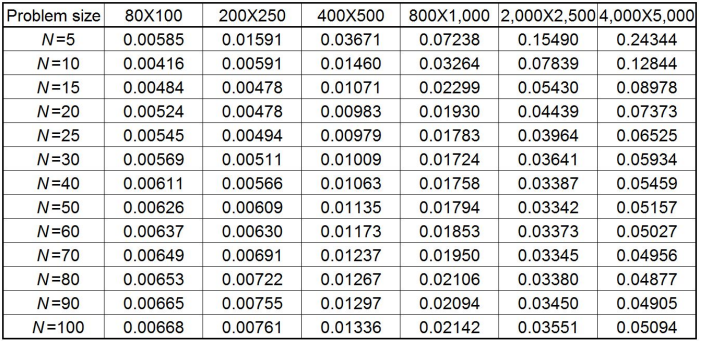


Рисунок 1. Относительные погрешности между значениями целевой линейной функции , полученным алгоритмом LinSup, когда он был остановлен, и значениями линейной целевой функции, полученными методом Simplex, когда MATLAB сообщили, что решение было достигнуто, для различных значений числа N на шаге 8 Алгоритма 4. Числа в таблице являются средними значениями для нескольких проблем разных размерностей (см. текст работы).

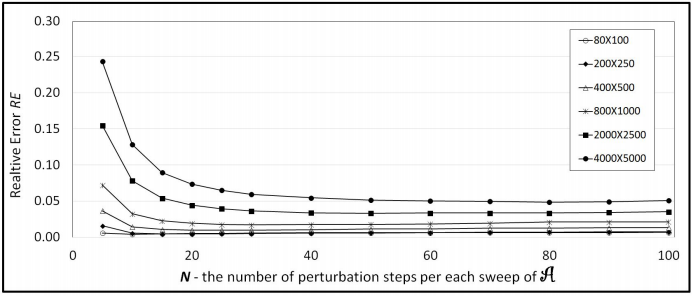


Рисунок 2: Графики данных, включенных в таблицу на рисунке 1. На основании этих графиков мы решили использовать *N = 30* во всех наших последующих вычислительных экспериментах.

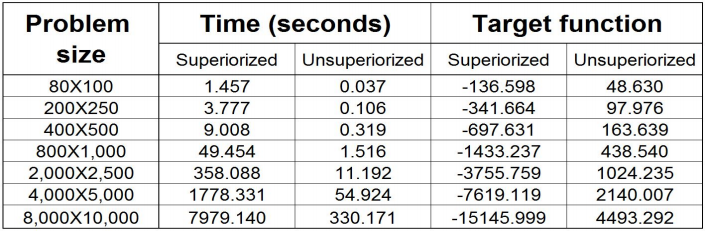


Рисунок 3: Все значения в этой таблице являются усредненными по более, чем 10 различным задачам для каждой размерности, кроме последней (8000 х 10000), для которой мы сделали только один проход, как исключение. Время выполнения в секундах, соответственно, показывает, что суперизация алгоритм требует больше времени, чем простой алгоритм поиска. Правые части столбцов в таблице подтверждают истинность гипотезы в Разделе 6 для экспериментов, проводимых нами.

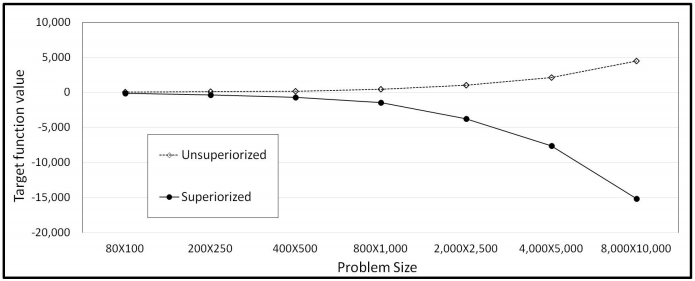


Рисунок 4: На этом графике изображены данные таблицы 3. График показывает, что разрыв между значениями целевой функции с суперизация и без нее монотонно возрастает с увеличением размерности проблемы.

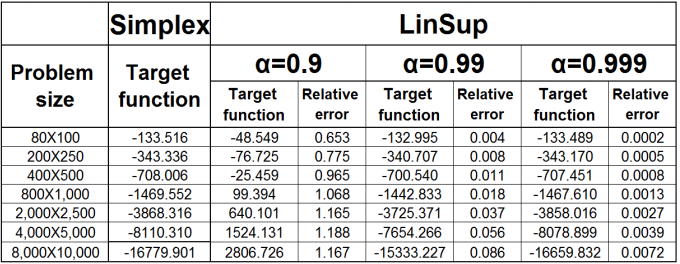


Рисунок 5: В этой таблице показаны значения целевая функция для Simplex алгоритма и значения целевой функции на выходе LinSup алгоритма при запуске на 3 различных значениях ядра . Также показаны относительные погрешности RE (8).

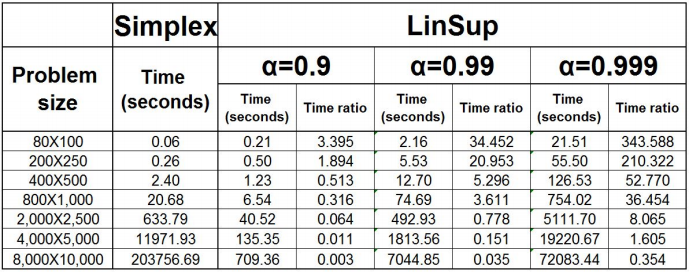


Рисунок 6: Эта таблица показывает время выполнения в секундах для Simplex алгоритма и для LinSup по 3-х различным значениям ядра . Также показано отношения времени TR (9).

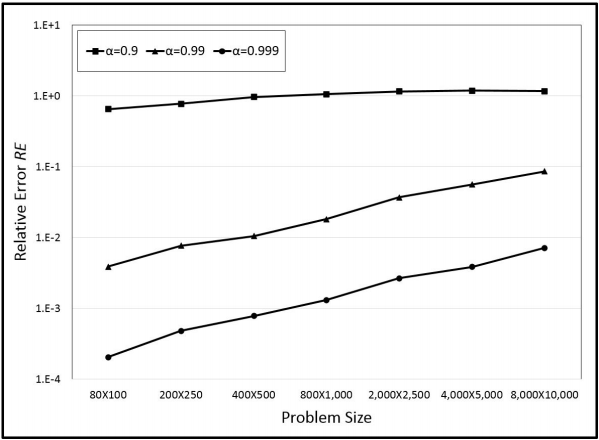


Рисунок 7: Графики относительных погрешностей RE, на логарифмической шкале, в зависимости от размерностей проблемы для LinSup по 3 различным значениям ядра , основанные на данных из таблицы на Рисунке 5. Для каждого относительные ошибки увеличивается с увеличением размерностей. Для всех размерностей проблемы относительные погрешности уменьшаются с ростом значения .

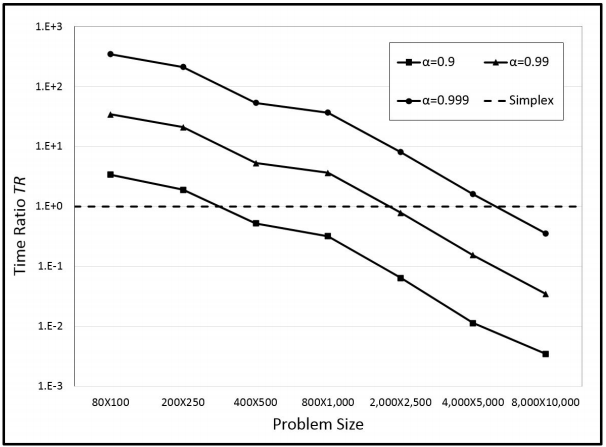


Рисунок 8: Графики отношения времени TR, на логарифмической шкале, в зависимости от размерности проблемы для LinSup по 3 различным значением ядра , основанные на данных из таблицы на Рисунке 6. Для каждого отношение время уменьшается с увеличением размерностей проблемы. Для всех размерностей проблемы отношение времени уменьшается при уменьшении значения .

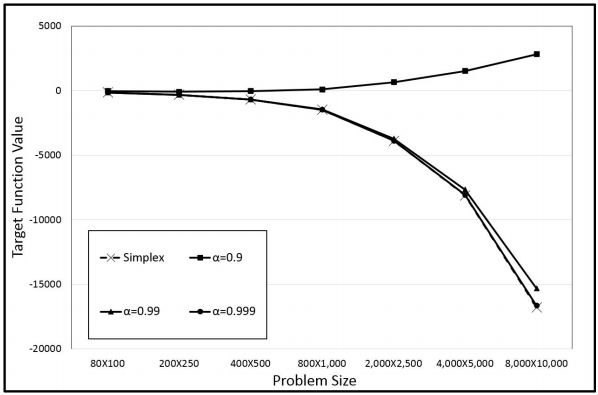


Рисунок 9: Зависимость значений целевой функции от размерности задачи по 3 различным значениям ядра . При самом большом значении движение алгоритма LinSup по уменьшению целевой функции достигает наилучших результатов. Эти значения близки к результатам, полученным из алгоритма Simplex.

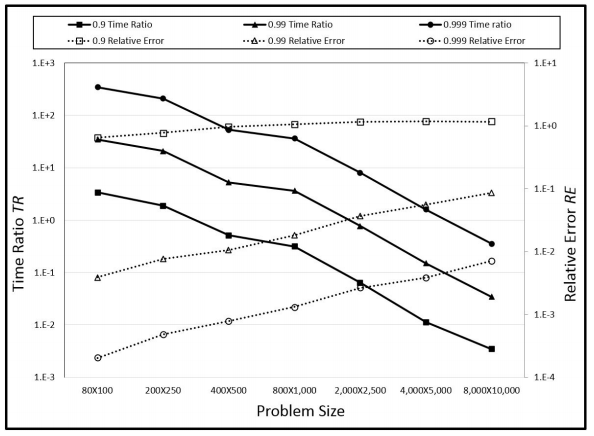


Рисунок 10: Наложение графиков 7 и 8. Этот рисунок наглядно показывает компромисс между уменьшением значения целевой функции и скоростью работы алгоритма LinSup.

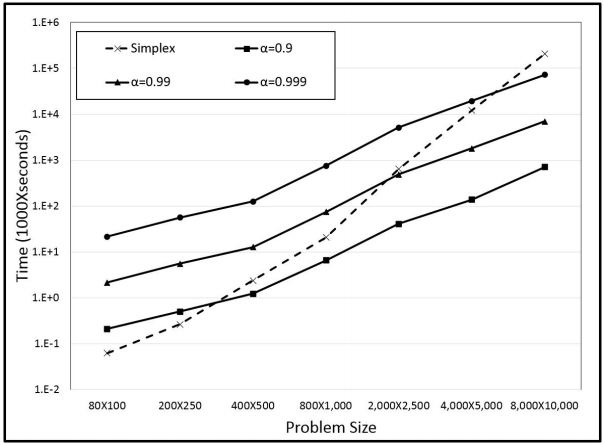


Рисунок 11: Время выполнения в тысячах секунд, на логарифмической шкале, в зависимости от размерности проблемы для алгоритма Simplex и для LinSup по 3 значениям ядра . Обратите внимание на резкое увеличение времени работы Simplex алгоритма (пунктирная линия).

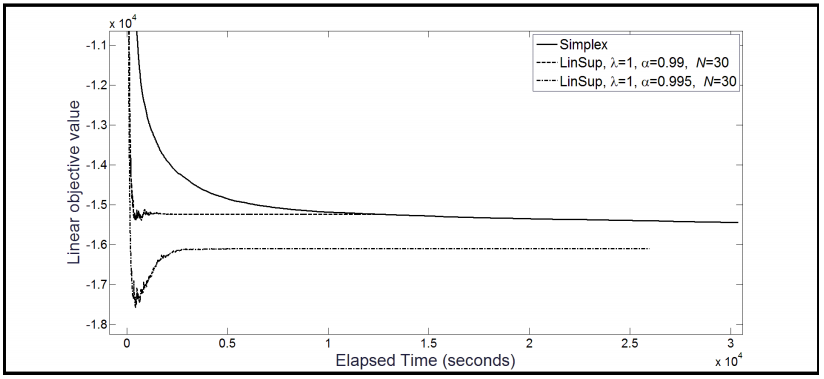


Рисунок 12: Если бы Simplex алгоритм останавливался не оптимально, например, после 5000 секунд, то оба запуска LinSup алгоритма давали бы более низкие значения линейной целевой функции.

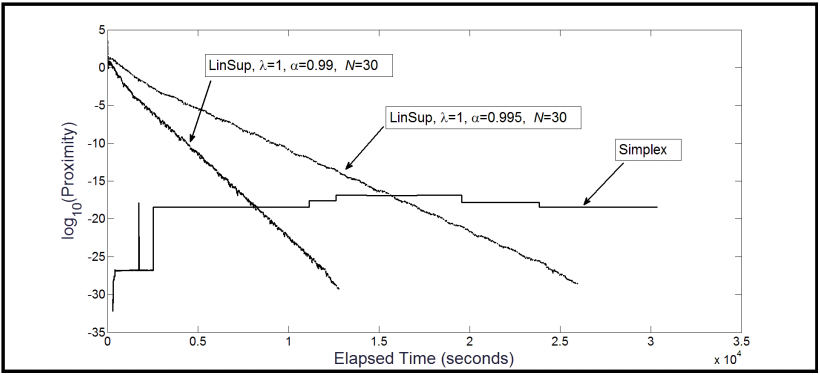


Рисунок 13: После 20000 секунд, оба запуска LinSup алгоритма будут иметь более низкие значения функции близости, чем Simplex алгоритм, а один запуск с наибольшим значением ядра будет даже иметь более низкое значение линейной целевой функции.

# Приложение 3. Оригинал статьи на английском языке