Polinomiális LP algoritmusok

Matematika Bsc szakdolgozat

Leitereg Miklós

Témavezető:

Jüttner Alpár

Operációkutatás Tanszék

Eötvös Loránd Tudományegyetem
Természettudományi Kar

2018

Köszönetnyilvánítás

Szeretném megköszönni témavezetőmnek, Jüttner Alpárnak a szakirodalom összegyűjtésében, és a téma kiválasztásában nyújtott segítséget, a rendszeres konzultációkat, valamint a szakdolgozat elkészítéséhez adott hasznos tanácsokat.

A szakdolgozat az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával készült el.
(EFOP-3.6.3-VEKOP-16-2017-00002).

Tartalomjegyzék

1 Bevezetés

Szakdolgozatomban az operációkutatás egyik fontos alapfeladatával az LP-feladattal foglalkoztam. Az LP-feladat általános volta miatt sok probláma felírható LP-problámaként. Illetve az mivel IP probléma Brach-and-Bound algoritmus segítségével az LP-problémára visszavezethető. Egy gyors LP-algoritmus segíthet megoldani NP-nehéz feladatokat.
Az általam vizsgált algoritmusok a feladat geometriai hátterét kihasználva egyszerű frissítési és skálázási lépéseket használva polinomiális futásidőt tudnak elérni. Bár a belső pontos módszerektől elmarad a futásidőbecslésük, ügyes implementációval az ellipszoid módszerrel ellentétben gyakorlatban is használható LP megoldót kapunk.

A 2. fejezetben egy történeti áttekintést nyújtok a polinomiális LP algoritmusokról. A 3. fejezetben ismertetem a jelökéseket, a felhasznált fogalmakat, és pár alapvető összefüggést amikre később szükség lesz. A 4. és 5. fejezetekben leírok és részletesen elemzek egy-egy algoritmust, amik egyszerű elemi műveleteket használva polinomiális futásidőt tudnak elérni. A 6. fejezetben leírom az algoritmusok implementációja során szerzett tapasztalatok, és összehasnolnítom a két algoritmus futásidejét a gyakorlatban.

2 Történeti áttekintés

Az LP feladat megoldása a kezdetektől fogva az operációkutatás egyik központi témája volt. Bár csak lineáris feltételeket, és célfüggvényt tudunk ebben a témakörben kezelni, az idők során mégis, hogy kiderült sok fontos probléma felírható lineáris programozási feladatként. Sok példát láthatunk erre a gráf algorimtusok körében. A legrövidebb út, maximális folyam, maximális párosítás megkeresése mind visszavezethető LP feladatok megoldására. Az idők során többféle megközelítéssel próbálták az LP feladotot megoldani. Ebben a fejezetben sorra veszem az LP megoldó algoritmusok néhány fő típusát.

Először Neumann vetett fel egy módszert az LP-feladat megoldására. Az Algoritmus a b vektort állítja elő A oszlopvektorainak konvex kombinációjaként, úgy hogy kezdetben veszi A oszlopvektorainak egy tetszőleges konvex kombiációját, legyen ez y. Minden lépésben kiválasztja azt az oszlopvektort amivel $b - y$ a legkisebb szöget zárja be, és konstruálja y-nak és az oszlopvektornak azon konvex kombinációját, ami b-hez a legközelebb van.

Később a fenti ötletből kiindulva, a paramétereket változtatva, több hasonló módszert vezettek be. Ezeket gyűjtőnéven *elemi iterációs módszereknek* nevezzük. A szakdolgozat során elsősorban ilyen algoritmusokkal fogok foglalkozni. Az egyik legnagyobb előnyük az, hogy minden elvégzedő művelet egyszerű, így egy iterációt nagyon gyorsan el lehet végezni.
Az ilyen típusú módszereket általánosan is meg tudjuk adni.
Az egyszerűség kedvéért tegyük fel, hogy az most origót akarjuk előállítani A oszlopainak lineáris kombinációjaként. Legyen y aktuálisan előállított kombináció, és x az a vektor, ami az előállításban az egyes oszlopvektorok együtthatóit tartalmazza. Megadunk egy iterációt $y = Ax$ hosszának csökkentésére.
Legyen a az a vektor, ami $-y$-nal a legkisebb szöget zárja be. Minden lépésben az új y-t $y := \alpha y + \beta a$ alakban kapjuk, ahol α-t és β-t az adott algoritmus határozza meg. Itt már y nem feltétlenül konvex kombinációként áll elő, de $x \geq 0$ az iteráció során fenntartjuk.

Ezen az algoritmusoknak a futásidőbecsléséhez hasznos a következő Goffin [3] által bevezettet jelölés.

$$\rho_A := \max_y \min_i \hat{a}_i^T \hat{y}$$

Ahol $\hat{a}_i$ az A mátrix i-edik oszlopvektora normálva.
ρ_A az origó és $conv(\hat{A})$ határának távolsága. $\rho_A < 0$ pontosan akkor, ha az origó $conv(\hat{A})$ belsejében van. $\rho_A > 0$ pontosan akkor, ha az origó $conv(\hat{A})$ -n kívül van. Ahol $\hat{A}$ azt jelöli, hogy A oszlopait normáltuk.

A Neumann-féle iterációban a fenti paraméterek a következők. $\alpha, \beta > 0$ és $\alpha + \beta = 1$ mellett válasszuk úgy a paramétereket, hogy az új y hossza minimális legyen.
Belátható, hogy $\rho_A < 0$ esetben $\|y\|$ exponenciálisan konvergál a 0-hoz (Nem biztos, hogy véges sok lépésben megáll), $\rho_A > 0$ esetben pedig $1/\rho_A^2$ lépésben megáll.
Az első pár lépésben az y hossza gyorsan csökken, ezért használható arra, hogy a belső pontos módszerekhez egy jó kezdőpontot kapjunk.
Többféle javítása is ismert az algoritmusnak, például J. Gonçalvesa, R. Storerb és J. Gondzioc [4] megmutatták, hogy jelentős hatékonyság növekedés érhető el, ha nem csak az y-nal nagy szöget bezáró oszlophoz tartozó együtthatót növeljük, hanem a vele kis szöget bezáróét csökkentjük.
D. Li, K. Roos és T. Terlaky [5] adott a Neumann módszerre alapuló polinomiális futásidejű algoritmust, a 4. fejezetben leírt algoritmushoz által is használt "oszlop skálázás" segítségével.

Több más paraméter választási szabályt is szoktak használni. A 5. fejezetben leírt algoritmus például Dunagan-Vempala lépéseket használ. Itt $\alpha = 1$, $\beta = -(\hat{a}_k^T y)$.
Ennek a paraméter választásnak előnye, hogy x koordinátánként nem csökken, és, ha $\rho_A < 0$, akkor $\|y\|$ minden lépésben $\sqrt{1 - \rho_A^2}$ szorosára csökken,

A projekciós algoritmusok esetében hasonló elvet használunk a $Px > 0$ illetve a $\Pi x = 0, x \geq 0$ megoldására,
ahol P illetve Π projekció az $\{Ax = 0\}$-ra illetve ennek az ortogonális kiegészítőjére.

A *Szimplex módszer* volt az első algoritmus, amit az LP feladat algoritmukus megoldására hatékonyan fel lehetett használni. 1947-ben vezette be Dantzig. Gyakorlatban annyira hatékony, hogy máig is sok LP-solver alapját képezi. Átlagos futásideje polinomiális. Az algoritmus bemenete egy poliédert, egy célfüggvény, amit maximalizálunk a futás során, és egy csúcsa a poliédernek, amelyből indítjuk az algoritmust.

A futás során minden lépésben egy olyan olyan új csúcsába lépünk a poliédernek, amiben a célfüggvény értéke nem kisebb, mint az aktuális csúcsban. Ennek a módszernek a legnagyobb előnye hogy nem használja ki a feladatban szereplő számok méretét. "Worst case" komplexitásra vonatkozóan 1972-ben Klee és Minty adott olyan példát amelyre futásideje exponenciális. Arra, hogy melyik csúcsot választjuk az egyes lépésben többféle szabályt javasoltak, de sajnos majdnem mindegyik szabályhoz tudunk példát ami bizonyítja az exponenciális futásidőt.

Először 1979-ben Khachiyan-nak sikerült bizonyítania, hogy az LP feladat polinomiális futásidőben megoldható. Ez az eredménynek nagy elméleti jelentősége volt, mert előtte fontos nyitott kérdés volt a matematikában.
Az általa bevezetett *Ellipszoid módszer* eldönti egy poliéderről, hogy üres-e, és ha nem, megadja egy pontját. Az algoritmus alapgondolata a következő.
Kiindulunk egy olyan ellipszoidból, ami tartalmazza a poliédert. Ellenőrizzük, hogy az ellipszoid középpontja benne van-e a poliéderben, ha nincs, akkor veszünk egy szeparáló hipersíkot, azaz egy olyan hipersíkot, hogy az általa meghatározott egyik féltér tartalmazza a poliédert, és a másik az ellipszoid középpontját. A szeparáló hipersík segítségével konstruálunk egy új ellipszoidot, aminek a térfogata kisebb, és továbbra is tartalmazza a poliédert. Kihasználva a poliéder minimális térfogatára adott alsó becslést, azt kapjuk, hogy megfelelően sok iteráció után találunk egy megoldást, vagy kiderül, hogy a poliéder üres.
Nevezzük szeparációs orákulumnak azt a szubrutint, ami eldönti, hogy egy pont megengedett megoldás-e, és ha nem akkor ad egy fent leírt szeparáló hipersíkot. Az ellipszoid módszerre támaszkodva Grötschel-Lovász-Schrijver igen fontos elméleti eredménye, hogy ha adott egy szeparációs orákulum, akkor azt felhasználva polinomiális futásidőben meg tudjuk adni az orákulum által meghatározott poliéder egy pontját, azaz a szeparáció "ekvivalens" az optimalizációval.
Az algoritmus gyakorlatban nagyon lassúnak bizonyult. Ez felveti a kérdést, hogy mennyi jelentősége van a polinomialitásnak a gyakorlati hatékonyág szempontjából

1984-ben Karmarkar mutatott új típusú polinomiális LP algoritmust. A *Belső pontos módszer* a szimplex módszerhez hasonlóan egy lineáris célfüggvény értéket maximalizálja. Itt egy a poliéder belsejében lévő pontból indulunk ki, és a Szimplexel ellentétben a poliéder belső pontjain lépkedünk. A módszer futásidőbecslése polinomiális, lényegesen jobb mint az ellipszoid módszeré. Gyakorlati hatékonysága gyakran eléri a szimplex módszerét.

3 Jelölések és segédállítások

Jelölések:

Megegyezés szerint a latin nagy betűk jelöljenek mátrixokat, a kisbetűk oszlopvektorokat, a görög betűk skalármennyiségeket. Továbbiakban $A \in \mathbb{Z}^{m \times n}$ mátrix, $m \leq n$

$e = \{1, 1, ..., 1\}$

I az egység mátrix

$v^{(i)}$ a v vektor i-edik koordinátája e_i az i-edik egységvektor

a_i az A mátrix i-edik oszlopvektora

$(A|b)$ az A és b egymás mellé írásával kapott mátrix

$\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ a megfelelő méretű A, B, C, D mátrixok egymáshoz ragasztásával létrejött mátrix.

$\hat{v} = \frac{v}{\|v\|}$ azaz a vektor normáljta.

$\hat{A} = (\hat{a}_1|\hat{a}_2|..|\hat{a}_n)$

$V(H)$: H térfogata

$conv(A)$: A oszlopainak a konvex burka

V_m : az m dimenziós egységgömb térfogata

$B^{-1}_{ij} = \frac{det(B^*_{ji})}{det(B)}$ azaz az inverz mátrix ij eleme

$P_{A,b} \overset{\text{def}}{=} \{x \mid Ax = b, x \geq 0\}$

Ebben a fejezetben fölépítem a szükséges fogalmakat, és tételeket.

Első sorban az LP feladat különböző alakjai közti összefüggéseket mutatom meg.

3.1. Az LP feladat definíciója

Ahhoz, hogy definiálni tudjuk az LP feladatot, érdemes rögzítenünk a poliéder fogalmát.

$$Ax \leq b \qquad Ax \leq b, x \geq 0 \qquad Ax = b, x \geq 0$$

Ez csak pár a gyakran használt alakok közül. Könnyű látni, hogy a különböző alakok átalakíthatóak egymásba, úgy, hogy a feladat mérete csak konstansszorosára nő. A szakdolgozatban az $Ax = b, x \geq 0$ alakot fogom használni.

A továbbiakban, tételezzük fel, hogy A sorai függetlenek.
Ezt azért tehetjük meg, mert ha nem függetlenek, akkor vagy nincs megoldása $Ax = b$ -nek, vagy, ha nem, akkor hagyjunk el a többitől függő sorokat, amíg a sorok függetlenek nem lesznek. A kapott A' és b' -re $P_{A',b'} = P_{A,b}$
A' -t Gauss eliminációval állíthatjuk elő.

Bár rögzítettük, hogy mit nevezünk poliédernek, továbbra is többféle definícióját adhatjuk az LP feladatnak. Ezekről látni fogjuk, hogy ha az egyikre adunk egy algoritmust, akkor annak polinom sok hívásával meg tudjuk oldani a többit is.

3.1. Definíció (Eldöntési feladat)**.**
Adott A,b mellett a feladatunk, hogy eldöntsük, hogy $P_{A,b}$ üres-e.
Ezt a feladatot továbbiakban jelölje LP_1.

3.2. Definíció (Poliéderbeli pont keresése)**.**
Adott A,b mellett eldöntjük hogy $P_{A,b}$ üres-e, és ha nem, akkor megadjuk egy pontját
Ezt a feladatot továbbiakban jelölje LP_2.

3.3. Definíció (Optimalizálási feladat)**.**
Adott A,b,c mellett eldöntjük, hogy $P_{A,b}$ üres-e, és ha nem, akkor megadjuk egy olyan x^* pontját, hogy $cx^* = \max\{cx | x \in P_{A,b}\}$. Azaz egy adott lineáris célfüggvény szerinti maximumot keressük a poliéderben.
Ezt a feladatot továbbiakban jelölje LP_3.

Gyakran az optimalizálási feladatra adott algoritmusok egy bemenetként kapott poliéderbeli pontból kiindulva oldják meg a feladatot. Ez valójában nem jelent plusz feltételt, ilyen esetben a következőt tehetjük: $A' := \begin{pmatrix} A & Ib \end{pmatrix}, c' := (c, -\infty, ..., -\infty)$
Ennek a feladatnak ugyanazok a optimális megoldásai, és pontosan akkor van nem $-\infty$ értékű megoldása, ha az eredeti feladatnak van megoldása.

Fontos lesz számunkra, hogy mikor mondjuk azt egy LP algoritmusra hogy polinomiális.

3.4. Definíció (Polinomiális LP algoritmus)**.**
Polinomiális LP algoritmus alatt azt értjük, aminek a lépésszáma felülről becsülhető a bemenet méretének valamilyen polinomjával, és a fenti feladatok valamelyikét oldja meg.

Az LP algoritmus bemenete A, b (c). Ennek mérete, azaz $n * (m + 1) * b$ ahol b a bemenetben szereplő számok mérete, azaz ha A-ról és b-ről feltesszük, hogy egész értékűek, akkor b-re az igaz, hogy$-2^{b-1} \leq (A)_{ij} \leq 2^{b-1}$
A továbbiakban $L = mnb$ jelöli az input méretét.

3.2. A különböző LP alakok közti kapcsolatok

Az általunk tárgyalt algoritmusok a egy speciális alakját használják az LP feladatnak. Ahhoz, hogy az általános LP feladat megoldására is használható módszert kapjunk, visszavezetéseket kell megadnunk. A következő tételek mutatják meg, hogy, hogyan lehet az egyes alakokra adott algoritmusok felhasználni egy általánosabb feladat megoldásához.

3.5. Lemma. Ha LP_1 megoldható polinomiális futásidőben akkor LP_2 is

Bizonyítás. Amíg A-nak legalább $m+1$ oszlopa van, hagyjunk ki egy oszlopot A-ból, futtassuk azLP_1-et megoldó algoritmust. Ha így is létezik megoldás akkor ezt az oszlopot végleg töröljük, mert nincs rá szükségünk a megoldáshoz. Ha viszont az oszlop kihagyásával már nincs megoldás, akkor tudjuk hogy ez az oszlop minden bázismegoldásban nemnulla együtthatóval szerepel. Ekkor ezt az oszlopot bennehagyjuk A-ban.
Ha létezik megoldás, akkor létezik megengedett bázis (ez m oszlopot tartalmaz), ezért amíg legalább $m+1$ oszlop van addíg van olyan oszlop ami nincs benne mindegyik bázismegoldásban. Tehát az algoritmus legfeljebb n lépésben lecsökkenti az oszlopok számát m-re. A megmaradó m oszlop a B bázist alkotja, ekkor $x = B^{-1}b$ bázismegoldás.

□

A Következő visszavezetéshez szükségünk lesz a dualitás tétel következő alakjára.

3.6. Tétel (Dualitás tétel)**.**
Ha $P_{A,b}$ *nem üres, akkor* $\max\{c^T x | x \in P_{A,b}\} = \min\{y^T b | y^T A \geq c^T\}$

3.7. Lemma. Ha LP_2 megoldható polinomiális futásidőben akkor LP_3 is

Bizonyítás. Legyen LP_2-t megoldó algoritmus bemenete:

$$A = \begin{pmatrix} A & 0 & 0 & 0 \\ 0 & A^T & -A^T & -I \\ c^T & b^T & -b^T & 0 \end{pmatrix}, b = \begin{pmatrix} b \\ c \\ 0 \end{pmatrix}$$

Legyen kimenetként visszaadott vektor: $(x, y1, y2, z)$, $y := y1 - y2$
Ekkor $x \in P_{A,b}$, $y^T A \geq c^T$, $c^T x = y^T b$
Tehát Dualitás tétel miatt $c^T x$ maximális Ha nem kapunk megoldást, akkor $P_{A,b}$ üres, vagy $c^T x$ nem korlátos. □

Az általunk tárgyalt algoritmusok a

$$\exists x : Ax = 0, x > 0 \tag{1}$$

rendszert fogják tudni leghatékonyabban kezelni. Ennek a geomatriai jelentése a következő. Pontosan akkor létezik megoldása (1)-nek, ha az origó az A oszlopaiból álló halmaz konvex burkának belsejében van.
A következőkben megmutatjuk, hogy ha (1)-eldöntésére tudunk polinomiális algoritmust adni, akkor azt felhasználva meg tudjuk oldani az LP-feladatot.

Először adjuk (1) meg farkas duálisát.
Kihasználva, hogy $\exists x$ amire $Ax = 0, x > 0$ pontosan akkor, ha $\exists x$ amire $Ax = 0, x \geq e$ kapjuk, hogy ennek a feladatnak a farkas duálisa $\exists y : y^T A \geq 0, y \neq 0$
hasonlóan $\exists x : Ax = 0, x \geq 0, x^{(i)} > 0$ duálisa $\exists y : y^T A \geq 0, y^T a_i > 0$

Vegyük észre, hogy

3.8. Lemma. Ha van polinomiális algoritmusunk (1) eldöntésére akkor van polinomiális algoritmusunk

$$\exists x : Ax = 0, x \geq 0, x \neq 0 \tag{2}$$

eldöntésére is.

Bizonyítás. Ha el tudjuk dönteni (1)-et akkor mivel $A = \begin{pmatrix} A^T & -A^T & I \end{pmatrix}$ esetén a duális $\nexists y : y^T A^T = 0, y \geq 0, y \neq 0$ ezért el tudjuk dönteni $\exists x : Ax = 0, x \geq 0, x \neq 0$ sőt 3.5 miatt tudunk találni is ilyen x-et.

□

Jelöljük (0)-val az eredeti eldöntési feladatot, azaz $\exists x : Ax = b$, $x \geq 0$ eldöntését

3.9. Lemma. Tegyük fel, hogy Alg3 megoldja (2)-t és ha létezik, meg is adja egy megoldását. Ekkor ennek felhasználásával készíthetünk polinomiális LP algoritmust, ami megoldja (0)-t.

Bizonyítás. Vegyük észre, hogy (0) pontosan akkor megoldható ha $(A|-b)x = 0, x \geq 0$ -nak van olyan megoldása, ahol a $-b$ -hez tartozó koordinátában x pozitív. Továbbiakban ezt a koordinátát $x^{(n+1)}$ jelöli.
Tehát a kovetkező képpen kapunk (0)-t megoldó algoritmust:
$A \leftarrow (A| - b)$
Iteráljuk a következőt:

Alg3-at futtatjuk A-ra.
(I) Ha nem talál megoldást, akkor (0)-nak sincs.
(II) Ha olyan megoldást ad, $x^{(n+1)}$ pozitív, kaptunk egy megoldást (0)-ra
Ha nem ilyen megoldást kapunk, akkor $F := \{i|x^{(i)} > 0\}$, $G := \{i|x^{(i)} = 0\}$
Alkalmazzunk sor eliminációt az F-beli oszlopokon. Így állítjuk elő a következő mátrixot: $\begin{pmatrix} I^* & B \\ 0 & A' \end{pmatrix}$ Ahol $\begin{pmatrix} I^* \\ 0 \end{pmatrix}$ felel meg az F-beli oszlopoknak, és $I^* = \begin{pmatrix} I & S \end{pmatrix}$, S tetszőleges, akár üres is lehet.
$A \leftarrow A'$

Ez az algoritmus legfeljebb n iterációban biztosan lefut. Igazolnunk kell még, hogy
(a): néhány iteráció után miért igaz (I), és hogy
(b): hogyan találunk megoldást (II)-ben.
Indukció miatt elég egy iterációt néznünk.
(a): Ha x olyan megoldás, hogy $x^{(n+1)}$ pozitív, akkor ezt az x -et G-re megszorítva $A'x = 0, x \geq 0$ nak találjuk meg egy ugyanilyen megoldását
(b): Legyen x' olyan megoldása $A'x' = 0, x' \geq 0$ -ak, hogy $x'^{(n+1)}$ pozitív. Legyen z olyan, hogy $\begin{pmatrix} I^* \\ 0 \end{pmatrix} z + \begin{pmatrix} B \\ A' \end{pmatrix} x' = 0$ Ilyet tudunk találni, mert I^* -nak része az identitás. (z, x') jó megoldás lenne, de z nek lehetnek negatív koordinátái.
A-ról A'-re történő áttéréskor találtunk egy v vektort, amire $Av = 0, v \geq 0$ és v F-hez tartozó koordinátái szigorúan pozitívak. kellően nagy λ-ra: $x = (z, x') + \lambda v$ nemnegatív. Ez az x olyan megoldása $Ax = 0, x \leq 0$ -nak amire $x^{(n+1)}$ pozitív.

□

Az eddieket felhasználva kapjuk a következő tételt:

3.10. Tétel. Ha létezik algoritmus ami eldönti (1)-et Polinom időben, akkor $LP \in \mathbf{P}$

A következő fogalom megadja az összefüggést (0) és (2) között:

3.11. Definíció (tartó)**.** (2)-t tekintve azt mondjuk, hogy $i \in Supp$, azaz i eleme a tartónak, ha $\exists x : Ax = 0, x \geq 0, x^{(i)} > 0$
Tehát (0) pontosan akkor megoldható, ha $A = (A|-b)$ esetén $n+1$ eleme a tartónak. Egy x megoldást maximális tartójú megoldásnak hívunk, ha minden $i \in Supp$ esetén $x^{(i)} > 0$, az ilyen tulajdonságú megoldás keresését nevezzük maximális tartó feladatnak.

A 4. fejezetben tárgyalt lemmákhoz szükségünk lesz a következő két fogalomra.

3.12. Definíció (bázis és bázismegoldás)**.**
x bázis megoldás pontosan akkor, ha $x \in P_{A,b}$ és $x = B^{-1}b$, ahol B négyzetes részmátrixa A-nak. Ekkor B-t megengedett bázisnak hívjuk.

3.3. Felhasznált lineáris algebra

Mindkét tárgyalt algoritmushoz szükségünk lesz arra, hogy hogyan kell $\{p \mid Ap = 0\}$ lineáris altérre vetíteni.

3.13. Lemma. *Tetszőleges x pont $H = \{p \mid Ap = 0\}$ lineáris altérre vetített képe* $x - A^T(AA^T)^{-1}Ax$
megjegyzés: $(AA^T)^{-1}$ létezik, mert A-ról föltettük, hogy a sorai függetlenek.
Kihasználjuk, hogy $H^\perp = \{v \mid \exists y : A^T y = v\}$ Legyen x H-ra vetített képe z. Tehát $Az = 0$.
Továbbá $x - z$ merőleges H-ra tehát $\exists y : A^T y = x - z$ ezekből:

$$A(x - A^T y) = 0$$

$$Ax = AA^T y$$

$$(AA^T)^{-1}Ax = y$$

$$A^T(AA^T)^{-1}Ax = x - z$$

$$z = x - A^T(AA^T)^{-1}Ax$$

A későbbiekben a determinánsok becsléséhez a következő összefüggést fogjuk többször is felhasználni bizonyítás nélkül.

3.14. Lemma. Legyen $B = (b_1|b_2|..|b_n) \in \mathbb{R}^{n\times n}$
Ekkor $|det(B)| \leq \prod \|b_i\|$

4 Chubanov algoritmusa

Ebben a fejezetben leírom Chubanov 2013-ban publikált Polinomiális LP alogritmusát [1]. Ez az algoritmus projekciós típusú, azaz a $Py > 0$ feladatot oldja meg, ahol P a $\{x \mid Ax = 0\}$-ra való vetítés mátrixa. Ezzel megoldja (1)-et. Előnye, hogy a maximális tartó feladat megoldásához annyit kell tenni, hogy kihagyjuk a mátrixból azokat az oszlopokat, amikre tudjuk, hogy nincsenek benne a szupportban. Így az LP feladatot megoldó változat futásidő becslése $O(n^4L)$ lesz, lényegesen jobb mint a 5. fejezetben leírt algoritmusé.
Legyen $P = I - A^T(AA^T)^{-1}A$. Vegyük észre, hogy ez a mátrix szimmetrikus. P egy vetítés mátrixa, tehát a normája 1, ebből az is következik hogy az oszlopai legfeljebb 1 hosszúak.
Vegyük észre, hogy ha találunk y-t hogy $Py > 0$, akkor Py megoldása (1)-nek.
Az algoritmus működési elve a következő:
Fenntartunk egy nemnegatív y vektort, úgy, hogy $e^Ty = 1$, és törekszünk, hogy $Py > 0$ teljesüljön, illetve hogy $\|Py\|$ csökkenjen. Ha $\|Py\|$ kellően kicsi, akkor arra tudunk következtetni, hogy létezik k, hogy minden $x \in [0,1]^n$ megoldásra $x^{(k)} \leq \frac{1}{2}$. Ekkor a k-adik oszlopot megszorozzuk 2-vel. Néhány ilyen lépés után az $x \in [0,1]^n$ megoldásokra $x^{(k)} \leq (\frac{1}{2})^t$. Ekkor tudjuk (ha t kellően nagy), hogy minden x megoldásra $x^{(k)} = 0$.

4.1. Felhasznált lemmák

A következő lemmák adják az előző állításhoz szükséges eszközt.

4.1. Lemma.
Tekinstük az $Ax = 0$, $0 \leq x \leq 1$ rendszert.
Ha létezik x megoldás, hogy $x^{(k)} \geq K$, akkor létezik x bázismegoldás, hogy $x^{(k)} \geq K$

Bizonyítás. Tetszőleges c-re: ha c^Tx korlátos, akkor c^Tx maximuma felvétetik bázismegoldáson.
Ezt a tételt $c = e_k$ ra alkalmazzva és kihasználva, hogy a poliéder korlátos, kapjuk a lemmát. □

4.2. Lemma.
Tekintsük $Ax = b$, $x \geq 0$ redszert
Ekkor létezik olyan $\varepsilon > 0$, hogy minden x bázismegoldásra $|x^{(i)}| < \varepsilon$ esetén $x^{(i)} = 0$

Bizonyítás. Legyen x bázismegoldás.

$$x = B^{-1}b \Rightarrow x^{(i)} = \frac{p}{det(B)}$$

Ahol p egész. □

ε=min $1/|det(B)|$ a feladat méretében polinomiális méretű.

4.3. Lemma.
Létezik $\varepsilon > 0$, hogy, ha minden $x \in [0,1]^n$ megoldásra $x^{(i)} < \varepsilon$ akkor minden x megoldásra $x^{(i)} = 0$

Bizonyítás. Írjuk fel az $Ax = 0$, $0 \leq x \leq 1$ rendszert $Ax = b$, $x \geq 0$ alakban. Legyen ε mint (4.2)-ben. Ekkor minden x megoldásra : $x^{(i)} < \varepsilon \Rightarrow$
minden x bázismegoldásra : $x^{(i)} < \varepsilon \quad (4.2) \Rightarrow$
minden x bázismegoldásra : $x^{(i)} = 0 \quad (4.1) \Rightarrow$
minden x megoldás : $x^{(i)} = 0$
Ha az $x \in [0,1]^n$ esetre tudjuk, hogy $x^{(i)} = 0$, akkor tudjuk tetszőleges x-re is tudjuk, mert a feladat homogén. □

4.2. Az alap eljárás

A következő szubrutin fogja az algoritmus alapját képezni:

Algorithm alap eljárás

INPUT: A, $y^{in} : y^{in} \geq 0$, $e^T y^{in} = 1$
$y := y^{in}$
$P := I - A^T(AA^T)^{-1}A, x := Py$
while $\max_i y^{(i)} < 2e^T[x]^+$ **do**
  if x>0 **then**
    return (a):x
  else
    k olyan, hogy $x^{(k)} \leq 0$
    $y := \alpha y + (1-\alpha)e_k$
    $x := \alpha x + (1-\alpha)p_k$
    if x=0 **then**
      return (b):y
return (c):y

Minden lépésben választunk k-t hogy $x^{(k)} < 0$, ekkor $p_k^T x \leq 0$, mert $Px = x$. A következő x-et ezen p_k és x konvex kombinációjaként kapjuk. Azaz α-t úgy választjuk, hogy $\|x'\|$ minimális legyen. Tehát

$$\alpha = \frac{p_k^T(p_k - x)}{\|p_k - x\|^2}$$

Az algoritmus 3 féle képpen állhat le.
(a): $x = Py$, $x > 0$ Tehát visszaadja egy megoldását (1)-nek.
(b): Talál y-t, hogy $Py = 0$. Ekkor $z = yA^T(AA^T)^{-1}$ duális megoldás. Mert $zA = y \geq 0$, $y \neq 0 \Rightarrow z \neq 0$
(c): visszaad egy y-t amire létezik k, hogy $y^{(k)} \geq 2e^T[x]^+$ Erre a k-ra teljesül, hogy $\forall x \in [0,1]^n$ megoldásra : $x^{(k)} \leq \frac{1}{2}$

$[v]^+$ jelölje azt a vektort, amiben v negatív koordinátáit 0-val helyettesítettük Nevezzük a $\max_i y^{(i)} < 2e^T[x]^+$ feltételt (mf)-nek, mint megállási feltétel

Bizonyítás. (mf) miatt a kapott x, y-ra $(*)$: $y^{(k)} \geq 2e^T[x]^+$
Vegyük $\max e_k^T x \mid Ax = 0,\ 0 \leq x \leq 1$ Ennek a duálisa. $\min z^T e \mid A^T y + z \leq e_k$, $z \geq 0$
Legyen

$$y = \frac{yA(AA^T)^{-1}}{y^{(k)}} \quad yA = \frac{y(I-P)}{y^{(k)}} \quad z = \frac{[yP]^+}{y^{(k)}} = \frac{[x]^+}{y^{(k)}}$$

Tehát (y, z) duál megoldás, $(*)$ miatt $z^T e \leq \frac{1}{2}$ □

Később ezt arra fogjuk használni, hogy a_k-t elosztjuk 2-vel, így $[0,1]^n$ beli megoldásokat továbbra is a $[0,1]^n$ be képezzük

Technikai okok miatt, tekintsünk úgy az algoritmusra, hogy ugyan az utolsó y bizonyítja a megfelelő k létezését, de mi az utolsó előtti iterációban kiszámolt y vektorral szeretnénk tovább számolni, tehát a továbbiakban y^{out} jelölje azt az utolsó y-t, amire még teljesül (mf), és vegyük úgy, hogy ezt az y-t adja vissza az algoritmus. Ha (mf) már y^{in}-re se teljesül akkor $y^{out} = 0$
Az LP megoldó algoritmus során az utolsó ilyen tulajdonsággal rendelkező y-t $\bar{y}$ fogja jelölni.

4.4. Lemma. Az alap algoritmus által végzett iterációk száma legfeljebb

$$\frac{1}{\|Py^{out}\|^2} - \frac{1}{\|Py^{in}\|^2} + 1$$

Bizonyítás. Egy iterációban legyen $x' = \alpha x + (1-\alpha)p_k$
x' választása miatt $x' \perp p - x$
Legyen $\bar{x} = [p, x] \cap \{x \mid p^T x = 0\}$ ahol $[p, x]$ a p x szakasz. Ez a metszéspont létezik, mert $px \leq 0$
Pithagoras tétel miatt $1/\|x'\|^2 = 1/\|\bar{x}\|^2 + 1/\|p\|^2$
$\|\bar{x}\| \leq \|x\|$ és $\|p\| \leq 1$ Tehát

$$\frac{1}{\|x'\|^2} \geq \frac{1}{\|x\|^2} + 1$$

□

Vegyük észre, hogy

$$\frac{1}{2n\sqrt{n}} \geq \|x\| \Rightarrow y^{(k)} \geq \frac{1}{n} \geq 2\|x\|\sqrt{n} \geq 2e^T[x]^+$$

$1/\|Py\|^2 = 1/\|x\|^2 \leq 4n^3$, tehát az eljárás legfeljebb $O(n^3)$ iterációt végezhet.
Az eljárás egy iterációja O(n) lépésben elvégezhető.

4.3. Az LP algoritmus

Jelölés: (4.3)-nak megfelelően legyen L_{min} az a feladat méretében polinomiális legkisebb érték, amire $x^{(i)} \leq (\frac{1}{2})^L_{min}$ esetén tudjuk, hogy $x^{(i)} = 0$

Az LP megoldó algoritmus a következő:

Algorithm LP algoritmus

INPUT: $Ax = 0$, $x > 0$

OUTPUT: $x > 0$ megoldás, vagy olyan k hogy $x^{(k)} = 0$ minden megoldásban

$t_i = 0$ ahol $i = 1..n$

$\bar{y}, y^{in} = e/n$

$M = I$

$D = I$

while $t_i < L_{min}$ minden $i = 1..n$ re **do**

Hívjuk meg az ALAP ELJÁRÁST A-val és y^{in}-nal

Ha x megoldással tér vissza, akkor **return** Mx megoldás.

Ha z duális megoldással tér vissza, akkor **return** z duális

megoldása az eredeti feladatnak.

Ha (c) esettel lép ki, akkor $a_k := a_k/2, m_k/2$

(azaz A és M k-adik oszlopát 2-vel osztjuk.

$t_k := t_k + 1$

if $y^{out} \neq 0$ **then**

$\bar{y} := y^{out}$

$D := I$

D k-adik oszlopát osszuk 2-vel

$y^{in} := \frac{D\bar{y}}{e^T D\bar{y}}$

return k amire $t_k \geq L_{min}$

Ha A_0 a bemeneti mátrix akkor az aktuális $A = A_0M$. Ebből rögtön látszik, hogy, ha $Ax = 0$, akkor $A_0Mx = 0$, azaz Mx megoldás.
Ugyan így, ha z duális megoldást kapunk, akkor $A^Tz \geq 0$, tehát $M^TA_0^Tz \geq 0$, azaz $A_0^Tz \geq 0$, tehát z duális megoldása az eredeti feladatnak. Ha megkeressük az olyan i-ket, hogy $a_i^Tz > 0$, akkor találunk olyan oszlopokat, amik nincsenek benne a bázisban.
Az algoritmus futása során mindíg megtartunk egy olyan y-t, amire (mf) teljesül. Ezt jelöli $\bar{y}$. Ennek függvényében számoljuk ki azt az y-t ami az alapeljárás inputja lesz. D az a mátrix ami tárolja a legutolsó $\bar{y}$ kiszámítása óta végzett felezési lépéseket.

4.4. Futásidő becslés

Jelölje P_A az aktuális A -hoz tartozó vetítés mátrixát.
Az iterációk számát $\|P_A y^{(in)}\|$ segítségével fogjuk tudni becsülni. Ehhez szükségünk lesz arra, hogy $y^{in} := \frac{D\bar{y}}{e^T D\bar{y}}$ lépésnél hogyan változik ez a mennyiség.

4.5. Lemma. Legyen D mint az algoritmusban, $y := \bar{y}$, legyen $A' = AD$, $y' = \frac{Dy}{e^T Dy}$, $N = |\{i|D_{ii} < 1\}|$, Ekkor

$$\frac{1}{\|P_A y\|^2} - \frac{1}{\|P_{A'} y'\|^2} \leq 8Nn^2$$

Bizonyítás. Legyen $z = A^T(AA^T)^{-1}Ay$. $P_{A'}z = 0$

$$\|P_{A'}D\| = \|P_{A'}(Dz - Dy)\| \leq \|(Dz - Dy)\| \leq z - y\| = \|P_A y\|$$
$$\frac{1}{\|P_{A'}y'\|} = \frac{e^T Dy}{\|P_{A'}Dy\|} \geq \frac{e^T Dy}{\|P_A y\|} \geq \Big(1 - \sum_{i:D_{ii}<1} y^{(i)}\Big)\frac{1}{\|P_A y\|}$$

mert

$$e^T Dy + \sum_{i:D_{ii}<1} y^{(i)} \geq 1$$

Tehát

$$\frac{1}{\|P_{A'}y'\|^2} \geq \Big(1 - \Big(2\sum_{i:D_{ii}<1} y^{(i)} - \Big(\sum_{i:D_{ii}<1} y^{(i)}\Big)^2\Big)\Big)\frac{1}{\|P_A y\|^2}$$
$$\frac{1}{\|P_A y\|^2} - \frac{1}{\|P_{A'}y'\|^2} \leq \Big(2\sum_{i:D_{ii}<1} y^{(i)} - \Big(\sum_{i:D_{ii}<1} y^{(i)}\Big)^2\Big)\frac{1}{\|P_A y\|^2} \leq \frac{4N\sqrt{n}}{\|P_A y\|} \leq 8Nn^2$$

Az utolsó lépésben kihasználtuk, hogy y teljesíti (mf)-et. Így:

$$y^{(i)} < 2e^T[P_A y]^+ \leq 2\|P_A y\|\sqrt{n}$$

□

Legyen $L'_{min} = nL_{min}$. Az LP algoritmus legfeljebb ennyi iterációt végez.

4.6. Tétel. Az LP algoritmus futtatása során az Alap Eljárás legfeljebb $O(n^3 + n^2 L'_{min})$ iterációt végez.

Bizonyítás. Legyen $r = 1/\|P_A y\|^2$ Kezdetben $r \geq 1$. Az alap eljárás minden iterációjában r legalább 1-gyel nő.
$y^{in} := \frac{D\bar{y}}{e^T D\bar{y}}$ lépésben (4.5)-ben leírtaknak megfelelően csökken. (itt számunkra azért y^{in} releváns, mert ez a vektor lesz az Alap Eljárás bemenete). N értéke az LP algoritmus minden iterációjában 1-gyel nőhet legfeljebb, így r is legfeljebb $8n^2$-tel csökkenhet. (mf) miatt tudjuk, hogy $r \leq 4n^3$. Tehát ha K az Alap eljárás által elvégzett iterációk száma, akkor:
$1 + K - 8n^2 L'_{min} \leq 4n^3$ □

Az Alap Eljárás egy iterációja elvégezhető $O(n)$ időben. P_A kiszámítása elvégezehető $O(n^3)$ időben
Tehát az LP algoritmus futásideje $O(n^4 + n^3 L'_{min})$

A maximum szupport feladatot megoldó algoritmus úgy kaphatjuk, hogy amikor kiderül egy k-ról , hogy $x^{(k)} = 0$, akkor a_k -t elhagyjuk A -ból, és újra meghívjuk a fenti algoritmust.
Így minden alkalommal egyel csökken A oszlopainak száma, tehát $O(n^4 L)$ futásidejű LP megoldót kapunk.

4.5. Kerekítés

Ahhoz, hogy ténylegesen biztosítani tudjuk a polinomiális futásidőt, az algoritmus során kerekítéseket fogunk használni, hogy korlátot tudjunk adni a számok méretére.

Ehhez kétféle kerekítést használunk: Minden lépésben kerekíteni fogjuk α-t, és minden n-edik lépésben kerekíteni fogjuk y-t, és újraszámoljuk x-et.

Jelöljük $\alpha^{\#}$-al a kerekítet α értéket.
Legyen $x' = \alpha x + (1-\alpha)p$, $x^{\#} = \alpha^{\#} x + (1-\alpha^{\#})p$
$\alpha^{\#}$-t úgy fogjuk megadni, hogy:

$$\frac{1}{\|x'\|^2} - \frac{1}{\|x^{\#}\|^2} \leq \frac{1}{2}$$

Tudjuk, hogy $1/\|x'\|^2 < 4n^3$, és megfelelő α mellett $1/\|x^{\#}\|^2 < 16n^3$. Ezt kihasználva:

$$\frac{1}{\|x'\|^2} - \frac{1}{\|x^{\#}\|^2} \leq (\|x'\| - \|x^{\#}\|)\Big(\frac{\|x'\| + \|x^{\#}\|}{\|x'\|^2 \|x^{\#}\|^2}\Big) < (\|x'\| - \|x^{\#}\|)128n^6$$

Tehát legyen

$$\alpha^{\#} = \frac{\lfloor 512n^6 \alpha \rfloor}{512n^6}$$

Ekkor $\|x'\| - \|x^{\#}\| < 256n^6$ tehát $\frac{1}{\|x'\|^2} - \frac{1}{\|x^{\#}\|^2} \leq \frac{1}{2}$

Kerekítés2: Legyen $y^{\#} = \mu \lfloor y/\mu \rfloor c$ ahol c konstanst úgy választjuk, hogy $e^T y^{\#} = 1$ teljesüljön. Könnyen látszik, hogy $|y^{\#} - y|^{(i)} \leq 3\mu$ ezt kihasználva:

$$|\|x^{\#}\| - \|x'\|| \leq \|x^{\#} - x'\| = \|P_A(y^{\#} - y')\| \leq 3n\mu$$

$\mu := 1/512n^7$ Ez biztosítja, hogy $\frac{1}{\|x'\|^2} - \frac{1}{\|x^\#\|^2} \leq \frac{1}{2}$

Tehát az algoritmus úgy módosul, hogy minden lépésben kiszámítjuk, $\alpha^\#$-t és y' helyett $y^\#$ lesz az új y. És minden n-edik lépésben kerekítés2-t is elvégezzük. Ezek a kerekítési lépések nem változtatnak a lépésszám becsléseken, Az iterációk számának a becslése nem változik, mert r legalább $1/2$-de növekszik minden lépésben.
Kerekítés2 elvégzése után közvetlenül y koordinátái kifejezhetők $1/\mu$ nevezőjű törtekkel. A Kerekítés2 utáni k adik lépésben y koordinátái kifejezhetők $(1/\mu)^k$ nevezőjű törtekkel. Minden n lépésben végzünk egy ilyen kerekítést, így a számok mérete a feladat méretében polinomiális.

5 Egy "Coordinate descent" típusú algoritmus

Ebben a fejezetben bemutatok egy "Coordinate descent" típusú algoritmust, ami polinom időben megoldja (1)-et. Az algoritmus D. Dadush, L. Végh, G. Zambellitől származik [2].
Az algoritmus Dunagan-Vempala lépéseket tesz, amíg $\|y\|$ bizonyos korlátnál nagyobb sebességgel csökken. Amikor nem tudunk kellő csökkenést elérni, az azt jelenti, hogy y irányban $conv(\hat{A}) \cap conv(-\hat{A})$ "lapos", ekkor skálázunk, azaz megnyújtjuk y irányban. Amikor $\|y\|$ elég kicsi, akkor egy lépésben meg tudjuk adni a feladat egy megoldását. Kihasználva minden lépésben vagy $conv(\hat{A}) \cap conv(-\hat{A})$ térfogata nő, vagy $\|y\|$ csökken kapunk becslést az iterációk számára.

Az algoritmusunk az összes ismert polinomiális algoritmushoz hasonlóan erősen kihasználja a feladatban szereplő számok méretét, így a futásidő függni fog L-től.

Az algoritmus leáll, ha $A^T y \geq 0$, ezt azért teheti meg, $\|y\| \neq 0 \Rightarrow y \neq 0$ azaz (1)-nek egy duális megoldását találtuk meg, azaz nincs megoldás. Megjegyzés: sajnos az algoritmus nem minden esetben ad duális megoldást, amikor nincs megoldás.

A skálázási lépésnél A oszlopait kell y irányban kétszeresére nyújtanunk, ehhez az $(I + \hat{y}\hat{y}^T)$ mátrixal szorozzuk meg jobbról. Könnyű ellenőrizni, hogy ez a mátrix valóban ennek a transzformációnak a mátrixa. (Vegyük észre, hogy ez a mátrix szimmetrikus)
A frissítési lépésnél $\alpha = 1$ feltétel mellett azt a β-t választjuk, amire $\|y'\|$ minimális

5.1. Az algoritmus

Paraméterek:

$$\varepsilon := \frac{1}{20m}, \quad N := 4mL, \quad \delta := \min_i \frac{1}{\|(AA^T)^{-1}a_i\|}$$

Algorithm

INPUT: A

OUTPUT: x ha (1)-nek van megoldása, vagy az állítás, hogy nincs megoldás

$x := e \quad y := Ax \quad t := 0$

while $\|y\| \geq \delta$ and $t \leq N$ **do**

  if $A^T y \geq 0$ **then**

    STOP: nincs megoldás

  else

    $k = \underset{i}{\text{argmax}}\ \hat{a}_i^T y$

    if $\hat{a}_k^T \hat{y} < -\varepsilon$ **then** "frissítés":

      $y := y - (\hat{a}_k^T y)\hat{a}_k, \quad x := x - (\hat{a}_k^T y)\frac{e_k}{\|a_k\|}$

    else "skálázás":

      $A := (I + \hat{y}\hat{y}^T)A$

      y:=2y

      t:=t+1

if $\|y\| < \delta$ **then**

  OUTPUT: $x := x - A^T(AA^T)^{-1}Ax$

else

  OUTPUT: nincs megoldás

Először bizonyítjuk, hogy az algoritmus által visszaadott x valóban megoldás.
Ehhez legyen az algoritmus során végzett nyújtások összege $\widetilde{T}$, azaz $\widetilde{T} = \prod(I + \hat{y}_i \hat{y}_i{}^T)$, azaz kilépéskor $A = \widetilde{T}A$. Az algoritmus által visszaadott x az $\{x \mid \widetilde{T}Ax = 0\}$ -re vetített vektor. $\widetilde{T}$ invertálható, azaz $\widetilde{T}Ax = 0 \Leftrightarrow Ax = 0$. Tehát már csak $x > 0$ -t kell ellenőriznünk.
$x_{\text{out}} = x - A^T(AA^T)^{-1}Ax$ ide az előbbi észrevétel miatt mindegy, hogy A-t, vagy $\widetilde{T}A$-t írunk
$\widetilde{T}AX = y$ Tehát $x - A^T(AA^T)^{-1}Ax = x - A^T(AA^T)^{-1}\widetilde{T}^{-1}y$
Könnyen látható, hogy $\|\widetilde{T}^{-1}\| \leq 1$
$x_{\text{out}}^{(i)} = x^{(i)} - a_i^T(AA^T)\widetilde{T}^{-1}y \geq x^{(i)} - \|a_i^T(AA^T)\|\|\widetilde{T}^{-1}y\| > 1 - \frac{1}{\delta}\delta > 0$ Itt, kihasználtuk, hogy $x^{(i)} > 0$. Ez következik abból, hogy x kezdetben e -vel egyenlő, és minden lépésben nemnegatív vektort adunk hozzá.

5.2. Szükséges lemmák

Jelöljük az indexben 0 -val a változók kezdeti értékeit
Többször fel fogjuk használni azt a tényt, hogy ha $v \in \mathbb{R}^m$ és $\forall i : |v^{(i)}| \leq l$, akkor $\|v\| \leq \sqrt{m}l$
A futásidő becsléshez a következő állításokra lesz szükségünk:

5.1. Lemma. $\|y_0\| \leq 2^L$

Bizonyítás.

$$\|y_0\| = \|Ae\| \leq \sum_{i=1}^{n} \|a_i\| \leq n\sqrt{m}2^b \leq 2^L$$

□

$P_A := conv(\hat{A}) \cap conv(-\hat{A})$

5.2. Lemma. $V(P_{A_0}) \geq 2^{-2mL}V_m$

Bizonyítás. Vegyük $conv(A)$ tetszőleges lapját, legyen p az origó erre a lapra vetített képe.
Állítás: $\|p\| \geq 2^{-L}$
Biz.: Legyen a lap amire vetítettünk $H = \{x|\exists y : By = x, e^T y = 1\}$ $B \subset A$ bázis, ekkor $H = \{x|e^T B^{-1}p = x\} = \{x|v^T p = x\}$ ahol $v^T = e^T B^{-1} \Rightarrow \|p\| = \frac{1}{\|v\|}$

$$\|v\| \leq \sum_{i=1}^{m} \|b_i\| \leq m\sqrt{m}\left|\frac{det(B_{ji}^*)}{det(B)}\right| \leq m^{\frac{3}{2}}\prod_{i=1}^{m}\|a_i\| \leq m^{\frac{3}{2}}(\sqrt{m}2^b)^m \leq 2^L$$

$|det(B)| \geq 1$, mert az elemei egészek, és invertálható
Ezzel az állítást igazoltuk.
Tehát $conv(A) \supset 2^{-L}V_m$ ugyanígy $conv(A) \supset 2^{-L}V_m$

$$(conv(\hat{A}) \cap conv(-\hat{A})) \supset (conv(A) \cap conv(-A))\frac{1}{\|a_k\|} \supset 2^{-2L}V_m$$

Tehát $V(P_{A_0}) \geq 2^{-2mL}V_m$ □

5.3. Lemma. $\delta \geq 2^{-L}$

Bizonyítás. Cauchy-Binet formulából:

$$(AA^T)^{-1}_{ij} = \left|\frac{det(AA^T)^*_{ji}}{det(AA^T)}\right| \leq \sum_{B\subset A} det(B)det(B^T) \leq \binom{n}{m}(\sqrt{m}2^b)^m$$

$\left|det(AA^T)\right| \geq 1$ mivel AA^T elemei egészek, és invertálható.

$$\|(AA^T)^{-1}a_k\| \leq \sqrt{m}2^b m\binom{n}{m}(\sqrt{m}2^b)^m \leq 2^L$$

□

$T := (I + \hat{y}\hat{y}^T)$
A "skálázási" lépéseknél P_A térfogata $\frac{3}{2}$ szeresére nő.
Megjegyzés: ha P_A -ban A oszlopai normálás nélkül szerepelnének, akkor a térfogat pont 2 szeresére nőne. Az állítás geometriai tartalma az, hogy P_A y irányban lapos volt, ezért a vektorok hossza a nyújtás során nem nő túl sokat, azaz a normálás hatása kicsi.

5.4. Lemma. $\hat{a}_k^T\hat{y} \geq -\varepsilon \Rightarrow V(P_{TA}) \geq \frac{3}{2}V(P_A)$

Bizonyítás. Legyen $TA = A'$
Állítás: $TP_A \subset (1 + 3\sqrt{3}\varepsilon)P_{A'}$ azaz $z \in P_A \Rightarrow Tz \in (1 + 3\sqrt{3}\varepsilon)P_{A'}$
$z := \sum \lambda_j \hat{a}_i$ ahol $\sum \lambda_i = 1, \lambda \geq 0$
$Tz = \sum \lambda_i T\hat{a}_i = \sum \lambda_i \|T\hat{a}_i\|\hat{a}'_i \overset{(*)}{=} \sum \lambda_i \sqrt{1 + 3(\hat{y}^T\hat{a}_i)^2}\hat{a}'_i$
$x_i := \hat{y}^T\hat{a}_i \quad \hat{a}_k^T\hat{y} \geq -\varepsilon \Rightarrow x_i \in [-\varepsilon, 1]$
$z \in P_A \Rightarrow |\sum \lambda_i x_i| = |\hat{y}^T z| < \varepsilon$
$0 \in P'_A$ miatt elég belátni, hogy: $\sum \lambda_j \sqrt{1 + 3x_i^2} \leq 1 + 3\sqrt{3}\varepsilon$

$$\sum \lambda_j \sqrt{1 + 3x_i^2} \leq \sum \lambda_i + \sqrt{3}\sum |\lambda_i x_i|$$

$$\sum |\lambda_i x_i| = \sum_{x_i>0} \lambda_i x_i + \sum_{x_i<0} |\lambda_i x_i|$$

$$|\sum \lambda_i x_i| = \sum_{x_i>0} \lambda_i x_i - \sum_{x_i<0} |\lambda_i x_i| < \varepsilon$$

$$\sum_{x_i>0} \lambda_i x_i + \sum_{x_i<0} |\lambda_i x_i| < \varepsilon + 2\sum_{x_i<0} |\lambda_i x_i| < \varepsilon + 2\sum \lambda_i \varepsilon < 3\varepsilon$$

$$\sum \lambda_j \sqrt{1 + (3x_i)^2} \leq 1 + 3\sqrt{3}\varepsilon$$

Ezzel megmutattuk, hogy $z \in conv(\hat{A}) \Rightarrow Tz \in (1 + 3\sqrt{3}\varepsilon)conv(\hat{A}')$
Ugyan így igazolható $Tz \in (1 + 3\sqrt{3}\varepsilon)conv(\hat{A}')$, tehát az állítást beláttuk.
Felhasználva az állítást, illetve hogy $det(T) = 2$

$$V(P_{A'}) \geq \frac{2}{(1 + 3\sqrt{3}\varepsilon)^m} V(P_A) \geq \frac{3}{2} V(P_A)$$

□

$(*)$: $T\hat{a}_i = \hat{a}_i + \hat{y}(\hat{y}^T\hat{a}_i) = \hat{a}_i - \hat{y}(\hat{y}^T\hat{a}_i) + 2\hat{y}(\hat{y}^T\hat{a}_i)$
$\hat{a}_i - \hat{y}(\hat{y}^T\hat{a}_i) \perp \hat{y}(\hat{y}^T\hat{a}_i)$
$\|T\hat{a}_i\|^2 = \|\hat{a}_i - \hat{y}(\hat{y}^T\hat{a}_i)\|^2 + 4\|\hat{y}(\hat{y}^T\hat{a}_i)\|^2 = \|\hat{a}_i\|^2 - \|\hat{y}(\hat{y}^T\hat{a}_i)\|^2 + 4\|\hat{y}(\hat{y}^T\hat{a}_i)\|^2 = 1 + 3(\hat{y}^T\hat{a}_i)^2$

A következő lemma biztosítja, hogy a frissítési lépésben $\|y\|$ kellő mértékben csökken.

5.5. Lemma. $\hat{a}_k^T\hat{y} < -\varepsilon \Rightarrow \|y'\| \leq \|y\|\sqrt{1 - \varepsilon^2}$

Bizonyítás. $y' = y - (\hat{a}_k^T y)\hat{a}_k$
$y' \perp (\hat{a}_k^T y)\hat{a}_k \Rightarrow \|y'\| = \sqrt{\|y\|^2 - \|(\hat{a}_k^T y)\hat{a}_k\|^2} = \|y\|\sqrt{1 - (\hat{a}_k^T\hat{y})} \leq \|y\|\sqrt{1 - \varepsilon^2}$ □

Futásidő becslés

Legyen N a skálázási lépések száma. P_A része az egységgömbnek, és térfogata minden skálázásnál legalább 3/2 szeresére nő ezért:
$V(P_{A_0})(\frac{3}{2})^N \leq V_m$
$(\frac{3}{2})^N \leq 2^{2mL}$
$N \leq 4mL$
Tehát $N = O(mL)$
Legyen K a frissítési lépések száma. $\|y\|^2$ minden skálázásnál 4-szeresére nő, minden frissítésnél $(1 - \varepsilon^2)$ szeresére csökken, nem csökkenhet δ^2 alá ezért:
$4^N\|y_0\|^2(1 - \varepsilon^2)^k \geq \delta^2$
$4^N 2^{4L}(1 - \varepsilon^2)^k \geq 1$
$c_1 N + c_2 L + Kln(1 - \varepsilon^2) \geq 0$

$\frac{c_1 N + c_2 L}{\varepsilon^2} \geq K$
Tehát $K = O(m^3 L)$
Ha az algoritmus futása közben fönntartjuk $A^T A$-t és $A^T y$-t akkor egy frissítést $O(n)$ időben, egy skálázást $O(n^2)$ időben tudunk elvégezni.
Így az algoritmus futásideje $O((m^3 n + mn^2)L)$
Azaz ennyi aritmetikai műveletet kell elvégeznünk.

Tehát az algoritmus polinom futásidőben időben megoldja (1)-et.
Így 3.8-at felhasználva kaphatunk egy bizonyítást $LP \in \mathbf{P}$-re.

Meggondolandó, hogy az algoritmus során milyen számábrázolást használunk. A precíz futásidő becsléshez, biztosítani kell tudnunk, hogy az egyes aritmetikai műveleteket polinomiális lépésszámban el tudjuk végezni. Például ha minden számot pontosan, azaz racionális számként ábrázolunk, akkor minden mműveletnél megkétszeretődik az ábrázoláshoz szükséges bitek száma. Ez elkerülhető például, ha megadjuk, hogy az alagoritmusban melyik lépésben milyen kerekítést kell elvégezni. Ilyen kerekítési eljárást, ehhez az algoritmushoz itt nem adunk meg.

5.3. Maximális tartó feladat

A fent leírt algoritmus egy változatanak felhasználásval meg tudjuk oldanni a maximum szupport problémát.
Használni fogjuk, hogy $i \in szupport$ duálisa $\exists y : A^T y \geq 0, a_i^T y > 0$
A következő változtatásokat kell tenni az algoritmuson
Paraméterek:

$$\varepsilon := \frac{1}{12m^{3/2}}, \quad N := 12m^2 L, \quad K = 2^{3mL} e^{N/4m}$$

δ : mint eddig
Legyen $\|a_i\| = \alpha_i$ azaz a bemeneti mátrixban az oszlopvektorok hosszai.

Algorithm

INPUT: A

OUTPUT: Maximális tartójú megoldás.

while $A \neq \emptyset$ **do**

Futtasuk a fenti algoritmust az új paraméterekkel

aszerint, hogy az algoritmus hogy tért vissza:

if visszaadott egy x megoldást **then**

OUTPUT: x

if visszaadott egy y -t **then**

$A \leftarrow A \setminus \{a_i \mid a_i^T y > 0\}$

if $y > \|\delta\|$ -val tért vissza **then**

$A \leftarrow A \setminus \{a_i \mid \|a_i\| > K\alpha_i\}$

OUTPUT: nincs megoldás

Megjegyzés, ahogy A oszlopait elhagyjuk, az algoritmus során fenn kell tartani, hogy a sorok függetlenek maradjanak.

5.6. Tétel. Ha az eredeti algoritmus N skálázást végzett (ahol N az új paraméter), Akkor $\|a_i\| > K\alpha_i \Rightarrow i \notin Supp$

Ennek a tételnek a bizonyítása hasonlóan megy mint 5.4 lemmáé
Az intuíció mögötte a következő:
Ha (1) -nek nincs megoldása, akkor $P_A \subset U$, ahol U valódi altere $\mathbb{R}^m$ -nek. Ebben az esetben $V(P_A)$ jelentse, ebben az altérben a térfogatot. Vegyünk egy v irányú nyújtást. Ha v vetülete U-ra nagy, akkor $V(P_A)$ jelentősen nő. Ha v közel merőleges az U -ra, akkor $V(P_A)$ legfeljebb kevéssel csökken. Ha a_i hossza a skálázsok során jelentősen megnőtt, akkor ez azt jelenti, hogy a skálázások sokszor estek a_i irányába. Ha $i \in szupport$ igaz lenne, akkor ebből következne, hogy a skálázó vektornak "átlagosan" nagy volt a vetülete U-ra, azaz $V(P_A)$ jelentősen megnőtt, itt kihasználva hogy P_A része az egységgömbnek ellentmodásra jutunk. Tehát $i \notin szupport$.

Ezalapján látható, hogy az új algoritmus minden lépésben elhagy egy oszlopot, így legfeljebb m lépésben megtalálja a maximális tartójú megoldást.

Ez az algoritmus lényegesen gyorsabb, mintha a 3.10-hez felhasznált algoritmusokat használnánk, az eredeti algoritmust, mint szubrutint alkalmazva. A futásidő becslése a maximum szupport feladat megodlására $O(M^6nL + M^2n^2L)$.

6 Implementáció

Ebben a fejezetben leírom az algoritmusok implementációja során szerzett tapasztalatokat. A két algoritmusról külön-külön, és összehasonlításban is összegzem az eredményeimet.

Mindekét esetben lebegőpontos értékeket használtam. Ez garantálja, hogy egy aritmetikai művelet egység időt vesz igénybe. Az alapértelmezett implementációját használtam a lebegőpontos számoknak, így fix a számok nagyságától is függő pontossággal számolunk. Közös tulajdonságuk, hogy az iterációk számára egy korlátot adunk, azaz előre megmondjuk, hogy hány iterációt fogunk végezni. Az elméleti elemzés során durva becsléseket adtunk ezen korlátok kiszámításához. Gyakorlatban a becslésekben szereplő determinánsokat akár ki számolhatjuk, én a legtöbb esetben az oszlopvektorok normáit használtam. Hátránya mindkét algoritmusnak, hogy minél jobb becslést tudunk adni az iterációk számára annál jobb lesz a gyakorlatban is a futásidő. (Ez általába nem jellemző az algoritmusokra).

6.1. Chubanov algoritmusa

Ennek az algoritmusnak jó tulajdonsága, hogy a mátrixban lévő számok nagysága is, és $x = Py$ is csökken, illetve ezutóbbi legfeljebb kismértékben nő, így a lebegőpontos értékek által biztosított pontosság elég a megbízható működéshez.
Bár előnye, hogy könnyebben tudja a maximum szupport feladatot kezelni, de a gyakorlati tapasztalat az, hogy sokkal nehezebb a duális megoldást előállítani ezen algoritmus segítségével. Ez az algoritmus szinte sosem lép ki duális megoldással, mert ehhez az kellene, hogy $x = 0$ előálljon.

6.2. Coordinate descent algoritmus

Ennek az algoritmusnak a fő gyengéje az, hogy még ha az elméleti elemzésben kapott becsléseken, a konkrét A mátrixot használva javítunk is, nagyon sok iteráció kell, hogy P_A értéke megnőjön eléggé ahhoz, hogy biztosan tudjuk, hogy nincs megoldás. Ez azért is okoz problémát, mert minden skálázási lépésben kérszeresére nő y hossza.

Ezen probléma elkerülésének egy módja, ha az algoritmus futása során A oszlopait, és y-t normáljuk, és tároljuk a hosszukat, sőt a hosszuk logaritmusát. Ezekre a hosszokra csak akkor van szükség ha a megoldást szeretnénk előállítani. Gyakorlati tapasztalat, hogy ha van megoldás akkor néhány skálázási lépés elég, és így nem kell a normák logaritmusát vennünk, azaz a kapott megoldás pontos lesz.
Ahhoz, hogy visszakapjuk a duális megoldást, el kell tárolni az irányokat, amikben skáláztunk. Ha nem kaptunk megoldást, ezekből az irányokból, és y ból konstruálhatjuk a duális megoldást. Ezt akkor is megtehetük, ha a ciklus $t > N$ n-el lépett ki. Ennek az algoritmusnak nagy előnye, hogy gyakran lép ki azzal, hogy talált duális megoldást, és néhány esetben akkor is valódi duális megoldást ad, ha P_A megnövekedése miatt lépett ki.

Ebben az algoritmusban ahhoz, hogy elérjük az elméleti futásidő becslést, fenn kell tartanunk az A^TA mátrixot és az A^Ty vektort. Ezeket a minden lépésben gyorsan tudjuk frissíteni. Az algoritmus helyes működése azon múlik, hogy ez a mátrix és vektor valóban konzisztens az aktuális A-val és y-nal, ennek a fenntartásához, és a numerikus hibák csökkentéséhez érdemes időnként, például minden n-edik skálázási lépésben újraszámolni az említett mátrixot, és vektort.

6.3. Összehasonlítás

Mindkét algoritmusnál azt tapasztaltam, hogy abban az esetben, amikor a feladatnak van megoldása, akkor néhány skálázási lépés elég. Ezekben az esetekben mindkét algoritmusnak jó a futásideje.
Mindkét algoritmusnak van előnye, és hátránya. Chubanov algoritmusa nem ad duális megoldást, de jók a numerikus tulajdonságai. A coordinate descent algoritmus néha pontatlan, de duális ettől az algoritmustól duális megoldást is remélhetünk.

Álljon itt egy táblázat a két algoritmus átlagos futásidejeivel.
$n/2 \times n$ méretű méretű mátrixok voltak az algoritmusok bemenetei. A mátrixokban szereplő számok -100-tól 100-ig egyenletes eloszlású véletlen egészek.
Minden méretre 10-szer futtattam az algoritmusokat. A jelölje chubanov algoritmusát, B a coordinate descent algoritmust.
Primál jelölje azoknak a futásoknak az átlagsebességét, amikor van megoldás, Duál azt amikor nincs.
Az értékeke másodpercben értendők. A 0 futásidő azt jeletni, hogy az algoritmus néhány iterációban lefutott, így a futásideje közel 0 másodperc.

A "-" azt jelenti, hogy a futás túl sok időt vett igénybe.

Futásidők				
n	Primál A	Primál B	Duál A	Duál B
10	0	0	0.130	1.469
20	0.005	0.015	1.495	19.84
30	0.005	0.023	8.923	107.7
40	0.05	0.421	32.75	385.9
50	0.285	0.223	94.41	859.5
60	0.201	0.241	230.2	-

A következő táblázatban olyan bemenetekre szerepel a futásidő, amiket úgy állítottam elő, hogy garantáltan legyen megoldás. Ezt könnyen elérhetjük, ha hozzáadunk egy új oszlopot A-hoz: $a_{(n+1)} = -Ax$. Én $x = (1, 1/2, .., 1/n)$-et használtam.

Futásidők		
n	Primál A	Primál B
200	0.359	0.687
400	2.984	4.962
600	10.56	15.18
800	23.82	34.29
1000	40.38	63.67
1200	70.75	106.1

Az algoritmusok C++-ban implementáltam, az Eigen lineáris algebra könyvtárat használtam fel a mátrix műveletek elvégzéséhez. A kódok a függelékekben olvashatók.

Irodalomjegyzék

[1] S. Chubanov, A polynomial projection algorithm for linear programming. (2013) http://www.optimization-online.org/DB_FILE/2013/07/3948.pdf

[2] D. Dadush, L. Végh, G. Zambelli, Rescaled coordinate descent methods for Linear Programming (2017) http://eprints.lse.ac.uk/84479/1/Vegh_Rescaled%20coordinate_2017.pdf

[3] J. Goffin, The Relaxation Method for Solving Systems of Linear Inequalities (1980) https://pubsonline.informs.org/doi/abs/10.1287/moor.5.3.388

[4] J. Gonçalvesa, R. Storerb és J. Gondzioc, A family of linear programming algorithms based on an algorithm by von Neumann (2009) https://www.tandfonline.com/doi/abs/10.1080/10556780902797236

[5] D. Li, K. Roos, T. Terlaky, A Polynomial Column-wise Rescaling von Neumann Algorithm (2015) http://www.optimization-online.org/DB_HTML/2015/06/4979.html

Függelékek

Chubanov Algoritmusa

```
class LPsolver
{
  public:
    int n,m,t,T,status;
    Eigen::VectorXd x,xlast,y,ylast,y_,ATy;
    Eigen::MatrixXd A,Pa;
    Eigen::MatrixXd Aori;
    bool step;
    double eps;

  LPsolver(Eigen::MatrixXd A)
  {
    status=0;
    this->A=A;
    Aori=A;
    m=A.rows(); n=A.cols();
    y.setConstant(n,1.0/n);
    y_=y;
    T=findT();
    Pa=Eigen::MatrixXd::Identity(n,n)
-A.transpose()*(A*A.transpose()).inverse()*A;
    x=Pa*y;
    eps=pow(10.0,-10);
  }

  int BP()
  {
    step=false;
    while(maxy() < 2.0*eTx())
    {

      int k=0;
      for(int i=0;i<n;i++)
      {
        if(x(i)<x(k)){k=i;}
      }
      if(x(k)>0){return 0;}
      double alpha=(Pa.col(k).transpose()).dot(Pa.col(k)-x)
/((Pa.col(k)-x).squaredNorm());
      ylast=y; xlast=x;
      y=alpha*y+(1-alpha)*Eigen::VectorXd::Unit(n,k);
      x=alpha*x+(1-alpha)*Pa.col(k);
      step=true;
      if(x.norm()<eps){return 1;}
    }
    return 2;
  }
```

```
int solve()
{
  t=0;
  int scale=0;
  std::vector<double> divs1(n),divs2(n);
  for(int i=0;i<n;i++)
  {
    divs1[i]=1.0;
    divs2[i]=1.0;
  }
  int st;
  while(t<T)
  {
    t++;
    st=BP();
    if(st==0){
      Eigen::VectorXd xout=x;
      for(int i=0;i<n;i++)
      {
        xout[i]*=divs1[i];
      }
      //return xout;
      return 1;
    }
    if(st==1)
    {
      Eigen::VectorXd zout=(A*A.transpose()).inverse()*A*y;
      //return zout;
      return -1;
    }
    if(st==2)
    {
      int k=0;
      for(int i=0;i<n;i++)
      {
        if(y(i)>y(k)){k=i;}
      }
      A.col(k)=A.col(k)/2.0;
      Pa=Eigen::MatrixXd::Identity(n,n)
-A.transpose()*(A*A.transpose()).inverse()*A;
      divs1[k]=divs1[k]/2.0;
      if(step)
      {
        y_=ylast;
        for(int i=0;i<n;i++){divs2[i]=1;}
      }
      divs2[k]=divs2[k]/2;
      for(int i=0;i<n;i++){y(i)=y_(i)*divs2[i];}
      Eigen::VectorXd e;
      e.setOnes(n);
      y=y/e.dot(y);
      x=Pa*y;
    }
  }
  return -2;
}
double findT()
```

```
  {
     double logLmin=0;
     for(int i=0;i<n;i++){logLmin+=log2(A.col(i).norm());}
     return n*logLmin;
  }
  double eTx()
  {
    double d=0;
    for(int i=0;i<n;i++){d+=std::max(0.0,x(i));}
    return d;
  }
  double maxy()
  {
     double ma=0;
     for(int i=0;i<n;i++)
     {
        if(y(i)>ma){ma=y(i);}
     }
     return ma;
  }
};
```

Coordinate descent algoritmus:

```
#include <iostream>
#include <Eigen/Dense>
#include <math.h>
#include <vector>
#include <list>

class LPsolver
{
  public:
    int n,m,status;
    double delta,logPa;
    long double ynorm;
    long double logynorm;
    Eigen::VectorXd x,y,ATy;
    Eigen::MatrixXd A,ATA;
    Eigen::MatrixXd Aori;
    std::vector<long double> Anorms;
    std::vector<long double> logAnorms;
    bool big;

    std::list<Eigen::VectorXd> dirs;

  LPsolver(Eigen::MatrixXd A)
  {
    status=0;
    this->A=A;
    Aori=A;
    m=A.rows(); n=A.cols();
    Anorms.resize(n);
    logAnorms.resize(n);
    for(int i=0;i<n;i++){Anorms[i]=A.col(i).norm();}
    for(int i=0;i<n;i++){logAnorms[i]=log(A.col(i).norm());}
    x.resize(n);
    x.setOnes();
```

```
    logPa=findPa();
    delta=finddelta();
    y=A*x;
    ynorm=y.norm();
    logynorm=log(y.norm());
    y.normalize();
    for(int i=0;i<n;i++){(this->A).col(i).normalize();}
    ATA=(this->A).transpose()*(this->A);
    ATy.resize(n);
    //for(int i=0;i<n;i++){ATy(i)=A.col(i).dot(y);}
    ATy=(this->A).transpose()*y;
    big=false;
  }

  int solve()
  {
    int t=0;
    int iter=0;

    while(logynorm>=log(delta) && logPa<=0)
    {
      iter++;
      if(!big){
        for(int i=0;i<n;i++)
        {
          if (Anorms[i]>pow(10,150))
          {
            big=true;
          }
        }
      }
      double mi=0;
      int k=0;
      for(int i=0;i<n;i++)
      {
        if(ATy(i)<mi)
        {
          mi=ATy(i);
          k=i;
        }
      }
      if(mi==0)
      {
        status=-1;

        Eigen::VectorXd Ty=y;
        for(auto v : dirs){Ty=(Eigen::MatrixXd::Identity(m,m)+
v*v.transpose())*Ty; Ty.normalize();}
        //return Ty
        return -1;
      }

      if(mi<-1.0/(11*m))
      {
        if(!big){
          x=x-(mi*(ynorm/Anorms[k]))*Eigen::VectorXd::Unit(n,k);
        } else {
```

```
        x=x-(mi*exp(logynorm-logAnorms[k]))*Eigen::VectorXd::Unit(n,k);
      }
      Eigen::VectorXd y_=y-mi*A.col(k);
      double len=y_.norm();
      ynorm*=len;
      logynorm+=log(len);
      y=y_.normalized();
      ATy=(ATy-mi*ATA.col(k))/len;
    }
    else
    {
      t++;
      A=(A+y*ATy.transpose());
      ynorm*=2;
      logynorm+=log(2);
      std::vector<double> norms(n);
      for(int i=0;i<n;i++){norms[i]=A.col(i).norm();}
      for(int i=0;i<n;i++){Anorms[i]*=norms[i];}
      for(int i=0;i<n;i++){logAnorms[i]+=log(norms[i]);}
      ATA=(ATA+3*ATy*ATy.transpose());
      for(int i=0;i<n;i++)
      {
        for(int j=0;j<n;j++)
        {
          ATA(i,j)=ATA(i,j)/(norms[i]*norms[j]);
        }
      }
      for(int i=0;i<n;i++){A.col(i).normalize();}
      ATy=2*ATy;
      for(int i=0;i<n;i++)
      {
        ATy(i)=ATy(i)/norms[i];
      }
      //pontositas
      if(t%(m*n)==0){
        ATA=A.transpose()*A;
        ATy=A.transpose()*y;
        if(!big){
          y.setZero();
          for(int i=0;i<n;i++){y+=A.col(i)*Anorms[i]*x(i);}
          ynorm=y.norm();
          logynorm=log(ynorm);
          y.normalize();
        }
      }
      logPa+=log(3.0/2.0) ;
      dirs.push_front(y);
    }
  }
  if(logynorm<log(delta))
  {
    status=1;
    for(int i=0;i<n;i++)
    {
      if(!big)
      {
        A.col(i)*=Anorms[i];
```

```
        } else {
          A.col(i)*=exp(logAnorms[i]);
        }
      }
      y*=ynorm;
      x=x-A.transpose()*(A*A.transpose()).inverse()*y;
      //return x;
      return 1;
    }
    else
    {
      status=-2;
      Eigen::VectorXd Ty=y;
      for(auto v : dirs){Ty=(Eigen::MatrixXd::Identity(m,m)+
v*v.transpose())*Ty; Ty.normalize();}
      //return Ty;
    }
    return -2;
  }
  double findPa()
  {
    std::vector<double> l(n);
    for(int i=0;i<n;i++){l[i]=A.col(i).norm();}
    std::sort(l.begin(), l.end(), std::greater<int>());
    double logPa=log(pow(m,1.5));
    for(int i=0;i<m;i++)
    {
      logPa+=l[i];
    }
    logPa+=l[0];
    return -m*logPa;
  }
  double finddelta()
  {
    Eigen::MatrixXd invAAT=(A*A.transpose()).inverse();
    double mi=0;
    for(int i=0;i<n;i++)
    {
      if((invAAT*A.col(i)).norm()>mi)
      {
        mi=(invAAT*A.col(i)).norm();
      }
    }
    return 1/mi;
  }
};
```