# 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

# 5.1 Результаты, и что они нам говорят