Московский Физико-Технический
Институт(Государственный Университет)

Выпускная квалификационная работа:
Решение линейных задач полуопределенного программирования модифицированным мультипликативно-барьерным методом

Факультет управления и прикладной математики
Кафедра интеллектальных систем
Аверьянов Ярослав Петрович

Научный руководитель: Д. ф.-м. н. Жадан В.Г.

г.Долгопрудный
2015г.

Оглавление

Глава 1

Введение

Линейные задачи полуопределенного программирования(SDP) - одна из наиболее значимых частей линейной оптимизации начиная с 90-х годов двадцатого столетия. SDP имеют множество приложений в теории управления, комбинаторной оптимизации и выпуклой оптимизации. Используя линейные задачи полуопределенного программирования, мы можем эффективно решать практические задачи. В настоящее время разработано много численных способов решения этой задачи, использующие прямой, двойственный и прямо-двойственный аффинно-масштабирующие типы решения. Если начальной точкой в этих случаях брать точку из относительной внутренности допустимого множества, то этот метод решения будет релаксационным, т.е. значение целевой функции будет постоянно убывать, но может произойти такой момент, что на одной из итерации точка попадет на границу допустимого множества, тогда при продолжении итерационного процесса мы можем получить неверное решение, так как итерационный процесс может "покинуть"допустимое множество. С целью предотвратить такие исходы мы будем рассматривать направление убывания как сумму двух направлений: первое - направление из минимальной грани конуса положительных полуопределенных матриц, содержащих текущую точку, второе - это направление, использующее собственный вектор двойственной невязки, подсчитанной в этой точке. Далее мы рассмотрим, как работают три варианта алгоритма: основной, из минимальной грани и алгоритм, использующий совместное направление, при котором возможен сход с минимальной грани. Будут представлены

графики зависимости времени работы алгоритмов и числа итераций, необходимых для их сходимости от начальной точки и точности метода. Также будет показано, что все три алгоритма являются релаксационными. Приложения SDP и некоторые особенности работы алгоритма будут рассмотрены в последних главах.

Глава 2

Постановка задачи

Если X есть квадратная матрица порядка n, тогда X есть положительно полуопределенная матрица порядка n, если

$$v^T X v \geq 0$$

для любого $v \in \mathbf{R}^n$.
Если X есть квадратная матрица порядка n, тогда X есть положительно определенная матрица порядка n, если

$$v^T X v > 0$$

для любого $v \in \mathbf{R}^n$ и $v \neq 0$.

Пусть $\mathbf{S}^n$ - пространство симметричных матриц порядка n, а $\mathbf{S}^n_+$ и $\mathbf{S}^n_{++}$ - множества положительно полуопределенных и положительно определенных матриц порядка n. Стоит отметить, что пространство $\mathbf{S}^n_+$ - выпуклый и замкнутый конус размерности $n(n+1)/2$. Введем скалярное произведение между двумя квадратными матрицами P и Q как след матрицы $P^T Q$

$$P \bullet Q = tr(P^T Q) = \sum\nolimits_{i,j=1}^{n} p_{ij} q_{ij}$$

Рассмотрим задачу полуопределенного программирования:

$$\begin{gathered} \min\ C \bullet X, \\ A_i \bullet X = b^i, \quad i = J^m, \qquad X \succeq 0. \end{gathered} \tag{2.1}$$

Матрицы C, A_i принадлежат пространству $\mathbf{S}^n$. Следует также отметить, что (2.1) очень близка к задаче линейного программирования, если рассматривать матрицу из $\mathbf{S}^n$ как вектор из n^2 координат.

Двойственной к (2.1) является задача

$$\begin{array}{c} \max \langle b, u \rangle, \\ \sum_{i=1}^m u^i A_i + V = C, \qquad V \succeq 0, \end{array} \tag{2.2}$$

в которой $b = (b^1, \ldots, b^m)^T \in \mathbf{R}^m$, $V \in \mathbf{S}^n$,

Здесь и далее предполагается, что эти задачи имеют решения. Введем допустимое множество в первоначальной задаче как
$\mathbf{F}_P$:

$$\mathbf{F}_P = \mathbf{F}_A \cap \mathbf{S}^n_+, \qquad \mathbf{F}_A = \left\{ X \in \mathbf{S}^n : \ A_i \bullet X = b^i, \ i \in J^m \ \right\}.$$

Его относительной внутренностью является множество $F_P^0 = F_P \cap \mathbf{S}^n_{++}$.

Введем далее еще некоторые понятия, которые пригодятся нам:
Для точки $X \in \mathbf{S}^n_+$ обозначим через $G_{min}(X; \mathbf{S}^n_+)$ минимальную грань конуса $\mathbf{S}^n_+$, содержащую X:

$$G_{min}(X; \mathbf{S}^n_+) = \{ Y \in \mathbf{S}^n_+ : \ R(Y) \subseteq R(X) \}$$

,где $R(M)$ - пространство столбцов матрицы M. Следует отметить, что если матрица $X \in \mathbf{S}^n_+$ имеет ранг r, то грань $G_{min}(X; \mathbf{S}^n_+)$ имеет размерность $r(r+1)/2$.
Обратимся теперь к допустимому множеству $\mathbf{F}_P$ в прямой задаче. Для матрицы $X \in \mathbf{F}_P$ обозначим через $G_{min}(X; \mathbf{F}_P)$ минимальную грань множества $\mathbf{F}_P$, содержащую X.

Глава 3

Алгоритм решения задачи

Пусть M - квадратная матрица порядка n. Символом $vecM$ обозначается прямая сумма ее столбцов. Для симметричных матриц имеет смысл вместо вектор-столбца $vecM$ рассматривать вектор-столбец $hvecM$. В него также помещаются последовательно сверху вниз столбцы матрицы M, но не полностью, а только их нижние части, начинающиеся с диагонального элемента. $svecM$ - то же самое, что и $hvec$, только элементы не на диагонале умножаются на $\sqrt{2}$. Далее введем элиминационную и дуплицирующую матрицы:
$L_n = \sum\limits_{i>=j}^{n} u_{ij}(vecE_{ij})^T$
$D_n = \sum\limits_{i>=j}^{n} u_{ij}(vecT_{ij})^T$, где u_{ij} - вектор размера $n(n+1)/2$ с 1 на $(j-1)n+i-j(j-1)/2$ позиции, остальные - 0.
Их свойства:
$L_n vecM = hvecM$.
$D_n hvecM = vecM$.
Матрица L_n имеет размер $n(n+1)/2 \times n^2$, матрица D_n — размер $n^2 \times n(n+1)/2$. Пусть E_n — квадратная матрица порядка n, все элементы которой равны единице. Пусть, кроме того, D_2 — диагональная матрица порядка $n_\triangle$, на диагонали которой располагается вектор $svecE_n$. Наряду с матрицами L_n и D_n в дальнейшем будем пользоваться также матрицами $\tilde{L_n} = D_2 L_n$ и $\tilde{D_n} = D_n D_2^{-1}$. Таким образом, если M — симметричная матрица порядка n, то, как можно проверить,

$$svecM = \tilde{L}_n vecM = \tilde{D}_n^T vecM, \quad vecM = \tilde{D}_n svecM.$$

Для матриц $\tilde{L}_n$ и $\tilde{D}_n$ сохраняется свойство: $\tilde{L}_n\tilde{D}_n = I_{n_\triangle}$.
При работе с векторами вида $vecM$ будем нумеровать их элементы не числами натурального ряда, а парами индексов из следующего набора:

$$J^n_= = \{(1,1),\ldots,(n,1),(1,2),\ldots,(n,2),\ldots,(1,n),\ldots,(n,n)\}. \qquad (3.1)$$

Следующий набор парных индексов удобно использовать при работе с векторами $hvecM$ или $svecM$. Это набор

$$J^n_\triangle = \{(1,1),\ldots,(n,1),(2,2),\ldots,(n,2),(3,3),\ldots,(n,n)\},$$

содержащий $n_\triangle$ парных индексов. В него входят только диагональные номера и младшие внедиагональные номера. Отметим, что при вычислении вектора u мы брали индесы i и j как раз из множества $J^n_\triangle$.

3.1 Основной алгоритм

Мы предполагали, что решения (1.1) и (1.2) существуют, следовательно

$$\begin{array}{rcl} X \bullet V & = & 0, \\ A_i \bullet X & = & b^i, \quad i \in J^m, \\ V & = & C - \sum_{i=1}^m u^i A_i, \\ X \succeq 0, & & V \succeq 0 \end{array} \qquad (3.2)$$

, в силу необходимых и достаточных условий оптимальности, обязательно имеет решение.

Обозначим через $X \circ V$ симметризованное произведение матриц X и V из $\mathbf{S}^n$, т.е. матрицу $X \circ V = (XV + VX)/2$. Данное произведение обладает хорошим свойством, а именно, для матриц $X \in \mathbf{S}^n_+$ и $V \in \mathbf{S}^n_+$ равенство $X \circ V = 0_{nn}$ возможно в том и только в том случае, когда $XV = VX = 0_{nn}$.
Воспользуемся также тем, что и равенство $X \bullet V = 0$ для X и V из $\mathbf{S}^n_+$ выполняется тогда и только тогда, когда $XV = VX = 0_{nn}$. Поэтому первое равенство из (3.2) может быть переписано в виде

$$X \circ V = 0_{nn}. \qquad (3.3)$$

Заменим теперь в системе (3.2), (3.3) матричные равенства на их векторные аналоги:

$$X^{\otimes} vecV = 0_{n^2}, \qquad A_{vec} vecX = b, \qquad vecV = vecC - A_{vec}^T u \tag{3.4}$$

Здесь $X^{\otimes}$ — кронекеровская сумма матрицы X, определяемая как

$$X^{\otimes} = [X \otimes I_n + I_n \otimes X]/2.$$

Через A_{vec} обозначена матрица размера $m \times n^2$, строками которой являются векторы $\mathrm{vec}A_i$, $i \in J^m$. Учтем симметричность матриц. Тогда система (2.1) заменяется на следующую:

$$\tilde{X}^{\otimes} svecV = 0_{n_\triangle}, \qquad A_{svec} svecX = b, \qquad svecV = svecC - A_{svec}^T u. \tag{3.5}$$

$\tilde{X}^{\otimes} = \tilde{L_n} X^{\otimes} \tilde{D_n}$,
A_{svec} — матрица размера $m \times n(n+1)/2$ со строками $svecA_i$, $i \in J^m$.
Поэтому (3.1) преобразовывается в уравнение относительно вектора u:

$$\Gamma(\tilde{X}^{\otimes}) u = A_{svec} \tilde{X}^{\otimes} svecC, \tag{3.6}$$

где через $\Gamma(\tilde{X}^{\otimes})$ обозначена матрица: $\Gamma(\tilde{X}^{\otimes}) = A_{svec} \tilde{X}^{\otimes} A_{svec}^T$.
Разрешая это уравнение относительно u:

$$u = u(X) = \Gamma^{-1}(\tilde{X}^{\otimes}) A_{svec} \tilde{X}^{\otimes} svecC. \tag{3.7}$$

Далее введем

$$V(u) = C - \sum_{i=1}^{m} u^i A_i, \qquad V(X) = V(u(X)).$$

Пусть точка X_* является решением прямой задачи (2.1). Тогда пара $[u_*, V_*]$, где $u_* = u(X_*)$, $V_* = V(u_*)$, есть решение двойственной задачи (2.2).

Пусть $svecV(X) = svecC - A_{svec}^T u(X)$. Заменим в первом равенстве из (2.2) вектор $svecV$ на найденный $svecV(X)$:

$$\tilde{X}^{\otimes} \left(I_{n_\triangle} - A_{svec}^T \Gamma^{-1}(\tilde{X}^{\otimes}) A_{svec} \tilde{X}^{\otimes} \right) svecC = 0_{n_\triangle}. \tag{3.8}$$

Это система $n_\triangle$ нелинейных уравнений относительно $n_\triangle$ переменных — компонент вектора $svecX$. Для ее решения можно применять различные численные методы решения систем нелинейных уравнений, в частности, метод простой итерации.
Обозначим

$$\Delta_{svec}(X) = \tilde{X}^{\otimes}\left(I_{n_\triangle} - A_{svec}^T\Gamma^{-1}(\tilde{X}^{\otimes})A_{svec}\tilde{X}^{\otimes}\right)svecC. \tag{3.9}$$

Беря $X_0 \in \mathbf{F}_P^0$ и применяя метод простой итерации, приходим к следующему итерационному процессу

$$svecX_{k+1} = svecX_k - \alpha_k\Delta_{svec}(X_k). \tag{3.10}$$

Здесь $\alpha_k > 0$ — шаг спуска. В матричном представлении итерационный процесс (3.10) запишется как

$$X_{k+1} = X_k - \alpha_k\Delta(X_k), \quad \Delta(X) = X \circ V(X). \tag{3.11}$$

Итерационный процесс (3.11) является релаксационным. Показано, что если шаг α_k берется постоянным и достаточно малым, то он обладает локальной сходимостью. Однако для того, чтобы добиться наибольшего убывания значения целевой функции на текущей итерации, следует взять шаг α_k максимально возможным при условии, что следующая точка X_{k+1} удовлетворяет неравенству: $X_{k+1} \succeq 0$. Но при таком выборе шага α_k точка X_{k+1} может оказаться на границе множества F_P. В этом случае направление $\Delta(X)$ принадлежит касательному подпространству к конусу $\mathbf{S}_+^n$ в этой точке и, в принципе, может выводить за пределы данного конуса и, стало быть, за пределы $\mathbf{F}_P$.

3.2 Направление из минимальной грани

Пусть теперь X принадлежит относительной границе множества $\mathbf{F}_P$, т.е. среди собственных чисел матрицы X имеются нулевые. Как изменится направление $\Delta(X)$, если дополнительно потребовать, чтобы оно принадлежало минимальной грани, содержащей X? С этой целью обратимся к “сужению” исходной задачи (2.1) на эту грань.

Пусть $X \in \mathbf{S}_+^n$ и ранг X равен $r < n$. Для матрицы X имеет место разложение $X = QD(\eta)Q^T$, причем положительные собственные числа находятся в начале вектора η. Для этого разложения, мы рассматриваем сингулярное разложение матрицы X и, так как мы находимся в пространстве симметричных положительно полуопределенных матриц, то это разложение будет верно и для нашего случая. Пусть Q_B — левая $n \times r$ подматрица ортогональной матрицы Q, а $D(\eta_B)$ — левая верхняя диагональная подматрица матрицы $D(\eta)$ порядка r, т.е. $X = Q_B D(\eta_B) Q_B^T$. Тогда указанное "сужение" задачи (2.1) на минимальную грань $G_{min}(X; \mathbf{S}_+^n)$ запишется как

$$\begin{gathered} \min\ C \bullet X, \\ A^i \bullet X = b^i, \quad i \in J^m, \qquad X \in G_{min}(X; \mathbf{S}_+^n). \end{gathered} \tag{3.12}$$

Также известно, что $X \in G_{min}(X; \mathbf{S}_+^n)$ представима в виде $X = Q_B Z Q_B^T$, где $Z \in \mathbf{S}_+^r$. Подставляя в (3.12) вместо X данное разложение, приходим к

$$\begin{gathered} \min\ C^{Q_B} \bullet Z, \\ A_i^{Q_B} \bullet Z = b^i, \quad i \in J^m, \qquad Z \succeq 0, \end{gathered} \tag{3.13}$$

где $C^{Q_B} = Q_B^T C Q_B$ и $A_i^{Q_B} = Q_B^T A_i Q_B$, $i \in J^m$. Двойственной к этой задаче будет

$$\begin{gathered} \max \langle b, u \rangle, \\ \textstyle\sum_{i=1}^m u^i A_i^{Q_B} + V^{Q_B} = C^{Q_B}, \qquad V^{Q_B} \succeq 0. \end{gathered} \tag{3.14}$$

Разрешая эту задачу аналогично предыдущему пункту, приходим к следующим выражениям:

$$\tilde{X}_{Q_B}^{\otimes} = \tilde{D}_n^T (Q_B \otimes Q_B) \tilde{L}_r^T D(\tilde{\eta}_B^{\otimes}) \tilde{L}_r (Q_B^T \otimes Q_B^T) \tilde{D}_n. \tag{3.15}$$

$\Gamma(\tilde{X}_{Q_B}^{\otimes}) = A_{svec} \tilde{X}_{Q_B}^{\otimes} A_{svec}^T$
$\Omega(\tilde{X}_{Q_B}^{\otimes}) = I_{n\triangle} - A_{svec}^T \Gamma^{-1}(\tilde{X}_{Q_B}^{\otimes}) A_{svec} \tilde{X}_{Q_B}^{\otimes}$
$\Delta_{svec}^{Q_B}(X) = \tilde{X}_{Q_B}^{\otimes} \Omega(\tilde{X}_{Q_B}^{\otimes}) svec C$

Недостатком направления $\Delta_{svec}^{Q_B}(X)$ есть невозможность, двигаясь вдоль него, покинуть минимальную грань. Следовательно, необходимо каким-то образом модифицировать направление из минимальной грани для

того, чтобы находить оптимальные точки и на других минимальных гранях $\mathbf{F}_P$. Поэтому теперь мы будем брать направление как сумму двух направлений: одно лежит в минимальной грани, другое покидает ее.

3.3 Возмущенное направление

Пусть по-прежнему точка X принадлежит границе $\mathbf{F}_P$ и пусть $V(X)$ не есть положительно полуопределенная матрица. Для $V(X)$ также имеет место разложение $V(X) = HD(\theta)H^T$, где H — ортогональная матрица со столбцами h_i, $i \in J^n$, $\theta \in \mathbf{R}^n$ — вектор собственных значений $V(X)$. Считаем, что ранг матрицы $V(X)$ равен $s \leq n$ и что первым s столбцам $h_1, h_2, \ldots, h_s$ матрицы H соответствуют ненулевые собственные значения $V(X)$. Тогда

$$V(X) = \sum_{j=1}^{s} \theta_j H_j, \qquad H_j = h_j h_j^T \tag{3.16}$$

Возьмем далее некоторое $\varepsilon > 0$. Теперь вместо направления $\Delta^{Q_B}(X)$, принадлежащего минимальной грани, нас будет интересовать направление

$$\Delta^{Q_B, h_{j*}}(X) = \Delta^{Q_B}(X) + \Delta^{h_{j*}}(X), \tag{3.17}$$

где

$$\Delta^{Q_B}(X) = Q_B \Lambda_B Q_B^T, \quad \Lambda_B = D(\eta_B) \circ V^{Q_B}(\tilde{u}), \qquad \Delta^{h_{j*}}(X) = V^{h_{j*}}(\tilde{u}) H_{j_*}. \tag{3.18}$$

Здесь первое направление $\Delta^{Q_B}(X)$ принадлежит грани $G_{min}(X; \mathbf{S}_+^n)$, содержащей точку X. Поскольку $h_{j_*} \notin L(Q_B)$, второе направление $\Delta_{h_{j*}}(X)$ выводит за пределы этой грани.

Пусть $\tilde{G}_{hj*} = (h_{j_*}^T \otimes h_{j_*}^T)\tilde{D}_n$ и $\tilde{E}_{hj_*}^{\otimes} = \tilde{G}_{hj_*}^T \tilde{G}_{hj*}$.

Введем направление и число:

$\psi = \tilde{G}_{hj*} A_{svec}^T \Gamma^{-1}(\tilde{X}_{Q_B}^{\otimes}) A_{svec} \tilde{G}_{hj_*}^T > 0 \qquad c(\varepsilon, \psi) = (1 + \varepsilon\psi)^{-1} > 0$

Тогда направление смещения вектора $svecX_k$ следует брать как:

$$\begin{gathered} \Delta^{Q_B, h_{j*}}(X) = [\tilde{X}_{Q_B}^{\otimes} \Omega(\tilde{X}_{Q_B}^{\otimes}) + \\ \tilde{c}(\varepsilon, \psi) \Omega^T(\tilde{X}_{Q_B}^{\otimes}) \tilde{E}_{h_{j*}}^{\otimes} \Omega(\tilde{X}_{Q_B}^{\otimes})] svecC \end{gathered} \tag{3.19}$$

Где $\tilde{c}(\varepsilon, \psi) = \varepsilon c(\varepsilon, \psi)$.

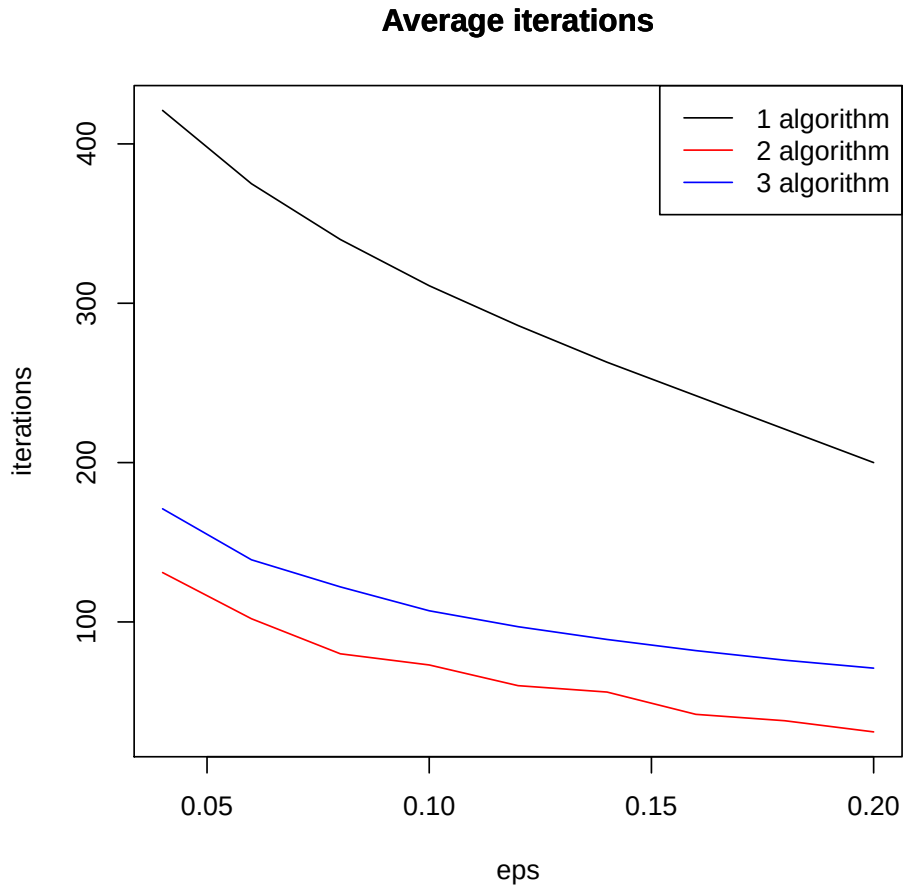
Следует также отметить, что при выборе соответствующего направления наш метод остается релаксационным и при этом он остается в множестве $\mathbf{F}_P$.

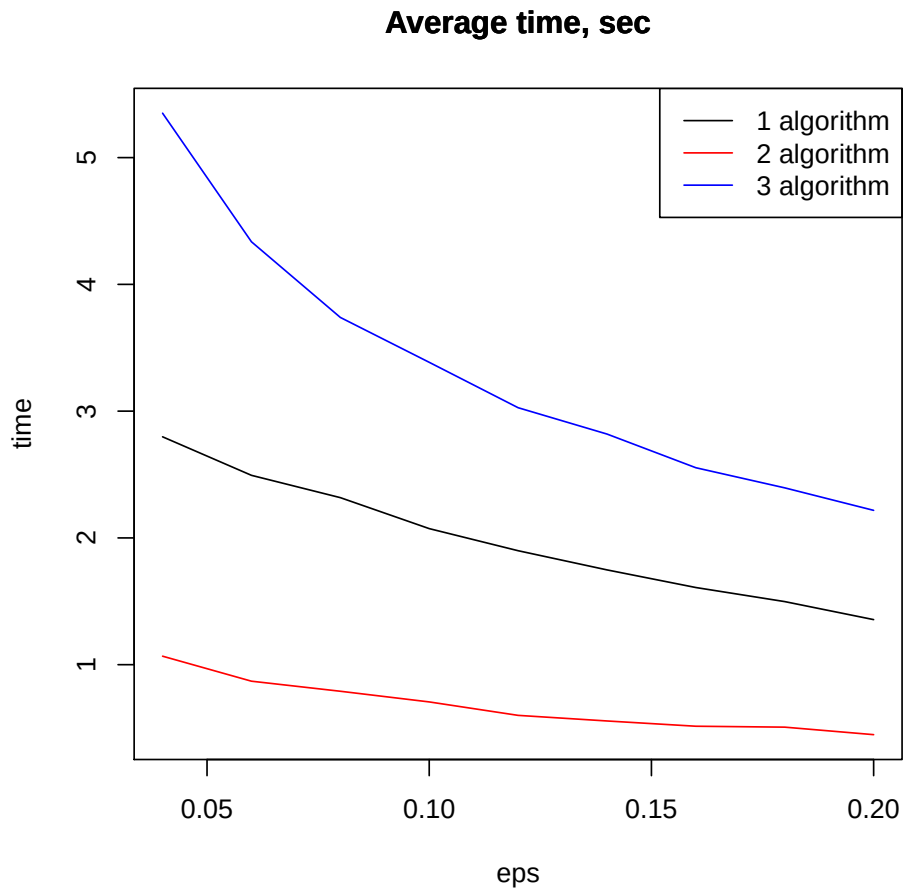
Теперь мы получили алгоритм, который будет универсальным вне зависимости от того, где находится текущая точка(на границе или в относительной внутренности). Однако этот алгоритм использует множество функций, чтобы в конце посчитать финальное направление, следовательно логично предположить, что потребуется много времени для его вычисления.

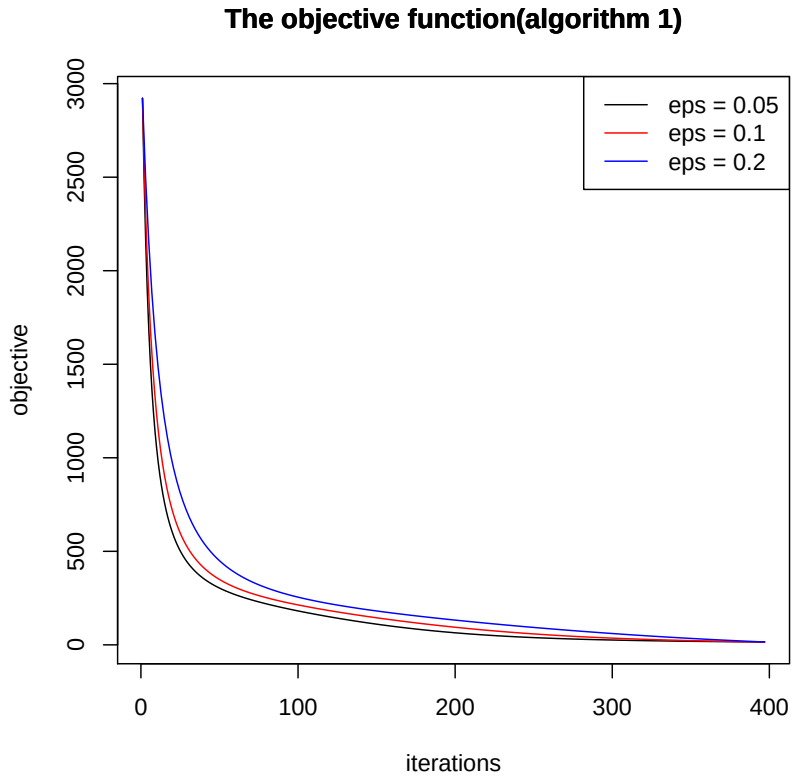
Глава 4

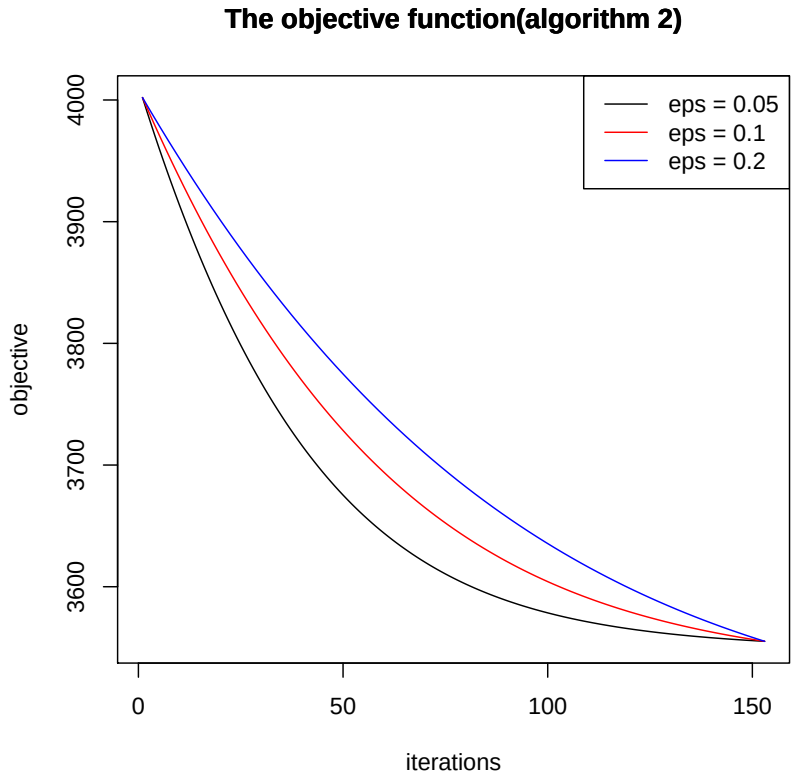
Результаты численного решения

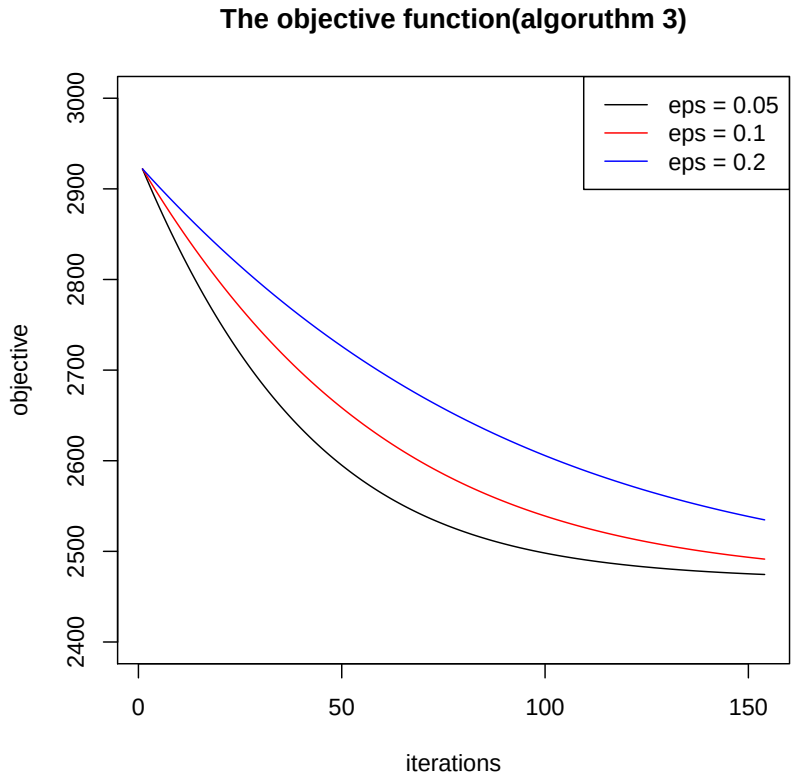
Далее было проведено численное решение задачи полуопределенного программирования(SDP) на языке Python при помощи библиотек NumPy и SciPy(библиотеки для линейной алгебры). Были построены графики зависимости среденего числа итераций и времени работы алгоритмов от точности, с которой мы хотим получить решение при достаточно малых $\alpha_k = 0.001$, показано, что все три алгоритма являются релаксационными.




Average time, sec
1 algorithm
2 algorithm
3 algorithm
time
1
2
3
4
5
0.05
0.10
0.15
0.20
eps

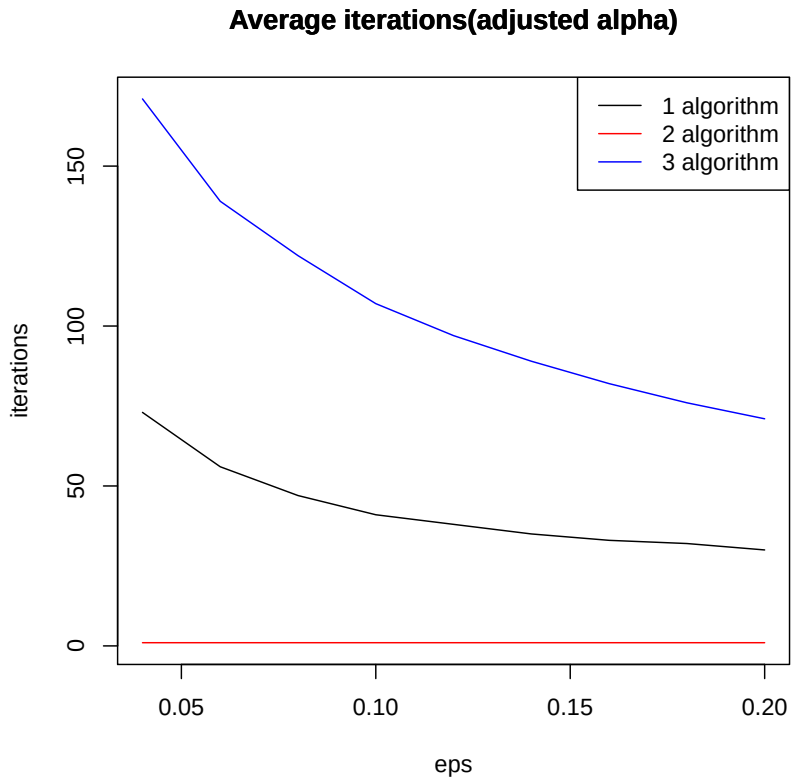

The objective function(algorithm 1)
eps = 0.05
eps = 0.1
eps = 0.2
objective
0
500
1000
1500
2000
2500
3000
0
100
200
300
400
iterations

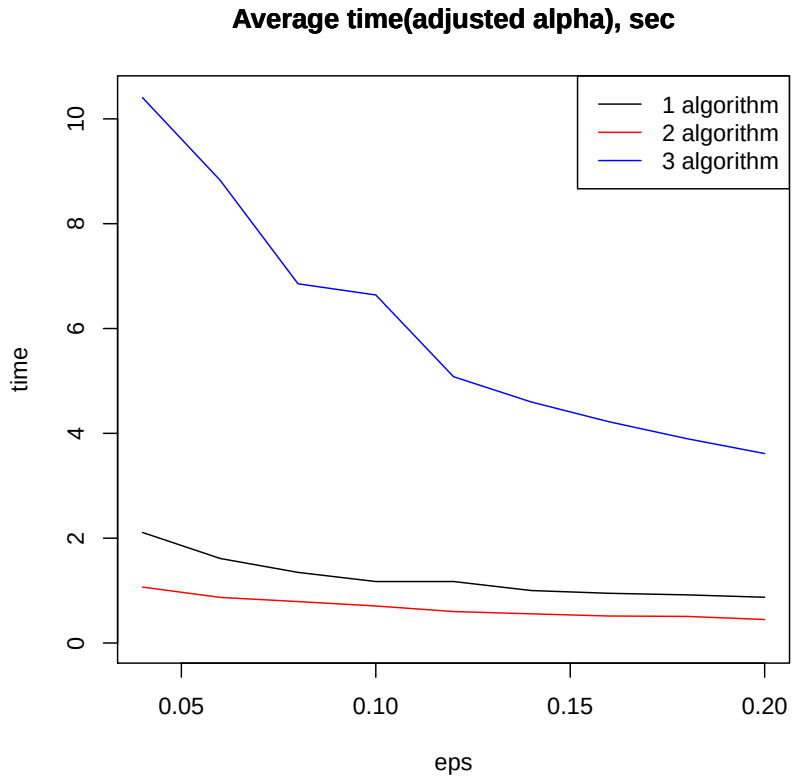




Далее строились зависимости среднего числа итераций и времени работы алгоритмов от точности, с которой мы хотим получить решение при максимальных α_k: на следующей$(k+1)$ итерации матрица принадлежала

допустимому множеству F_P.





Как мы видим из графиков, в случае малых α_k первый алгоритм

требует очень много итераций, но при этом время его работы оказывается меньше, чем у третьего алгоритма. В случае уже настроенных α_k, которые находились с помощью коньюктивного приведения симметричной матрицы $D(\eta_B)$, число итераций для всех трех алгоритмов существенно снижается и при этом третий алгоритм оказывается самым неэффективным как с точки зрения количества итераций, так и времени работы, второй алгоритм сходится и вовсе за один шаг.

Начальной точкой в первом и третьем алгоритме выбиралась матрица:
$\begin{pmatrix} 31 & 21 & 14 \\ 21 & 50 & 67 \\ 14 & 67 & 135 \end{pmatrix} > 0.$

Начальная точка второго алгоритма - вырожденная матрица: $\begin{pmatrix} 10 & 20 & 30 \\ 30 & 60 & 90 \\ 30 & 90 & 190 \end{pmatrix}$.

Матрицы C $= \begin{pmatrix} 1 & 2 & 3 \\ 2 & 6 & 7 \\ 3 & 7 & 11 \end{pmatrix}$, A1 $= \begin{pmatrix} 2 & 2 & 14 \\ -1 & 1 & 0 \\ -4 & 3 & 2 \end{pmatrix}$,

A2 $= \begin{pmatrix} 1 & 2 & 13 \\ 1 & 9 & 8 \\ -5 & 1 & 1 \end{pmatrix}$, A3 $= \begin{pmatrix} 2 & 1 & 5 \\ 1 & 3 & -7 \\ -6 & 7 & 2 \end{pmatrix}$.

Глава 5

Приложения алгоритма

Рассмотрим некоторые приложения алгоритма решения SDP: задача максимального разреза графа и задача минимизации собственных значений.

5.1 Максимальный разрез графа

Пусть G - неориетированный граф с узлами $N = 1, ..., n$ и с множеством ребер E. Пусть $w_{ij} = w_{ji}$ - вес ребер для $(i, j) \in E$. Предполагается, что $w_{ij} \succeq 0$. MAX CUT problem формулируется следующим образом: определить подмножество S множества узлов N такое, что сумма весов ребер, которые идут из S в S^c было максимально.
Сформулируем MAXCUT problem следующим образом: пусть $x_j = 1$, если $j \in S$ и $x_j = -1$, если $j \in S^c$. Тогда

$$MAXCUT : max(\frac{1}{4}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(1 - x_i x_j)) \tag{5.1}$$

$$\text{При условии } x_j \in \{-1, 1\}, j = 1, ..., n.$$

Пусть теперь

$$Y = xx^T \tag{5.2}$$

Также определим W как матрицу, у которой элемент $(i, j)^{th} = w_{ij}$ при $i = 1, ..., n$ и $j = 1, ..., n$. Тогда MAXCUT может быть сормулирована так:

$$MAXCUT : max(\frac{1}{4}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} - W \bullet Y) \tag{5.3}$$

При условии $x_j \in \{-1, 1\}, j = 1, ..., n$ и $Y = xx^T$.

Далее заметим, что $Y = xx^T$ - симметричная положительно определенная матрица, тогда

$$MAXCUT : max(\frac{1}{4}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} - W \bullet Y) \tag{5.4}$$

П.у. $Y_{jj} = 1, j = 1, ..., n, \qquad Y \succeq 0$

5.2 Задача минимизации собственных значений

Обычно задача оптимизации собственного значения формулируется следующим образом: составить матрицу $S = B - \sum_{i=1}^{k} w_i A_i$ для заданной симметричной матрицы B и $A_i, i = 1, ..., k$, используя веса $w_1, ..., w_k$, таким образом, чтобы минимизировать разность наибольшего и наименьшего собственных значений матрицы S.

$$EOP : min_{w,S}\lambda_{max}(S) - \lambda_{min}(S) \tag{5.5}$$

П.у. $S = B - \sum_{i=1}^{k} w_i A_i$

Покажем, как эта задача сведется к SDP.

Вспомним, что S может быть представлена как $S = QDQ^T$, где Q – ортогональная матрица и D – диагональная матрица, состоящая из собственных значений S. Условия:

$$\lambda I \preceq S \preceq \mu I$$

могут быть записаны как:

$$Q(\lambda I)Q^T \preceq QDQ^T \preceq Q(\mu I)Q^T.$$

Умножая это неравенство на Q^T слева и Q справа, получаем:

$$\lambda I \preceq D \preceq \mu I$$

Что эквивалентно

$$\lambda \leq \lambda_{min}(S)\lambda_{max}(S) \leq \mu$$

Таким образом EOP может быть записана как:

$$EOP : minimize_{w,S,\mu,\lambda} \quad \mu - \lambda \tag{5.6}$$

$$S = B - \sum_{i=1}^{k} w_i A_i \tag{5.7}$$

$$\lambda I \preceq S \preceq \mu I \tag{5.8}$$

Это задача SDP.

Глава 6

Заключение

В данной работе была рассмотрена задача полуопределенного программирования(SDP), а точнее модифицированный мультипликативно-барьерный метод, в котором сначала рассматривался основной алгоритм решения SDP, но главный его недостаток состоял в том, что этот метод можно рассматривать только в $\mathbf{F}_P^0$, поэтому далее было разработано направление из минимальной грани, чтобы на каждой итерации точки могли лежать на этой минимальной грани. После этого был рассмотрен алгоритм, использующий возмущенное направление, в котором учитывались все особенности предыдущих двух алгоритмов и поэтому это универсальный вариант решения SDP.

В практической части были исследованы эффективность(среднее число итераций) и время работы трех методов. Брались десять точек $\varepsilon \in [0.05, 0.2]$ и достаточно малые $\alpha_k = \alpha = 0.001$. Оказалось, что первый метод существенно уступает второму и третьему по эффективности, но третий алгоритм использует большего всего времени для того, чтобы сойтись к оптимальной точке. К тому же было показано, что все три метода - релаксационные, т.е. на каждой итерации значение целевой функции обязательно должно убывать. После этого шаги алгоритмов α_k брались таким образом, чтобы при каждой следующей итерации матрица находилась в допустимом множестве и шаг α_k был максимален. Оказалось, что второй алгоритм сходится за одну итерацию и время, которое требуется для сходимости, меньше 1 секунды. Также существенно уменьшилось количество итераций, однако увеличилось время работы программы и

третий алгоритм оказался самым неэффективным. Если мы хотим брать самый эффективный алгоритм, то, безусловно, полезно знать, в какой точке мы находимся и к какой точке сойдется итерационный процесс. Исходя из этого и нужно выбирать один из трех типов алгоритма решения SDP.

Приложение А

Дополнение

Здесь я выкладываю код на языке Python, в котором осуществлялся подсчет направлений убывания целевой функции для всех трех алгоритмов:

```
def iteration(X0,a,c,alpha,eps):
    w = 0
    n_trian = len(X0[:][0])*(len(X0[:][0]) + 1)/2
    cur = X0
    res0 = np.array([0]*len(svec(cur)))
    res = svec(cur)
    while (np.linalg.norm(res - res0) >= eps):
        delta_svec = np.array(X_kron_tilda(cur).dot(np.eye(n_trian) -
            Asvec(a).T.dot(np.linalg.inv(gamma_X_tilda(cur,a))).dot(Asvec(a)).dot(X_kron_tilda(c
        delta_matrix = np.array(symmetric_mult(cur,Vu(cur,a,c)))
        cur = cur - alpha*delta_matrix
        res0 = res
        res = svec(cur) - alpha*delta_svec
        w = w + 1
    return np.array((cur,res,w))

def iteration_min_edge(X0,a,c,alpha,eps):
    w = 0
    n = len(X0[:][0])
    n_trian = len(X0[:][0])*(len(X0[:][0]) + 1)/2
    cur = X0
    res0 = np.array([0]*len(svec(cur)))
    res = svec(cur)
    while (np.linalg.norm(res - res0) >= eps):
        delta_svec = delta_svec_qb(cur,a,c)
```

```
        res0 = res
        res = svec(cur) - alpha*delta_svec
        cur = cur - alpha*changing(delta_svec,n)
        w = w + 1
    return np.array((cur,res,w))

def iteration_final_dir(X0,a,c,alpha,eps):
    w = 0
    n = len(X0[:][0])
    n_trian = len(X0[:][0])*(len(X0[:][0]) + 1)/2
    cur = X0
    res0 = np.array([0]*len(svec(cur)))
    res = svec(cur)
    while (np.linalg.norm(res - res0) >= eps):
        delta_svec = delta_svec_qb_hj(cur,a,c,eps)
        res0 = res
        res = svec(cur) - alpha*delta_svec
        cur = cur - alpha*changing(delta_svec,n)
        w = w + 1
    return np.array((cur,res,w))
```

Литература

1. Handbook of Semidefinite Programming / eds. H. Wolkowicz, R. Saigal, L. Vandenberghe. Dordrecht: Kluwer Acad. Publ., 2000. 656 p.
2. Magnus J.R., Neudecker H. The elimination matrix: some lemmas and applications // SIAM J. Alg. Disc. Meth. 1980. Vol. 1.
no. 4. P. 422–449. 3. Жадан В.Г. Прямой мультипликативно-барьерный метод с наискорейшим спуском для линейной задачи полуопределенного программирования // Оптимизация и приложения. Выпуск 2. М.: ВЦ РАН, 2011. С. 107-131.
4. Бабынин М.С. , Жадан В.Г. Прямой метод внутренней точки для линейной задачи полуопределенного программирования // Журнал вычисл. математики и матем. физики. 2008. Т. 48, № 10. С. 1780–1801.
5. Robert M. Freund Introduction to Semidefinite Programming (SDP) March, 2004. 54 p.