# Введение

В данной работе мы предлагаем линейный метод superiorization в качестве инструмента для обработки задач линейного программирования. Метод линейной superiorization 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 задаче поиска минимума функции в задаче линейного программирования, но и в задачах с большими размерами, он является хорошей альтернативой симплекс-метода линейного программирования, с которым мы и будем сравнивать его здесь.

Эта работа опирается на более ранние теоретические работы о superiorization, которые включены в список используемых источников, в частности, [18]. В качестве используемых инструментов используются только экспериментальные расчеты. Несмотря на это, мы не обсуждаем вычислительные проблемы сами по себе, а используем вычисления в качестве инструмента для практической проверки, в стиле "доказательство концепции".

**Что такое superiorization?** Многие ограниченные методы оптимизации основаны на методах безусловной оптимизации, которые приспособлены для борьбы с ограничениями. Так, например, градиентные методы, в которых безусловная минимизация внутреннего шага "управляет" процессом и проекцией в допустимом множестве и выполняется после каждого шага минимизации. Эта проекция на ограничения внутри метода приводит к необходимости решения нетривиальной задачи оптимизации и необходимость ее решения в каждой итерации препятствует работе методов градиента и ограничивает их эффективность только на допустимых множествах. Барьерные или штрафные методы аналогичным образом основаны на безусловной оптимизации вместе 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 superiorization можно рассматривать как противоположную точку зрения. Вместо того, чтобы адаптировать алгоритмы минимизации без ограничений для обработки ограничений, адаптируются сами алгоритмы 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: (i) **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. (ii) **Прикладной**: в некоторых существенных реальных приложениях выбор целевой функции экзогенными для моделирования и сбора данных приводит к возникновению ограничений. В таких ситуациях, подобный выбор целевой функции часто приводит к тому, что нет никакой необходимости, ни оправдания, чтобы искать точку минимума в этих ограничениях. Для получения «правильных результатов», часто бывает достаточно найти допустимую точку, которая уменьшает значение целевой функции (обязательно точку минимума). В некоторых научных приложениях, операции с целевыми функциями являются затратными и играют центральную роль в модели, но в других приведенные выше рассуждения могут по-прежнему применяться.

**Текущее исследование.** Текущую работу по superiorization можно оценить из материалов на странице в сети Интернет [10]. В частности, [26] и [9] отзывы об интересах. Недавние исследования включает в себя множество отчетов, начиная от новых приложений в промышленной рентгеновской компьютерной томографии [39] и новые математические результаты на основе superiorization, таких как обоснование строгой монотонности фейеровских методов с использованием superiorization проекционных методов [18]. Недавнее подробное описание предыдущей работы, связанной с superiorization можно найти в [15, раздел 3].

**Линейная superiorization.** Линейная superiorization (далее сокращенно: LinSup) рассматривает проблемы линейного программирования (ЛП), в которых ограничения, а также целевая функция являются линейными. Два вопроса, которые мы хотим исследовать экспериментально: (i) Может ли LinSup обеспечить допустимую точку, в которой значение целевой функции ниже, чем в точке, полученной тем же алгоритмом но без superiorization, но при прочих равных условиях? и (ii) Какова «стоимость» LinSup алгоритма по сравнению с симплекс-методами для решения задач ЛП? На основании наших вычислительных экспериментов, представленных здесь, ответы на эти два вопроса: "да" и "очень хорошо", соответственно.

Интересным и перспективным аспектом современных экспериментов является зависимость результатов от размеров тестовой задачи. Мы обнаружили, что преимущества LinSup становятся монотонно более выраженным при увеличении размеров проблемы. Мы рассмотрели задачи с 8000 линейных неравенств и векторов по 10000 компонентов, и увидели тенденцию, которая также сохраняется за пределами этих размеров проблем. Таким образом, LinSup вполне может стать полезным инструментом для вычислений проблема 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 покажем численное поведение LinSup в сравнении с классическим Simplex алгоритмом линейной оптимизации. Общие рамки superiorization показываются в разделе 2, а LinSup представлена в разделе 3. Наши экспериментальные результаты были произведены с промощью 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 технические изменения и модификации в superiorized версии базового алгоритма, опубликованные в литературе за последние несколько лет с момента создания.

# Методология superiorization

Рассмотрим пару (*М,* ***А***), где *М*, называемая целевым набором, является заданным подмножеством заданного подмножества *Q* J-мерного евклидового пространства *M ⊂ Q ⊆ RJ*. Пусть ***А*** *: Q → Q* будет неким алгоритмический оператор, 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*) ка superiorization пару. Пусть ᵠ *: Q ⊆ RJ→ R* заданная действительная функция, называется базовой функцией.

Методология superiorization предназначена для ограниченных задач уменьшения функции следующего вида.

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

Методология superiorization подходит к этой проблеме путем исследования возмущений основного алгоритма, а затем с помощью предупреждения таких возмущений, "заставляет" возмущенный алгоритм, полученный из базового, делать в дополнение к своей первоначальной задаче уменьшение шагов функции. Таким образом, этот возмущенный алгоритм называется "superiorized версия базового алгоритма".

Если базовый алгоритм является вычислительно эффективным и полезным при нахождении точки М, и если это возмущение упругое и может быть просто и недорого вычислено. То преимущество этого способа состоит в том, что, по расчетной стоимости базового алгоритма, мы можем решить задачу условной редукции функции, регулируя итерации в соответствии с возмущениями по уменьшению целевой функции. Методология superiorization автоматически генерирует superiorized версию базового алгоритма. Вектор х \*, полученные путем применения superiorized версии базового алгоритма, не обязательно должны минимизировать ᵠ 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 superiorization "найти точку *М*" подразумевается, как получить бесконечную последовательность , которая сходится к точке х\*∈ *М*, при этом *М* должно быть непустым. При сильном superiorization "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* считать не нужно. Эти понятия были определены в работе [9].

Два важных частных случаев пар superiorization (*М,* ***А***) в указанных выше рамках приходят на ум, хотя и другие случаи также возможны.

**Случай 2**. Целевое множество М является множество решений выпуклой задачи следующего вида: Найти вектор х\* ∈ 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 другой задачи минимизации с ограничениями: минимизировать целевую функцию f в допустимой области , таким образом . В этом случае алгоритмический оператор и базовый алгоритм (1) может быть использованы к любому из широкого разнообразия ограниченных алгоритмов минимизации распространенных в литературе.

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

# Линейная superiorization

* 1. **Проблема и алгоритм**

Определим целевое множество *M:*

(2)

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

Для базового алгоритма мы выбираем ищущие проекционные методы. 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 они демонстрируют хорошие показатели сходимости и/или хорошие начальные модели поведения. Этот класс алгоритмов стал свидетелем большого прогресса в последние годы, и его алгоритмы были с успехом применены для многих научных, технологических и математических задач. Смотрите, например, в обзор [3] 1996 года, за последнее время книги и обзоры [11] и ссылки, книги [8] или [12].

Важное замечание в этом месте. Выпуклая проблема, упомянутая в **Случае 2**, может быть переведена в безусловную минимизацию некоторой функции близости, которая измеряет изменение точек алгоритма. Например, с помощью взвешенной суммы квадратов евклидовых расстояний до множеств CFP как функция близости и применение наискорейшего спуска к нему приводит метод одновременной проекции для CFP типа CIMMINO. Тем не менее, нет никакой функции близости, что дало бы метод последовательных проекций на CFP типа Качмажа см [2]. Таким образом, исследование осуществимости алгоритмов поиска для CFP разработан независимо от методов минимизации, смотрите ссылки, упомянутые выше. На протяжении многих лет исследователи пытались использовать методы прогнозирования для выпуклой задачи алгоритмов Л.П. более чем одним способом, например, см. книгу Chinneckís [20]. Мини-обзор отношений между линейным программированием алгоритмами 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 superiorization будет

(3)

где – скалярное произведение вектора x и данного вектора .

Следуя общим принципам методологии superiorization, которые были представлены для общих целевых функций ᵠ в предыдущих публикациях, смотрите, например, недавние обзоры [26] и [9], приведем следующий линейный алгоритм superiorization. Вход алгоритма состоит из исходных данных 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 размер шага , 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 Superiorization (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 Эгмон-Моцкин-Шенберга как базовый алгоритм**

Мы использовали проекционный метод Эгмон-Моцкин-Шенберга (AMS) [1,32], см. также [19, Алгоритм 5.4.2], в качестве базового алгоритма для уменьшающей функции ***A***  на шаге 16 Алгоритма 4. Обозначим полупространства представленные отдельным строками в (2) Hi,

, (4)

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

(5)

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

**Инициализация:**  задается произвольно.

**Итерация:** Следующая итерация вектора **x** расчитывается 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 зациклено на {1, 2…, I}.

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

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

* 1. **Детали реализации и объяснения**

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

1. **Размер шага возмущений**. Размеры шагов в Fлгоритме 4 должны быть такими, что таким образом, 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 размера шагов на шаге 9 Алгоритма 4. Число называется ядром последовательности .
2. **Контроль уменьшения размеров шагов при уменьшении 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 определяет 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* принимая его значение, как это было в конце последней итерации *N* и начала новой итерации с этого значения, 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 *"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, в алгоритме Simplex нет такой тонкой балансировки, хотя подобное было на ранних стадиях разработки (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 появилась в процессе принятие решения во многих предыдущих формулировках superiorized версий базового алгоритма, смотрите, например, шаг (ХIV) в "Superiorized версий алгоритма **Р**" в [27, стр 5537]. Так как мы имеем возможность доказать 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 уровеня 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 половиной квадратных 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. Первые были случайным образом выбраны в интервале [0; 1] с помощью алгоритма или пользователем, близость этого выбора к множеству M вычисляется функцией (7). Если близость равена нулю, то мы выбираем новый и повторяется это приращения в 10 раз до тех пор, пока мы не найдем точку с ненулевой близости, 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 superiorization 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 размеров *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* 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 одно 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 superiorization 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 Агмон-Моцкин-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 superiorization и другой без нее. «Без superiorization» 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 параметров, связанные с применением базового алгоритма ***A*** на шаге 16, а также правила остановки. При таких условиях запуск "с superiorization" дает (останавливается) точку , в которой значение имеет меньшее значение, чем в точке , в которой остановился запуск "без superiorization".

Чтобы доказать это утверждение, мы создали тестовые проблемы, как описано в пункте 6 в подразделе 3.3. На каждой такой задачи мы провели LinSup без superiorization и с superiorization 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 показано в таблице на Рисунке 3, соответственно, показывают, что superiorization алгоритм 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 который мы сделали только один проход. Боковые столбцы правой таблицы показывают верность нашего Утверждения 6. Эти данные приведены на рисунке 4, и можно четко отметить, что эта тенденция сохраняется и укрепляется с увеличением размером проблемы.

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

# Эксперимент 2: Линейная superiorization по сравнению с линейной оптимизацией Simplex методом.

Для сравнения производительности LinSup с этим алгоритмом линейной оптимизации мы использовали MATLAB [31] и выбрал алгоритм "Simplex" от "linprog". 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 Симплекс 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) для этого, 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 на этой итерации, 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 проблем для размерностей от 80 х 100 до 2000 x 2500, 5 различных и независимым друг от друга сгенерированным проблемам размером 4000 x 5000 и одной задачи размера 8000 x 10000.

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

# 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 3 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 Позволяем Simplex алгоритму останавливаться заранее

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 бы 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 практически через то же время, которое потребовалось алгоритму LinSup. Значения после остановки позволяют сравнить LinSup с неоптимально остановленным Simplex по одной и той же проблеме. Функция близости и количество вычислений значение целевой линейной функции Simplex алгоритмом считались после каждой итерации, потому что это не являются внутренней частью Simplex алгоритма.

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 неоптимально, 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 приближенное решение 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 дальнейшей работы, см. Раздел 6.

# 6. Выводы

Линейная superiorization (LinSup) не является, насколько мы знаем на данный момент, методом минимизации. Нахождение точки ограниченного минимума с помощью этого метода не может быть гарантирована. 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 является вычислительно-эффективным, оно 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 superiorization (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 разности проблем, над которыми LinSup будет работать лучше, чем линейные алгоритмы минимизации. Может быть, что это потребует использования сторонник 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, задачи 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 superiorization

Алгоритмическая структура superiorized версии базового алгоритма 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. Superiorization появляется в работе [5], хотя там и не использовались слова «superiorization» 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 superiorization. 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 superiorization. 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 "Superiorized версии алгоритма ***P***" 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 superiorized. Это последнее изменение не требуется superiorization с полной вариации (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 "Superiorized версия базового алгоритма" 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 superiorization. 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 superiorization 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 superiorization приведены ниже имеют свои тезисы и коды DOI, они размещены на: http://math.haifa.ac.il/yair/bibsuperiorization-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 superiorization: Between feasibility-seeking and minimization, Analele Stiintifice ale Universitatii Ovidius Constanta-Seria Matematica 23 (2015), 41-54.

[10] Y. Censor, “Superiorization and Perturbation Resilience of Algorithms: A Bibliography compiled and continuously updated by Yair Censor”, an Internet page: <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 superiorization 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 superiorization, 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 superiorization 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, Superiorization 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, Superiorization 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, Superiorization: An optimization heuristic for medical physics, Medical Physics 39 (2012), 5532-5546.

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

[29] O. Langthaler, Incorporation of the Superiorization 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, Superiorization 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 superiorization 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 Superiorization 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 superiorization 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.

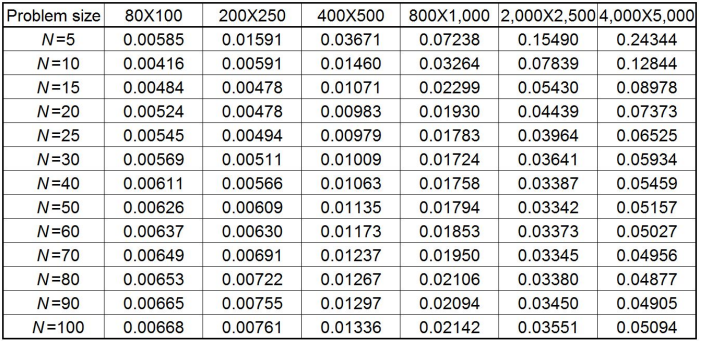


Рисунок 1. Относительные погрешности между значениями целевой линейной функции , полученным алгоритмом LinSup, когда он был остановлен, и значениями линейной целевой функции, полученными методом Simplex, когда MATLAB сообщили, что решение было достигнуто, для различных значений числа N на шаге 8 Алгоритма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

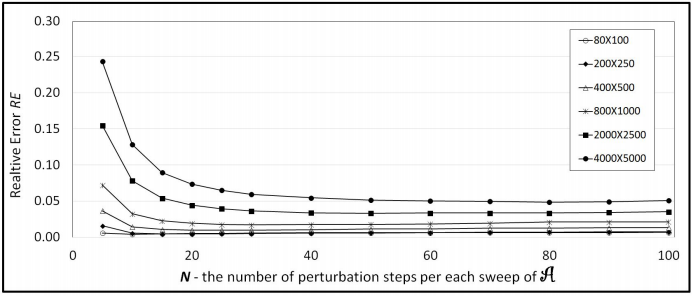


Рисунок 2: Графики данных, включенных в таблицу на рисунке 1. На основании этих графиков мы решили использовать *N = 30* во всех наших последующих вычислительных экспериментах.

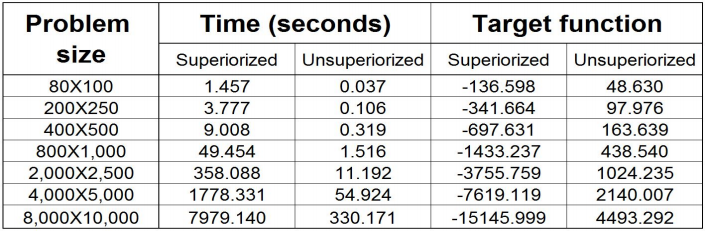


Рисунок 3: Все значения в этой таблице являются усредненными по более, чем 10 различным задачам для каждой размерности, кроме последней (8000 х 10000), для которой мы сделали только один проход, как исключение. Время выполнения в секундах, соответственно, показывает, что superiorization алгоритм требует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

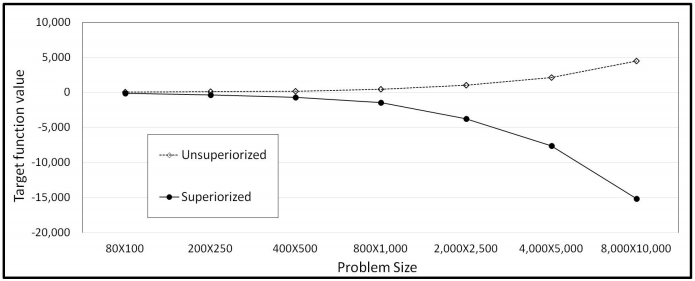


Рисунок 4: На этом графике изображены данные таблицы 3. График показывает, что разрыв между значениями целевой функции с superiorization и без нее монотонно возрастает с увеличением размерности проблемы.

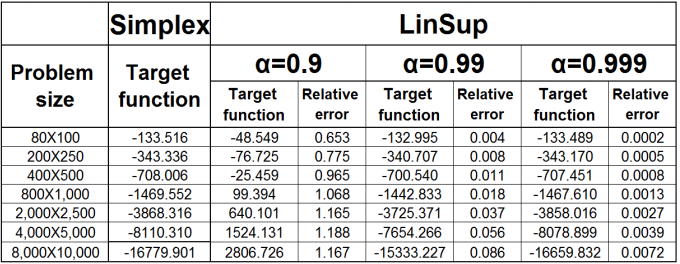


Рисунок 5: В этой таблице показаны значения целевая функция для Simplex алгоритма и значения целевой функции на выходе LinSup алгоритма при запуске на 3 различных значениях ядра . Также показаны относительные погрешности RE (8).

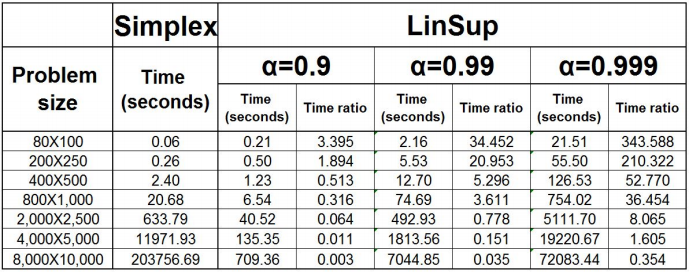


Рисунок 6: Эта таблица показывает время выполнения в секундах для Simplex алгоритма и для LinSup по 3-х различным значениям ядра . Также показано отношения времени TR (9).

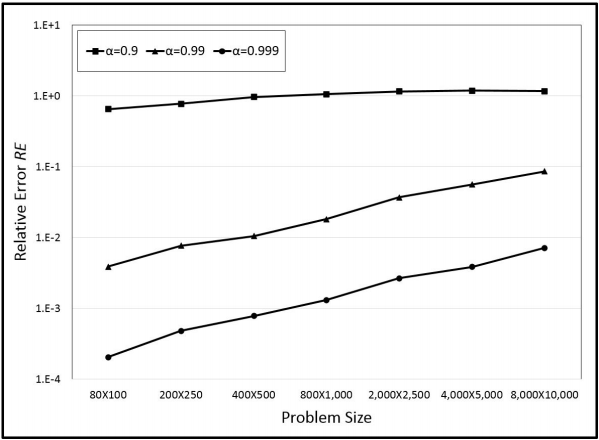


Рисунок 7: Графики относительных погрешностей RE, на логарифмической шкале, в зависимости от размерностей проблемы для LinSup по 3 различным значениям ядра , основанные на данных из таблицы на Рисунке 5. Для каждого относительные ошибки увеличивается с увеличением размерностей. Для всех размерностей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

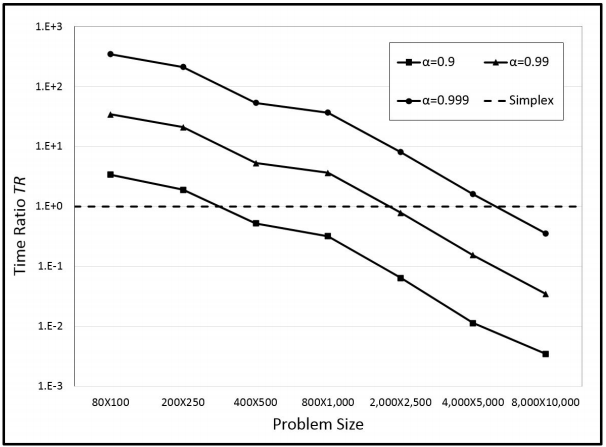


Рисунок 8: Графики отношения времени TR, на логарифмической шкале, в зависимости от размерности проблемы для LinSup по 3 различным значением ядра , основанные на данных из таблицы на Рисунке 6. Для каждого отношение время уменьшается с увеличением размерностей проблемы. Для всех размерностей проблемы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

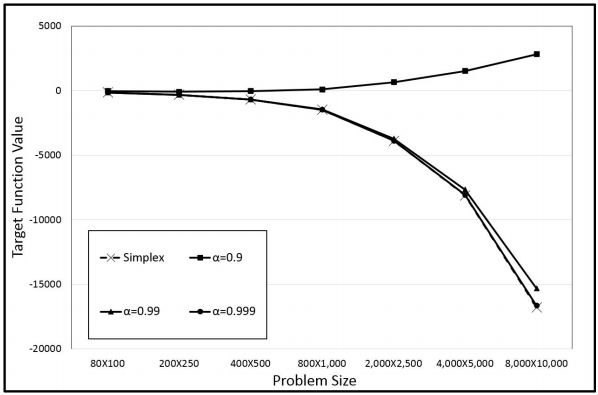


Рисунок 9: Зависимость значений целевой функции от размерности задачи по 3 различным значениям ядра . При самом большом значении движение алгоритма LinSup по уменьшению целевой функции достигает наилучших результатов. Эти значения близки к результатам, полученным из алгоритма Simplex.

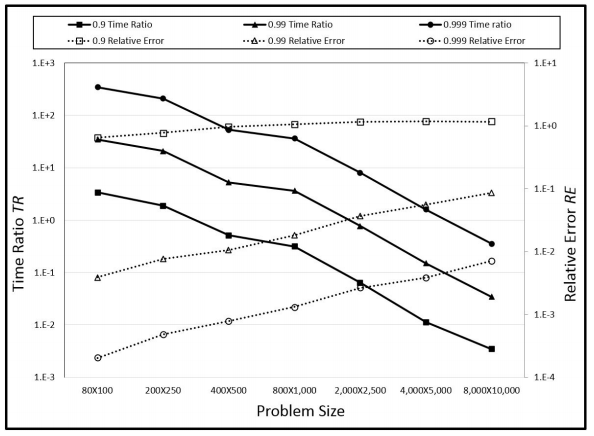


Рисунок 10: Наложение графиков 7 и 8. Этот рисунок наглядно показывает компромисс между уменьшением значения целевой функции и скоростью работы алгоритма LinSup.

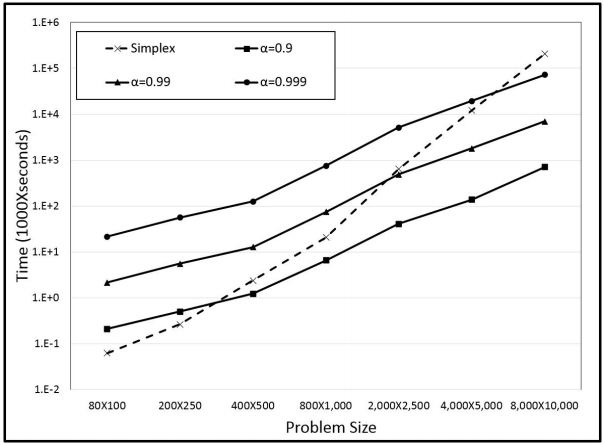


Рисунок 11: Время выполнения в тысячах секунд, на логарифмической шкале, в зависимости от размерности проблемы для алгоритма Simplex и для LinSup по 3 значениям ядра . Обратите внимание на резкое увеличение времени работы Simplex алгоритма (пунктирная линия).

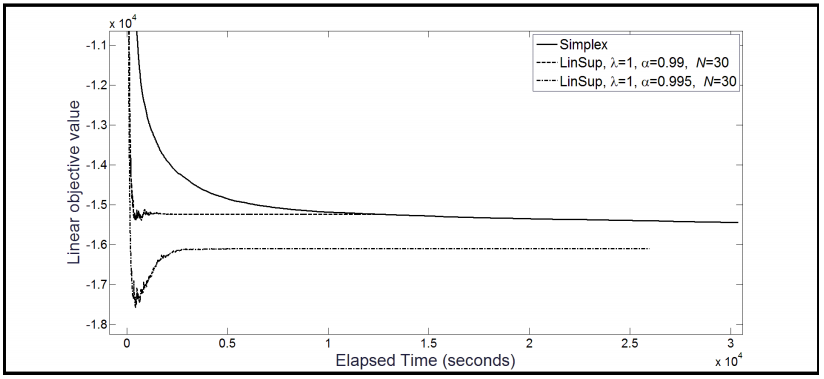


Рисунок 12: Если бы Simplex алгоритм останавливался не оптимально, например, после 5000 секунд, то оба запуска LinSup алгоритма давали бы более низкие значения линейной целевой функции.

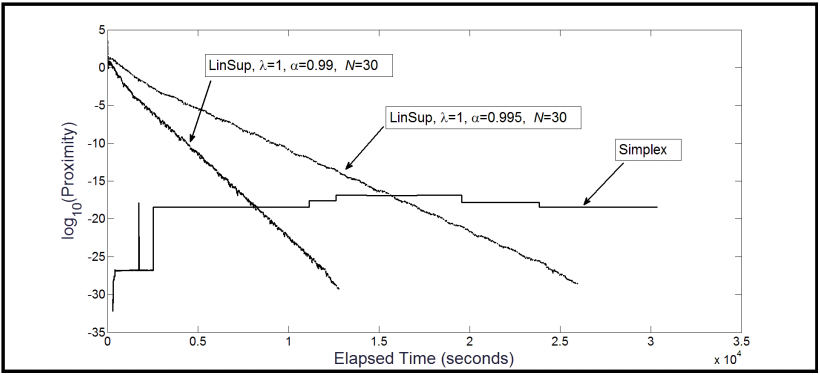


Рисунок 13: После 20000 секунд, оба запуска LinSup алгоритма будут иметь более низкие значения функции близости, чем Simplex алгоритм, а один запуск с наибольшим значением ядра будет даже иметь более низкое значение линейной целевой функции.