Основы

Компрессионное зондирование — это метод обработки сигналов, который восстанавливает сигналы на основе небольшого количества измерений, используя разреженность сигнала в некоторой области.

Процесс компрессивного зондирования (Compressive Sensing Process)

- Базисное представление:
  Сигнал может быть представлен в базисе, допускающем сжатие, например, вейвлетах или базисе Фурье.
- Матрица выборки:
  Матрица выборки строится для выборочной выборки сигнала. Эта матрица указывает, какие компоненты сигнала будут измеряться.
- Разреженное представление:
  Цель состоит в том, чтобы найти разреженное представление сигнала, в котором большинство коэффициентов равны нулю. Это достигается путем решения уравнения $Ax = b$ при условии, что x является разреженным.
- Реконструкция:
  Как только редкие коэффициенты найдены, исходный сигнал может быть восстановлен из этих коэффициентов, что позволяет эффективно его хранить и передавать.

CS предлагает структуру для одновременного зондирования и сжатия конечномерных векторов, которая основана на линейном снижении размерности. Довольно удивительно, что она предсказывает, что разреженные высокоразмерные сигналы могут быть восстановлены из крайне неполных измерений с использованием эффективных алгоритмов. Чтобы быть более конкретным, пусть x будет вектором длины n. В CS мы не измеряем x напрямую, а скорее получаем $m < n$ линейных измерений формы $y = Ax$ с использованием $m \times n$ CS матрицы A. В идеале матрица разработана так, чтобы максимально сократить количество измерений, при этом позволяя восстанавливать широкий класс сигналов из их векторов измерений y. Таким образом, мы хотели бы выбрать $m \ll n$. Однако это делает матрицу A неполноценной по рангу, что означает, что она имеет непустое нулевое пространство. Это означает, что для любого конкретного сигнала x_0 бесконечное количество сигналов x даст те же измерения $y = Ax = Ax_0$ для

выбранной CS матрицы. Поэтому для обеспечения восстановления мы должны ограничить себя специальным классом входных сигналов x.
Наиболее распространенной структурой сигнала, используемой в CS, является разреженность. В своей простейшей форме разреженность подразумевает, что x имеет лишь небольшое количество ненулевых значений. В более общем смысле, идеи CS могут применяться, когда подходящее представление x является разреженным. Удивительный результат, лежащий в основе CS, заключается в том, что если x (или подходящее представление x) является k-разреженным, т. е. имеет не более k ненулевых элементов, то его можно восстановить из $y = Ax$, используя количество измерений m, которое имеет порядок $k\,logn$. Более того, восстановление возможно с использованием простых алгоритмов с полиномиальным временем. Кроме того, можно показать, что эти методы устойчивы к шуму и неправильному моделированию x.

Алгоритмы

Поиск ортогонального соответствия (OMP(Orthogonal Matching Pursuit)) и итеративное установление порога (iterative thresholding). Сначала мы рассмотрим OMP, который начинается с нахождения столбца A, наиболее коррелирируемыми с результатами измерений. Затем алгоритм повторяет этот шаг, сопоставляя столбцы с остаточным сигналом, который получается путем вычитания вклада частичной оценки сигнала из исходного вектора измерения. Алгоритм формально определяется как *Алгоритм 1.1*, где $H_k(x)$ обозначает жесткий пороговый параметр оператора на x, который обнуляет все записи, за исключением k записей x с наибольшей величиной. Критерий остановки может состоять либо из ограничения на количество итераций, которое также ограничивает количество ненулевых значений в x, либо требование, чтобы $y \approx \widehat{Ax}$ в каком-то смысле. Обратите внимание, что в любом случае, если OMP запускается для m итераций, он всегда будет давать оценку $\widehat{x}$ такой, что $y = \widehat{Ax}$.
Итеративные Алгоритмы определения порога часто еще более просты. В качестве примера рассмотрим Iterative Hard Thresholding (IHT), который описан в *алгоритме 1.2*. Начиная с начальной оценки сигнала $\widehat{x_0} = 0$, алгоритм выполняет итерацию шага градиентного спуска с последующим жестким определением порога до тех пор, пока не будет достигнут критерий сходимости.
Простейшие гарантии для OMP гласят, что для ровно k-разреженного x с отсутствием шума измерений $y = Ax$, OMP восстановит x ровно за k итераций. Этот анализ выполняется как для матриц, удовлетворяющих RIP, так и для матриц с ограниченной когерентностью. Однако в обоих результатах требуемые

константы относительно малы, потому что результаты применимы только тогда, когда $m = O(k^2 log(n))$.

Алгоритм 1.1. Поиск ортогонального соответствия

Входные данные: CS-матрица/dictionary(словарь) A, вектор измерения y

Инициализировать: $\widehat{x_0} = 0$, $r_0 = y$, $\Lambda_0 = \varnothing$.

for i = 1; i := i + 1 до выполнения критерия остановки **do**

$g_i \leftarrow A^{\mathrm{T}} r_{i-1}$ {сформировать оценку сигнала по остаточному сигналу}

$\Lambda_i \leftarrow \Lambda_{i-1} \cup supp(H_1(g_i))$ {добавить самую большую остаточную запись в службу поддержки}

$\widehat{x_i}|_{\Lambda_i} \leftarrow A_{\Lambda_i}^{\dagger} y$, $\widehat{x_i}|_{\Lambda_i^c} \leftarrow 0$ {обновить оценку сигнала}

$r_i \leftarrow y - A\widehat{x_i}$ {обновить остаточное значение измерения}

end for

Вывод: Разреженное представление x

Алгоритм 1.2. Итеративное установление жесткого порога

Входные данные: CS-матрица/dictionary(словарь) A, вектор измерения y, уровень разреженности k

Инициализировать: $\widehat{x_0} = 0$.

for i = 1; i := i + 1 до выполнения критерия остановки **do**

$\widehat{x_i} = H_k(\widehat{x_{i-1}} + A^{\mathrm{T}}(y - A\widehat{x_{i-1}}))$

end for

Вывод: Разреженное представление x

Давайте представим жадные преследования как семейство алгоритмов, которые имеют следующие два общих фундаментальных шага: выбор элемента и обновление коэффициентов. Эти методы обычно инициализируются нулевой оценкой $\widehat{x}^{[0]} = 0$. При такой инициализации начальная остаточная ошибка $r^{[0]} = y - A\widehat{x}^{[0]} = y$ и опорный набор (т.е. индексы ненулевых элементов) первой оценки $\widehat{x}^{[0]}$ есть $T = \varnothing$. Затем каждая итерация обновляет эти количества, добавляя дополнительные элементы (столбцы из A) в набор опор T

и путем обновления сигнала оценивает $\hat{x}$, тем самым уменьшая остаточную ошибку наблюдения r. Это делается с незначительными вариациями, как указано в *алгоритме 8.1*.

Алгоритм 8.1. Общая структура жадного преследования (General greedy pursuit framework)

Входные данные: $y,\ A,\ k$

for i = 1; i := i + 1 до выполнения критерия остановки **do**

Вычислите $g^{[i]} = A^{\mathrm{T}} r^{[i]}$ и выберите элементы (столбцы из A) на основе величины элементов $g^{[i]}$

Рассчитайте пересмотренную оценку для $\hat{x}^{[i]}$ (и, следовательно, $\hat{y}^{[i]}$) за счет уменьшения функции стоимости

$$F(\hat{x}^{[i]}) = ||y - A\hat{x}^{[i]}||_2^2$$

end for

Вывод: $r^{[i]}$ и $\hat{x}^{[i]}$

Одним из простейших алгоритмов преследования является Matching Pursuit (MP) (известный как Чистый жадный алгоритм в теории приближений), обобщенный в *алгоритме 8.2*. Приближение является инкрементным, выбирая один столбец из A за раз и в каждой итерации обновляя только коэффициент, связанный с выбранным столбцом. На каждой итерации обновление $\hat{x}_{j^{[i]}}^{[i]} = \hat{x}_{j^{[i]}}^{[i-1]} + g_{j^{[i]}}^{[i]} / ||A_{j^{[i]}}||_2^2$ минимизирует аппроксимацию стоимостью $||y - A\hat{x}^{[i]}||_2^2$ относительно выбранного коэффициента. Здесь и на протяжении главы A_j — это j-й столбец матрицы A. Обратите внимание, что MP обычно неоднократно выберите те же столбцы из A, чтобы дополнительно уточнить аппроксимацию. Однако известно, что $||r^{[i]}||$ линейно сходится к нулю всякий раз, когда столбцы A охватывают R^m. Таким образом, MP остановится за конечное число итераций, если норма $r^{[i]}$ используется для определения критерия остановки алгоритма.

Алгоритм 8.2. Соответствующее преследование (MP(Matching Pursuit))

Входные данные: $y,\ A,\ k$

$r^{[0]} = y,\ \hat{x}^{[0]} = 0$

for i = 1; i := i + 1 до выполнения критерия остановки **do**

$g^{[i]} = A^{\mathrm{T}} r^{[i-1]}$

$j^{[i]} = argmax_j |g_j^{[i]}| / ||A_j||_2$

$\hat{x}_{j^{[i]}}^{[i]} = \hat{x}_{j^{[i]}}^{[i-1]} + g_{j^{[i]}}^{[i]} / ||A_{j^{[i]}}||_2^2$

$r^{[i]} = r^{[i-1]} - A_{j^{[i]}} g_{j^{[i]}}^{[i]} / ||A_{i^{[i]}}||_2^2$

end for

Вывод: $r^{[i]}$ и $\hat{x}^{[i]}$

Более сложная стратегия реализована в Orthogonal Matching Pursuit (OMP) (известный как ортогональный жадный алгоритм в теории приближений). В ОМП аппроксимация x обновляется на каждой итерации путем проецирования y ортогонально на столбцы A, связанные с текущим набором поддержки $T^{[i]}$. Таким образом, OMP минимизирует $||y - A\hat{x}||_2$ за все $\hat{x}$ с поддержкой $T^{[i]}$. Полный алгоритм приведен в *Алгоритме 8.3.*, где † на шаге 7 представляет собой псевдообратный оператор. Обратите внимание, что в отличие от МП минимизация производится относительно всех выбранных на данный момент коэффициентов:

$$\hat{x}_{T^{[i]}}^{[i]} = argmin_{\bar{x}_{T^{[i]}}} ||y - A_{T^{[i]}} \bar{x}_{T^{[i]}}||_2^2.$$

В отличие от MP, OMP никогда не выбирает элемент повторно, и остаток на любой итерации всегда ортогонально равен всем выбранным в данный момент элементам.

Есть две основные проблемы с применением OMP к крупномасштабным данным. Во-первых, ком-затраты на размещение и хранение одной итерации OMP довольно высоки для крупномасштабных проектов. Во-вторых, выбор по одному атому за раз означает, что ровно k итераций необходимы для аппроксимации y с помощью k атомов A. Когда k велико, это может быть непрактично.

Алгоритм 8.3. Ортогональное сопоставление (OMP(Orthogonal Matching Pursuit))

Входные данные: $y,\ A,\ k$

Инициализировать: $r^{[0]} = y,\ \hat{x}^{[0]} = 0,\ T^{[0]} = \varnothing$

for i = 1; i := i + 1 до выполнения критерия остановки **do**

$g^{[i]} = A^{\mathrm{T}} r^{[i-1]}$

$j^{[i]} = argmax_j |g_j^{[i]}| / ||A_j||_2$

$T^{[i]} = T^{[i-1]} \cup j^{[i]}$

$\hat{x}^{[i]}$

$r^{[i]} = y - A\hat{x}^{[i]}$

end for

Вывод: $r^{[i]}$ и $\hat{x}^{[i]}$

Семейство алгоритмов направленного преследования обобщенно в *алгоритме 8.4*. Целью введения направленных обновлений является получение приближения к ортогональной проекции с уменьшенной стоимостью вычислений. Здесь мы сосредоточимся на обновлении на основе стратегии градиента.

Естественным выбором направления обновления является отрицательный градиент функции стоимости $||y - A_{T^{[i]}} \bar{x}_{T^{[i]}}||_2^2$, то есть

$$d_{T^{[i]}}^{[i]} := g_{T^{[i]}}^{[i]} = A_{T^{[i]}}^{T}(y - A_{T^{[i]}} \hat{x}_{T^{[i]}}^{[i-1]}).$$

К счастью, у нас уже есть это как побочный продукт процесса отбора. Мы просто ограничим вектор $g^{[i]}$ (который уже вычислен) элементами $T^{[i]}$. С использованием вышеприведенной формулы, поскольку обновление направления приводит к самой базовой форме направленного преследования, которую мы называем Gradient Pursuit (GP)

Алгоритм 8.4. Направленное преследование (Directional Pursuit)

Входные данные: $y,\ A,\ k$

Инициализировать: $r^{[0]} = y,\ \hat{x}^{[0]} = 0,\ T^{[0]} = \varnothing$

for i = 1; i := i + 1 до выполнения критерия остановки **do**

$g^{[i]} = A^{\mathrm{T}} r^{[i-1]}$

$j^{[i]} = argmax_j |g_j^{[i]}| / ||A_j||_2$

$T^{[i]} = T^{[i-1]} \cup j^{[i]}$

Рассчитать направление обновления $d_{T^{[i]}}^{[i]}$; $c^{[i]} = A_{T^{[i]}} d_{T^{[i]}}^{[i]}$ и $a^{[i]} = \frac{\langle r^{[i]}, c^{[i]} \rangle}{||c^{[i]}||_2^2}$

$x_{T^{[i]}}^{[i]} := x_{T^{[i]}}^{[i-1]} + a^{[i]} d_{T^{[i]}}^{[i]}$

$r^{[i]} = r^{[i-1]} - a^{[i]} c^{[i]}$

end for

Вывод: $r^{[i]}$ и $\hat{x}^{[i]}$

Локальную оптимизацию можно превратить в итерационный алгоритм, установив z = x[i] , и в этом случае мы получим алгоритм Iterative Hard Thresholding (IHT) 8.5. Основываясь на текущей оценке x[i] , этот алгоритм жадно находит глобальный минимум ограниченной суррогатной цели. Алгоритм IHT прост в реализации и эффективен с вычислительной точки зрения. Помимо сложения векторов, основными вычислительными шагами являются умножение векторов на A и его транспонирование, а также частичная сортировка, необходимая для порогового шага. Поэтому требования к хранилищу невелики, и при использовании структурированных матриц измерения умножение на A и A^T также часто может быть эффективно выполнено.

Алгоритм 8.5. Итеративный алгоритм жесткого определения порога (IHT)

Входные данные: $y,\ A,\ k$, μ

Инициализировать: $\hat{x}^{[0]} = 0$

for i = 0; i := i + 1 до выполнения критерия остановки **do**

$\hat{x}^{[i+1]} = H_k(\hat{x}^{[i]} + \mu A^T(y - A\hat{x}^{[i]}))$

end for

Вывод: $\hat{x}^{[i]}$

Используя автоматическое уменьшение размера шага, можно показать, что алгоритм, который обобщен как алгоритм 8.6, будет сходиться к фиксированной точке. Если rank(A) = m и rank(AT) = k для всех T так, что |T| = k, то нормализованный алгоритм IHT сходится к локальному минимуму задачи оптимизации.

Алгоритм 8.6. Нормализованный итерационный алгоритм жесткого определения порога (IHT)

Входные данные: $y,\ A,\ k$

Инициализировать: $\hat{x}^{[0]} = 0,\ T^{[0]} = supp(H_k(A^T y))$

for i = 0; i := i + 1 до выполнения критерия остановки **do**

$g^{[i]} = A^{\mathrm{T}}(y - A\hat{x}^{[i]})$

$\mu^{[i]} = \frac{||g_{T^{[i]}}^{[i]}||_2^2}{||A_{T^{[i]}} g_{T^{[i]}}^{[i]}||_2^2}$

$\bar{x}^{[i+1]} = H_k(\hat{x}^{[i]} + \mu^{[i]} g^{[i]})$

$T^{[i+1]} = supp(\hat{x}^{[i+1]})$

if $T^{[i+1]} = T^{[i]}$ **then**

$\hat{x}^{[i+1]} = \bar{x}^{[i+1]}$

else if $T^{[i+1]} \neq T^{[i]}$ **then**

if $\mu^{[i]} \leq (1 - c)\frac{||\bar{x}^{[i+1]} - \hat{x}^{[i]}||_2^2}{||A(\bar{x}^{[i+1]} - \hat{x}^{[i]})||_2^2}$ **then**

$\hat{x}^{[i+1]} = \bar{x}^{[i+1]}$

else if $\mu^{[i]} > (1 - c)\frac{||\bar{x}^{[i+1]} - \hat{x}^{[i]}||_2^2}{||A(\bar{x}^{[i+1]} - \hat{x}^{[i]})||_2^2}$ **then**

repeat

$\mu^{[i]} \leftarrow \mu^{[i]}/(k(1 - c))$

$\bar{x}^{[i+1]} = H_k(\hat{x}^{[i]} + \mu^{[i]} g^{[i]})$

until $\mu^{[i]} \leq (1 - c)\frac{||\bar{x}^{[i+1]} - \hat{x}^{[i]}||_2^2}{||A(\bar{x}^{[i+1]} - \hat{x}^{[i]})||_2^2}$

$T^{[i+1]} = supp(\bar{x}^{[i+1]})$

$\hat{x}^{[i+1]} = \bar{x}^{[i+1]}$

end if

end if

end for

Вывод: $r^{[i]}$ и $\hat{x}^{[i]}$

Вместо точного вычисления $A^{\dagger}_{T^{[i+0.5]}}y$ в каждой итерации, которая может быть вычислительно требовательной, была предложена более быстрая приближенная реализация алгоритма CoSaMP. Эта быстрая версия заменяет точную оценку по методу наименьших квадратов $\hat{x}_{T^{[i+0.5]}} = A^{\dagger}_{T^{[i+0.5]}}y$ на три итерации градиентного спуска или сопряженного градиентного решателя.

Алгоритм 8.7. Поиск соответствия сжимающей выборке (CoSaMP)

Входные данные: $y,\ A,\ k$, μ

Инициализировать: $\hat{x}^{[0]} = 0, T^{[0]} = supp(H_k(A^T y))$

for i = 0; i := i + 1 до выполнения критерия остановки **do**

$g^{[i]} = A^{\mathrm{T}}(y - A\hat{x}^{[i]})$

$T^{[i+0.5]} = T^{[i]} \cup supp(g^{[i]}_{2k})$

$\hat{x}_{T^{[i+0.5]}}^{[i+0.5]} = A^{\dagger}_{T^{[i+0.5]}}y$

$T^{[i+1]} = supp(\hat{x}_k^{[i+0.5]})$

$\hat{x}_{T^{[i+1]}}^{[i+1]} = \hat{x}_{T^{[i+1]}}^{[i+0.5]}$

end for

Вывод: $\hat{x}^{[i]}$, $r^{[i]}$

В 8.8 правило остановки, основанное на разности $||y - A\hat{x}^{[i+1]}||_2 \geq ||y - A\hat{x}^{[i]}||_2$. Это гарантирует, что метод остается стабильным даже в режиме, в котором условие RIP не выполняется. Основное различие между этими двумя подходами заключается в размере набора, добавляемого к T[i] в каждой итерации, а также в дополнительном решении по методу наименьших квадратов, необходимом в SP. Более того, возможность замены решения по методу наименьших квадратов в CoSaMP на три градиентных обновления означает, что оно может быть реализовано гораздо эффективнее, чем SP

Алгоритм 8.8. Подпространственное преследование (SP)

Входные данные: $y,\ A,\ k$

Инициализировать: $\hat{x}^{[0]} = A_{T^{[0]}}^{\dagger} y,\ T^{[0]} = supp(H_k(A^T y))$

for i = 0; i := i + 1 до $||y - A\hat{x}^{[i+1]}||_2 \geq ||y - A\hat{x}^{[i]}||_2$ **do**

$g^{[i]} = A^{\mathrm{T}}(y - A\hat{x}^{[i]})$

$T^{[i+0.5]} = T^{[i]} \cup supp(g_k^{[i]})$

$\hat{x}_{T^{[i+0.5]}}^{[i+0.5]} = A_{T^{[i+0.5]}}^{\dagger} y$

$T^{[i+1]} = supp(\hat{x}_k^{[i+0.5]})$

$\hat{x}^{[i+1]} = A_{T+1}^{\dagger} y$

end for

Вывод: $\hat{x}^{[i]},\ r^{[i]}$

Единственное отличие алгоритма 8.9 от алгоритма 8.5 заключается в том, что мы заменили оператор жесткого порогового значения Hk на проектор модели UoS PU. Обратите внимание, что может использовать аналогичную стратегию для адаптивного определения μ, как это было предложено для метода IHT.

Алгоритм 8.9. Прогнозируемый алгоритм Ландвебера(PLA)

Входные данные: $y,\ A,$ μ

Инициализировать: $x^{[0]} = 0$

for i = 0; i := i + 1 до выполнения критерия остановки **do**

$\hat{x}^{[i+1]} = Pu(\hat{x}^{[i]} + \mu A^{*}(y - A\hat{x}^{[i]}))$, где A^{*} сопряженное A

end for

Вывод: $\hat{x}^{[i]}$

Алгоритм 8.10 описывает модифицированную версию стандартного алгоритма CoSaMP, применимую к структурированным разреженным моделям. Для этих моделей единственной необходимой модификацией является замена жесткого порогового шага на проекцию UoS.

Алгоритм 8.10. Алгоритм CoSaMP для структурированных разреженных моделей.

Входные данные: $y,\ A,\ k$

Инициализировать: $x^{[0]} = 0, T^{[0]} = supp(Pu(A^T y))$

for i = 0; i := i + 1 до выполнения критерия остановки **do**

$g = A^{\mathrm{T}}(y - Ax^{n})$

$T^{[i+0.5]} = T^{[i]} \cup supp(P_{U^2}(g))$

$x^{[i+0.5]}_{T^{[i+0.5]}} = A^{\dagger}_{T^{[i+0.5]}} y$

$T^{[i+1]} = supp(P_U(x^{[i+0.5]}))$

$x^{[i+1]}_{T^{[i+1]}} = x^{[i+0.5]}_{T^{[i+1]}}$

end for

Вывод: $\hat{x}^{[i]}$, $r^{[i]}$

Полное описание алгоритма Rank Aware ORMP кратко изложено в Алгоритме 8.11. Как и в случае с оригинальным ORMP, возможны эффективные реализации, основанные на факторизации QR. Обратите внимание, что крайне важно использовать векторы-столбцы ORMPnormalized, чтобы гарантировать повторный правильный выбор в максимальном случае ранга. При использовании аналогичной стратегии выбора в алгоритме MMV OMP имеет место эффект вырождения рангов – ранг матрицы остатков уменьшается с каждым правильным выбором, в то время как уровень разреженности обычно остается на k. Это означает, что алгоритм может приводить и приводит к неправильному выбору.

Алгоритм 8.11. Рекурсивный поиск соответствия с учетом порядка ранжирования (RA-ORMP)

Входные данные: Y, A

Инициализировать: $R^{[0]} = Y, T^{[0]} = \varnothing, \hat{x}^{[0]} = 0$

for i = 1; i := i + 1 до выполнения критерия остановки **do**

Вычислить ортонормированный базис для остатка: $U^{[i-1]} = orth(R^{[i-1]})$

$j^{[i]} = argmax_{j \notin T^{[i-1]}} ||A_j^Y U^{[i-1]}||_2 / ||P^{\perp}_{T^{[i-1]}} A_j||_2$

$T^{[i]} = T^{[i-1]} \cup j^{[i]}$

$\hat{x}^{[i]}_{\{T^{[i]},:\}} = A^{\dagger}_{T^{[i]}} Y$

$R^{[i]} = Y - A\hat{x}^{[i]}$

end for

После того, как для каждого субъекта k получено наилучшее преобразование τ_k, мы можем применить его обратное преобразование к обучающему набору Φk так, чтобы весь обучающий набор был выровнен по y. Затем можно найти глобальное разреженное представление $\hat{c}$ от y относительно преобразованного обучающего набора, решив оптимизационную задачу. Окончательная классификация выполняется путем вычисления расстояния l2 между y и его приближением $\Phi_k \delta_k(\hat{c})$ с использованием только обучающих изображений из k-го класса и присвоения y классу, который минимизирует расстояние.Полный алгоритм обобщен в алгоритме 12.1.

Алгоритм 12.1. Деформируемый SRC для распознавания лиц.

Входные данные: Фронтальные обучающие изображения $\Phi_1, \ldots, \Phi_c \in R^{m \times n}$ для C субъектов, тестовое изображение $y \in R^m$, и группа деформаций T.

for каждого субъекта k,

$\tau_k^0 \leftarrow I.$

do

$\bar{y}(\tau_k^i) \leftarrow \frac{y \circ \tau_k^i}{||y \circ \tau_k^i||_2}$; $J_k^i \leftarrow \frac{\partial}{\partial \tau_k} \bar{y}(\tau_k)|_{\tau_k^i}$;

$\Delta\tau_k = argmin||e||_1 \; s.t. \; \bar{y}(\tau_k^i) + J_k^i \Delta\tau_k = \Phi_k c + e.$

$\tau_k^{i+1} \leftarrow \tau_k^i + \Delta\tau_k$;

while $||\tau_k^{i+1} - \tau_k^i|| \geq \varepsilon.$

end

Установить $\Phi \leftarrow [\Phi_1 \circ \tau_1^{-1} | \Phi_2 \circ \tau_2^{-1} | \ldots | \Phi_c \circ \tau_c^{-1}]$.

Решить проблему минимизации: $\hat{c} = argmin||c||_1 + ||e||_1 \; s.t. \; y = \Phi c + e.$

Вычислить остаток $r_k(b) = ||c - \Phi_k \delta_k(c)||_2 \; for \; k = 1, \ldots, C.$

Вывод: identity(y) = $argmin_k r_k(c)$.

Алгоритм 12.2 преобразует задачу минимизации с ограничениями в последовательность задач безусловной минимизации с помощью "штрафной функции". Эта штрафная функция представляет собой сумму исходной функции цели и "штрафа" за нарушение ограничений.

Алгоритм 12.2. Метод увеличенного множителя Лагранжа для минимизации $l1$.

Входные данные: $y \in R^m,\ \Phi \in R^{m\times n}, c_1 = 0, e_1 = y, \nu_1 = 0.$

while не сходится (k = 1,2, …) **do**

$e_{k+1} = shrink(y - \Phi c_k + \frac{1}{\mu_k}\nu_k, \frac{1}{\mu_k})$;

$t_1 \leftarrow 1,\ z_1 \leftarrow c_k, w_1 \leftarrow c_k$;

while не сходится (l = 1,2, …) **do**

$w_{l+1} \leftarrow shrink(z_l + \frac{1}{\gamma}\Phi^T(y - \Phi\nu_l - e_{k+1} + \frac{1}{\mu_k}\nu_k), \frac{1}{\mu_k\gamma})$;

$t_{l+1} \leftarrow \frac{1}{2}(1 + \sqrt{1 + 4t^2})$;

$z_{l+1} \leftarrow w_{l+1} + \frac{t_{l-1}}{t_{l+1}}(w_{l+1} - w_l)$;

end while

$c_{k+1} \leftarrow w_l$;

$\nu_{k+1} \leftarrow \nu_k + \mu_k(y - \Phi c_{k+1} - e_{k+1})$;

end while

Вывод: $c^* \leftarrow c_k,\ e^* \leftarrow e_k$.