Z^o 行列によるＰ行列判別と線形相補性問題

東邦大学大学院　理学研究科　情報科学専攻
並木研究室　6513011　藤原　佳顕

平成 27 年 2 月 12 日

論 文 要 旨

数理計画問題の一つとして線形相補性問題 (LCP) と呼ばれる問題がある. この LCP は均衡問題やゲーム理論などの分野に応用されると共に, 線形計画問題や二次計画問題を特殊パターンとして持つような問題である. しかし LCP の扱いやすさは入力される行列とベクトルによって決定される. 例えば P 行列と呼ばれる行列クラスが入力されることで解の存在が保証され,P 行列の部分クラスである隠れミンコフスキー行列と呼ばれる行列が入力されると LCP は多項式時間で解けることがわかっている. 一方, 任意の行列が P 行列であることを調べる P 行列問題は,2^n 個の行列式計算が必要であるために co-NP 完全問題であることが知られており,P 行列問題には多項式時間の判別アルゴリズムは存在しないだろうという予想がなされている.

このような難しいとされる P 行列判別問題に対するアプローチの一つとして，P 行列クラスの上位クラス L と部分クラス U で挟み撃ちする手法が提案されている. 主に知られているサンドイッチとしては部分クラスとして隠れ prdd-1 行列を持ち, 上位クラスとして隠れ prdd-∞ 行列クラスを持つような prdd 行列サンドイッチと, 部分クラスに隠れ Z^o- 行列クラスを持ち, 上位クラスに隠れ Z^o+ 行列クラスを持つような Z^o 行列サンドイッチである. 先行研究では prdd 行列サンドイッチに着目し, 各行列を実数パラメータ k を用いることでパラメータ化し, ある程度 P 行列に近づくことができている.

本研究ではもう一方の P 行列サンドイッチである Z^o 行列サンドイッチに着目した. このサンドイッチから P 行列にせまるために, 実数 t を用い Z^o+ 行列と Z^o- 行列間を行き来できるようなパラメータ化を行った. 実際どのようにパラメータ化を行ったかというと Z^o- 行列,Z^o+ 行列,prdd-1 行列をそれぞれ端線行列表現を行いそれらの凸結合を取る形で $Z^o(t)$ 行列と呼ばれるものを作成した. この $Z^o(t)$ 行列は $t=0$ の時 prdd-1 行列になり,$t=\frac{1}{2}$ の時 Z^o- 行列になり,$t=1$ の時 Z^o+ 行列になっている.$Z^o(t)$ 行列の隠れ行列である隠れ $Z^o(t)$ 行列についても二者択一の定理が成り立っており, 判別は容易である. このことを用いて先行研究と同様に隠れ $Z^o(t)$ 行列のパラメータ t をどこまで下げられるかについて計算機実験をした. また LCP との関わりを調べるために P 行列の部分クラスについて LCP のピボットアルゴリズムが何回のループで終了できるかについての計算機実験を行っている.

目 次

1 序論

数理計画問題とはある与えられた制約条件のもとで目的の最適解 (最大値や最小値) を求める手法である。例えば生産計画問題によく使われ各式が一次式で表される線形計画問題, ポートフォリオ最適化などに使われ目的関数が二次式で表される二次計画問題などが有名である. このように数理計画問題はある目的を最適化しようとする意思決定を, 数理的に考えるときに有用である.

このような数理計画問題の一つとして線形相補性問題 (LCP) が知られている. この線形相補性問題は上記の線形計画問題や凸二次計画問題を特殊なパターンとして持つ問題として定式化できる他, 様々な均衡問題に利用されている. このような一般の線形相補性問題は多項式時間での解法を持たず, 解を持つかどうかの判定は難しいとされている. しかし LCP に入力される行列が P 行列と呼ばれる特殊な行列の時は LCP の解が存在することが保証されている. すなわち入力される行列が P 行列であることと LCP が解を持つことは必要十分条件となるため, 与えられた行列が P 行列かどうかを判定することは重要な研究テーマである.

P 行列とは全ての主座小行列の行列式が正となるような行列のことである. 主座小行列とは任意の添字の部分集合に対して行と列が同じ集合で添字付けされている部分正方行列のことである.P 行列であることを判別するにはこのような主座小行列を全ての添字の組み合わせについて調べる必要がある. すなわち $n \times n$ の正方行列に対して 2^n 乗個の行列の行列式を調べる計算時間が必要である. つまり n の値が大きくなればなるほど計算時間は爆発的に増加してしまう. このような P 行列の判定問題は co-NP 完全問題であるだろうという予想がされており, 多項式時間での解法は存在しないだろうとされている.

このような難しい P 行列問題に対して上位クラスで上から, 部分クラスで下からサンドイッチする P 行列サンドイッチというものが提案されている. 主に知られているサンドイッチとしては部分クラスに隠れ prdd-1 行列クラスを持ち, 上位クラスに隠れ prdd-∞ 行列クラスを持つサンドイッチと, 部分クラスに隠れ Z^o- 行列クラスを持ち, 上位クラスに隠れ Z^o+ 行列クラスを持つサンドイッチが知られている. これら各隠れ行列は二者択一の定理より線形計画問題を解くことで求めることができ, 判別は多項式時間である. つまり P 行列に近い行列クラスは多項式時間で判別可能になるのではないということである. 本研究では主に Z^o 行列クラスによるサンドイッチに着目した.

先行研究では prdd 行列クラスに着目し, その行列を実数 k でパラメーター化した k-prdd-in 行列クラス及び k-prdd-out 行列クラスというものが導入された. これらの行列クラスに対しても二者択一の定理が適用でき判別は多項式時間でできる.P 行列の下界はタイトであり k-prdd-in 行列クラスは小さくすることができず,k-prdd-out 行列クラスは $(n-2)$-prdd-out 行列クラスまで小さくすることができることがわかっている.

本研究では先行研究と同様に Z^o 行列クラスを実数 t でパラメーター化した $Z^o(t)$ 行列クラスを提案した. またこの $Z^o(t)$ 行列クラスについて計算機実験を行いどこまで t の値を狭められるか実験を行い予想をした. また prdd も含めた各行列に対して LCP のピボットアルゴリズムである PrincipalPivotMethod を適用した. このアルゴリズムはある特定の行列では最大 $2n$ 回程度のループで解けることがわかっている. 本研究ではサンドイッチに使われている行列を LCP が持つときに最大何回のループで計算可能かを調べ, 予想をした.

1.1 本論文の構成

まず第二章では数理計画問題で最も簡単であろう線形計画問題について紹介する. それに伴い双対問題, 双対定理, それを証明する二者択一の定理の紹介と証明をする. さらに線形計画問題を解くピボットアルゴリズムであるシンプレックス法とそれの効率を良くした改訂シンプレックス法についても紹介している.

第三章では線形相補性問題と P 行列を紹介している. また LCP に改訂シンプレックス法を適用した PrincipalPivotMethod や P 行列と LCP との関わりについても紹介している.

第四章ではまず本題である P 行列サンドイッチや隠れ行列の定義, その行列の二者択一の定理の証明を紹介している. 更に本研究のメインである Z^o 行列の拡張である $Z^o(t)$ 行列という新しい行列の性質などを紹介している. そして現段階での包含関係を紹介している.

第五章では先のパラメーター化行列にたいする計算機実験と, 改訂シンプレックス法を LCP に適用した PrincipalPivotMethod の繰り返し回数に対する計算機実験を行っている.

最後に付録としてそれらのソースコードを付けた.

2　線形計画問題と二者択一の定理

線形計画問題は数理計画問題の最も基礎的な一部であり, 日常の中のあらゆる場面で使用されている. 例えば生産計画問題や最短路問題などが代表的な例である. このような線計画問題は 1940 年に G.B.Dantzig によって初めて提案されたもので, 同時に解法であるシンプレックス法も考案された. 本章では数理計画問題の代表例として線形計画問題を紹介するとともに後の章で重要となる二者択一の定理, 改訂シンプレックス法についても紹介する.

2.1　線形計画問題とは

まず, ある条件のもとである関数を最大化または最小化するような数理モデルを数理計画問題 (mathematical programming problem), あるいは最適化問題 (optimization problem) と言い以下のように表現する.

$$\left|\begin{array}{ll} \text{最大化} & f(x) \\ \text{条　件} & \mathrm{x} \in S(\subseteq \mathbb{R}^n) \end{array}\right.$$

最大化または最小化の対象となる関数 f を目的関数 (objective function) と呼び, 条件を表す $\mathbb{R}^n$ の部分集合 S を制約条件 (constraints) という. 目的関数と制約条件どちらもが線形関数, すなわち一次式の形の等式, 不等式で表される場合の数理計画問題を線形計画問題 (linear programming problem:LP) といい, 経済, 金融, 工学, 経営学など様々な分野に用いられている. 以下に例題を示す.

例題 2.1 (生産計画問題)

3 つの材料を使い, 商品 A と商品 B を作る事を考える. 商品 A を作るには材料 1,2,3 がそれぞれ 1kg,1kg,3kg 必要であり, 商品 B を作るには材料 1,2,3 がそれぞれ 2kg,1kg,1kg 必要である.

また, 材料 1,2,3 の使用上限はそれぞれ 10kg,6kg,12kg までである. 商品 A,B は 1 つ作るとそれぞれ 2 万, 1 万の利益を出すとき, 商品 A,B をそれぞれ何個作ると最大の利益が得られるか.

この問題を表に整理する.

材料＼商品	商品 A	商品 B	上限
材料 1	1kg	2kg	10kg
材料 2	1kg	1kg	6kg
材料 3	3kg	1kg	12kg
利益	2 万	1 万	

ここで知りたいのはそれぞれの一日の最適生産量であるので,A,B の生産量をそれぞれ x_1, x_2 と置く. すると, 利益は A を作ると 2 万,B を作ると 1 万なので

$$2x_1 + x_2$$

と表される.

　また, 材料 1 は商品 A が作られるたびに 1kg, 商品 B が作られるたびに 2kg 消費され, 合わせて 10kg 以上は使えないので

$$x_1 + 2x_2 \leq 10$$

と表される.
材料 2,3 も同様に考えれば, 以下のように表される.

$$x_1 + x_2 \leq 6$$
$$3x_1 + x_2 \leq 12$$

更に,x_1, x_2 ともに 0 個より少なくは作れないので

$$x_1, x_2 \geq 0$$

と表される.
これらをまとめると以下のように表すことができる.

$$\text{最大化}\quad 2x_1 + x_2 \qquad \cdots\cdots \text{利益 (式 1)}$$
$$\text{条件式}\quad \begin{cases} x_1 + 2x_2 \leq 10 & \cdots\cdots \text{材料 1 の使用条件 (式 2)} \\ x_1 + x_2 \leq 6 & \cdots\cdots \text{材料 2 の使用条件 (式 3)} \\ 3x_1 + x_2 \leq 12 & \cdots\cdots \text{材料 3 の使用条件 (式 4)} \\ x_1, x_2 \geq 0 \end{cases}$$

これらの式は全て 2 変数の一次式で表されているのでグラフとして表現できる.

上のグラフで目的関数に当たる赤のグラフを条件式の青のグラフで作られる緑の範囲を動かして面積が最大になる時が最適解であり, その時の面積が最大の時, 最適値となる.

よって, この問題は最適解 $x_1, x_2 = (3, 3)$ の時, 最大値 (最適値)9 となる

以上のようにすべての式が一次式で表される問題を線形計画問題と呼ぶ. またこの例題のように制約のあるいくつかの原材料を元にして, いくつかの製品を生産し, そこから得られる利益を最大化する問題を生産計画問題と呼ぶ.

また目的関数の最大値を達成する時の解を最適解とよび, その時の目的関数の値を最適値と呼ぶ.

2.1.1 LP の形と表現

以上のような線形計画問題を一般化することを考える.n 個の変数 $x_1, x_2, ..., x_n$ に関する実数値関数 $f : \mathbb{R}^n \to \mathbb{R}$ が,n 個の実定数 $c_1, c_2, ..., c_n$ を用いて, 以下のように書き表すことの出来る時 f を線形関数 (linear function) あるいは 1 次関数という.

$$f(x_1, x_2, ..., x_n) = c_1x_1 + c_2x_2 + \cdots + c_nx_n = \sum_{j=1}^{n} c_jx_j$$

例えば $f(x_1, x_2, x_3) = 5x_1 + 4x_2 + x_3$ は線形関数であるが,$f(x_1, x_2, x_3) = x_1^2 + 2x_2\log x_3$ は線形関数ではない. 定数 b と線形関数 $f : \mathbb{R}^n \to \mathbb{R}$ で表される等式 $f(x_1, x_2, ..., x_n) = b$ を線形等式, 不等式 $f(x_1, x_2, ..., x_n) \geq b, f(x_1, x_2, ..., x_n) \leq b$ を線形不等式と呼ぶ.

線形計画問題 (linear programming problem,LP) を一般的な言い方に言い換えると,「与えられた有限個の線形等式あるいは線形不等式を満たす条件のもとで, 与えられた線形関数を最大化 (あるいは最小化) するような変数 $(x_1, x_2, ..., x_n)$ の具体的な値を求めよ, あるいはそのようなものが存在しないことを示せ」という問題であり, 次のように表されたものを一般形と呼ぶ.

$$\left|\begin{array}{ll} \text{最大化} & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ & \left\{\begin{array}{l} \alpha_{11}x_1 + \alpha_{12}x_2 + \cdots + \alpha_{1n}x_n = b_1 \\ \qquad\qquad \vdots \\ \alpha_{m1}x_1 + \alpha_{m2}x_2 + \cdots + \alpha_{mn}x_n = b_m \end{array}\right. \\ \text{条件 1} & \\ & \left\{\begin{array}{l} \beta_{11}x_1 + \beta_{12}x_2 + \cdots + \beta_{1n}x_n \leq u_1 \\ \qquad\qquad \vdots \\ \beta_{k1}x_1 + \beta_{k2}x_2 + \cdots + \beta_{kn}x_n \leq u_k \end{array}\right. \\ \text{条件 2} & \\ & \left\{\begin{array}{l} \gamma_{11}x_1 + \gamma_{12}x_2 + \cdots + \gamma_{1n}x_n \geq l_1 \\ \qquad\qquad \vdots \\ \gamma_{p1}x_1 + \gamma_{p2}x_2 + \cdots + \gamma_{pn}x_n \geq l_p \end{array}\right. \\ \text{条件 3} & \end{array}\right. \tag{1.1}$$

最大化の対象となる関数 $c_1x_1 + c_2x_2 + \cdots + c_nx_n$ のことを目的関数 (objective function) と呼ぶ. もし最小化が目的なら最小化と書く. ただし目的関数を -1 倍することで最大化を最小化に, 最小化を最大化にすることができるので最大化と最小化に本質的な違いは無い. また, 条件の代わりに制約条件という言葉も良く使われる. また各係数である c, α, β, γ はすでに値が決められている定数, 実定数であり, それに対して $x_j(j = 1, 2, ..., n)$ は様々な値を取りうる実変

数である.

線形計画法 (linear programming) とは線形計画問題にたいして, 数学的な性質を解析したりアルゴリズムの設計, 実装, 現実への応用などの総称である. 一般化の式 (1.1) はこのままでは分かりにくく, アルゴリズムの記述の際などに不都合を生じるので以下の 2 つの特殊な形の LP を導入する. 呼び名については様々あるが本論文では [1] に従う.

[不等式標準形]
条件が全て"≤" 向きの不等式で表され, 変数の非負条件を加えた形の最大化の LP を不等式標準形 (standard form of inequalities) と呼ぶ, 数式で表すと以下のようになる.

$$\left|\begin{array}{ll} \text{最大化} & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{条件} & \left\{\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \end{array}\right. \\ & (x_1, x_2, x_3, ..., x_n \geq 0) \end{array}\right.$$

$(x_1, x_2, x_3, ..., x_n \geq 0)$ のことを特別に変数の非負条件 (non-negativity conditions) と呼ぶ.

[等式標準形]
条件が全て"=" の線形等式で表され, 変数の非負条件を加えた形の最大化の LP を不等式標準形 (standard form of equalities) と呼ぶ, 数式で表すと以下のようになる.

$$\left|\begin{array}{ll} \text{最大化} & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{条件} & \left\{\begin{array}{c} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{array}\right. \\ & (x_1, x_2, x_3, ..., x_n \geq 0) \end{array}\right.$$

これまでに示した一般形の LP, 特殊系として各標準形の LP は変数の個数や制約条件式の個数をあまり変化させること無くお互いの形に変形できる.

[一般形→不等式標準形]
まず一般形から不等式標準形への変換を考える. 一般形 LP における条件式 1 で表される以下の式

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i \quad (i = 1, 2, ..., m)$$

は, 以下の 2 つの不等式に分解して考えることができる.

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i \quad (i = 1, 2, ..., m)$$
$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i \quad (i = 1, 2, ..., m)$$

すなわち上の不等式を同時に持つときに等式になるということである.
不等式の向き $\geq$ は, 二番目の不等式の両辺に -1 をかけることによって $\leq$ の形に統一できる.
さらに非負条件のない変数 x_j を自由変数 (free variable) というが, 自由変数 x_j については 2 つの人工変数 (artificial variable)$x_j^+ \geq 0$ と $x_j^- \leq 0$ を用い,

$$x_j = x_j^+ - x_j^- \quad (x_j^+, x_j^- \geq 0)$$

と分解して表現することが可能である. これらの式を $\leq$ に統一された不等式に代入することによって, 一般形の LP は不等式標準形に変数の個数, 制約条件式の個数ともにたかだか 2 倍で変換できる.

[不等式標準形→等式標準形]

次に不等式標準形から等式標準形への変換を考える. 不等式標準形の条件式,

$$\text{条件} \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \end{cases}$$

について考える. これらの m 個の不等式は,m 個の非負変数 $x_{n+i} \geq 0 (i = 1, 2, ..., m)$ を導入することにより, 等式標準形の条件式に簡単に変換可能である.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + x_{n+1} = b_1 \\ \vdots \quad \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + x_{n+m} = b_m \end{cases}$$

つまり, 左辺の少なかった分を新たな変数で補ってやれば良いという形である. このことから制約条件式の数は変化せず, 変数の個数は元々の問題の制約条件式の数だけ増やすことで不等式標準形から等式標準形へと変換可能である. なお, このような不等式のギャップを吸収する役目の非負変数 $(x_{n+1} \geq 0 (i = 1, 2, ..., m))$ のことをスラック変数 (slack variable) と呼ぶ.

[不等式標準形や等式標準形→一般形]

不等式標準形や等式標準形は一般形の特殊ケースである.

[LP の行列表現]

不等式標準形	最大化	$\boldsymbol{c}^T\boldsymbol{x}$
	条件	$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} (\boldsymbol{x} \geq \boldsymbol{0})$
等式標準形	最大化	$\boldsymbol{c}^T\boldsymbol{x}$
	条件	$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} (\boldsymbol{x} \geq \boldsymbol{0})$

ただし,

$$\boldsymbol{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}, \boldsymbol{A} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

である.

以上で各 LP の紹介と互換性についての説明を終える. 今後は適宜説明しやすい形の LP を採用する.

2.2 双対性理論

ここでは線形計画法の理論面の中核である双対性理論, 特に双対定理, 相補性定理, 基本定理について述べる. また証明で用いるとともに重要な定理である二者択一の定理についても述べる.

2.2.1 双対問題

ここでは双対問題というものを導入する. そのためにまず, 以下の様な線形計画問題を考える.

$$\left|\begin{array}{ll} \text{最大化} & 2x_1+3x_2+2x_3 \\ \text{条件} & \left\{\begin{array}{l} x_1+x_2+2x_3 \le 24 \\ 3x_1+x_2 \le 16 \\ 2x_2+x_3 \le 12 \end{array}\right. \\ & (x_1,x_2,x_3 \ge 0) \end{array}\right.$$

この問題において例えば (x_1, x_2, x_3)=$(2, 3, 4)$ は実行可能解である. そのときの目的関数の値は 21 になる. ということはもしこの問題により良い最適解が存在するのならばその最適解に対応する最適値は 21 以上である. このことを「21 は最適値の下界である」という. では逆に上界はどうなるか確かめるために以下のような操作を行う. まず各条件式を 2 倍,1 倍,1 倍して足し合わせる.

$$\begin{array}{rrrrrrrrrr} & 2 & \times & (& x_1 & + & x_2 & + & 2x_3 & \le & 24) \\ & 1 & \times & (& 3x_1 & + & x_2 & + & & \le & 16) \\ + & 1 & \times & (& & + & 2x_2 & + & x_3 & \le & 12) \\ \hline & & & & 5x_1 & + & 5x_2 & + & 5x_3 & \le & 76) \end{array}$$

新たな不等式 $5x_1+5x_2+5x_3 \le 76$ が得られた. この不等式と, 各変数 x_1, x_2, x_3 がそれぞれ非負であることから次の不等式

$$\text{目的関数値}=2x_1+3x_2+2x_3 \le 5x_1+5x_2+5x_3 \le 76$$

を得る. このことからもし最適解が存在すれば最適値は 76 以下であることがわかる. つまり上界は 76 であることが計算された.

次により一般的にそれぞれの式を y_1 倍,y_2 倍,y_3 倍したものを考える. つまり

$$\begin{array}{rrrrrrrrrr} & y_1 & \times & (& x_1 & + & x_2 & + & 2x_3 & \le & 24) \\ & y_2 & \times & (& 3x_1 & + & x_2 & + & & \le & 16) \\ + & y_3 & \times & (& & + & 2x_2 & + & x_3 & \le & 12) \\ \hline \multicolumn{10}{c}{(y_1+3y_2)x_1+(y_1+y_2+2y_3)x_2+(2y_1+y_3)x_3 \le 24y_1+16y_2+12y_3} \end{array}$$

を考える. 先ほどのように単純に上界が定まるわけではないが, 係数に着目した以下の様な不等式,

$$\begin{array}{lllllll} y_1 & + & 3y_2 & & & \ge & 2 \\ y_1 & + & y_2 & + & 2y_3 & \ge & 3 \\ 2y_1 & & & + & y_3 & \ge & 2 \end{array}$$

が成り立てば,

$$\begin{aligned} 2x_1 + 3x_2 + 2x_3 &\leq (y_1 + 3y_2)x_1 + (y_1 + y_2 + 2y_3)x_2 + (2y_1 + y_3)x_3 \\ &\leq 24y_1 + 16y_2 + 12y_3 \end{aligned}$$

が成り立ち,$24y_1 + 16y_2 + 12y_3$ が目的関数の上界となりうる.

ここで目的関数値の上界を最小化する問題を考えると以下のような最小化の LP になる.

$$\left| \begin{array}{ll} \text{最小化} & 24y_1 + 16y_2 + 12y_3 \\ \text{条件} & \left\{ \begin{array}{lllllll} y_1 & + & 3y_2 & & & \geq & 2 \\ y_1 & + & y_2 & + & 2y_3 & \geq & 3 \\ 2y_1 & & & + & y_3 & \geq & 2 \end{array} \right. \\ & (y_1, y_2, y_3, ..., x_n \geq 0) \end{array} \right.$$

このように最大化 LP の目的関数の上界を最小化する問題を, 双対問題 (dual problem) と呼ぶ. 全く同様の議論で不等式標準形の LP

$$\left| \begin{array}{ll} \text{最大化} & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{条件} & \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \end{array} \right. \\ & (x_1, x_2, x_3, ..., x_n \geq 0) \end{array} \right.$$

の双対問題は以下のようになる

$$\left| \begin{array}{ll} \text{最大化} & b_1y_1 + b_2y_2 + \cdots + b_my_n \\ \text{条件} & \left\{ \begin{array}{l} a_{11}y_1 + a_{21}y_2 + \cdots + a_{m1}y_m \leq c_1 \\ \vdots \\ a_{1n}y_1 + a_{2n}y_2 + \cdots + a_{mn}y_m \leq c_n \end{array} \right. \\ & (x_1, x_2, x_3, ..., x_n \geq 0) \end{array} \right.$$

となる.

双対問題を意識した LP は主問題 (primal problem) と呼ばれ, 双対問題と対で扱われる. 以下はその主問題と双対問題のペアの行列表現である.

$$\text{(P)} \left| \begin{array}{ll} \text{最大化} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{条件} & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}(\boldsymbol{x} \geq \boldsymbol{0}) \end{array} \right. \quad \text{(D)} \left| \begin{array}{ll} \text{最大化} & \boldsymbol{b}^T\boldsymbol{y} \\ \text{条件} & \boldsymbol{A}^T\boldsymbol{y} \geq \boldsymbol{c}(\boldsymbol{y} \geq \boldsymbol{0}) \end{array} \right.$$

等式標準形は以下のようになる.

$$\text{(P)} \left| \begin{array}{ll} \text{最大化} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{条件} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}(\boldsymbol{x} \geq \boldsymbol{0}) \end{array} \right. \quad \text{(D)} \left| \begin{array}{ll} \text{最大化} & \boldsymbol{b}^T\boldsymbol{y} \\ \text{条件} & \boldsymbol{A}^T\boldsymbol{y} = \boldsymbol{c} \end{array} \right.$$

(双対問題で非負条件がなくなることに注意) この双対問題 (D) について以下の性質が成り立つ.

性質 2.1 不等式標準形最大化問題を (P), その双対問題を (D) とする.(D) の双対問題は (P) 自身である.

(証明) 以下の双対問題

$$\text{(D)} \left| \begin{array}{ll} \text{最大化} & \boldsymbol{b}^T\boldsymbol{y} \\ \text{条件} & \boldsymbol{A}^T\boldsymbol{y} \geq \boldsymbol{c}(\boldsymbol{y} \geq \boldsymbol{0}) \end{array} \right.$$

は簡単な操作により以下の不等式標準形の最大化問題に変形できる.

$$\text{(D)} \left| \begin{array}{ll} \text{最大化} & -\boldsymbol{b}^T\boldsymbol{y} \\ \text{条件} & -\boldsymbol{A}^T\boldsymbol{y} \leq -\boldsymbol{c}(\boldsymbol{y} \geq \boldsymbol{0}) \end{array} \right.$$

この形は不等式標準形の LP の形なのでこの問題の双対問題 (D') を考えることができる.

$$\text{(D')} \left| \begin{array}{ll} \text{最大化} & -\boldsymbol{c}^T\boldsymbol{x'} \\ \text{条件} & (-\boldsymbol{A}^T)^T\boldsymbol{x'} \geq -\boldsymbol{b}(\boldsymbol{x'} \geq \boldsymbol{0}) \end{array} \right.$$

これを整理して書くと, 以下のようになる

$$\text{(D')} \left| \begin{array}{ll} \text{最大化} & \boldsymbol{c}^T\boldsymbol{x'} \\ \text{条件} & \boldsymbol{A}\boldsymbol{x'} \leq \boldsymbol{b}(\boldsymbol{x'} \geq \boldsymbol{0}) \end{array} \right.$$

となるので主問題 (P) と同じ形になる. 同様のことを行えば等式標準形でも成り立つ.□

これまでのことをまとめると以下のようになる.

$$\text{(P)} \left| \begin{array}{ll} \text{最大化} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{条件} & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}(\boldsymbol{x} \geq \boldsymbol{0}) \end{array} \right. \quad \begin{array}{c} \text{双対化} \\ \Rightarrow \\ \Leftarrow \\ \text{双対化} \end{array} \quad \text{(D)} \left| \begin{array}{ll} \text{最大化} & \boldsymbol{b}^T\boldsymbol{y} \\ \text{条件} & \boldsymbol{A}^T\boldsymbol{y} \geq \boldsymbol{c}(\boldsymbol{y} \geq \boldsymbol{0}) \end{array} \right.$$

2.2.2 諸定理と双対定理

はじめに前節での主問題 (P) と双対問題 (D) の集合を以下のように置く

$$\begin{aligned} X = &\quad \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\} \\ Y = &\quad \{\boldsymbol{y} \in \mathbb{R}^m | \boldsymbol{A}^T\boldsymbol{y} \geq \boldsymbol{c}, \boldsymbol{y} \geq \boldsymbol{0}\} \end{aligned}$$

定義 2.1 (実行可能)[1] $X \neq \emptyset$ のとき, 問題 (P) は実行可能 (feasible) であるという. 同様に $Y \neq \emptyset$ の時, 問題 (D) は実行可能であるという.

定義 2.2 (実行不可能)[1] $X = \emptyset$ のとき, 問題 (P) は実行不可能 (infeasible) であるという. 同様に $Y = \emptyset$ の時, 問題 (D) は実行不可能であるという.

これは解がどのようなもかにかかわらず, とにかく一つでも実行可能解があるときを実行可能であると呼び, 一つも解が無いときには実行不可能と呼ぶということである. また (P) や (D) が実行可能か実行不可能かのどちらか一方になることは明確である.
次に (P),(D) どちらも実行可能であるとき, その実行可能解に関する目的関数値の大小関係を示す性質として弱双対定理 (weak duality theorem) を紹介する.

定理 2.1 (弱双対定理)[1]　(P) と (D) が実行可能であるとし,$\boldsymbol{x} \in x, \boldsymbol{y} \in Y$ とする. 次の不等式が成り立つ.

$$\boldsymbol{c}^T\boldsymbol{x} \leq \boldsymbol{y}^T\boldsymbol{b}$$

(証明)　双対問題の定義と $\boldsymbol{x}, \boldsymbol{y}$ がそれぞれの実行可能解であることから

$$\boldsymbol{b}^T\boldsymbol{y} = \boldsymbol{y}^T\boldsymbol{b} \geq \boldsymbol{y}^T\boldsymbol{A}\boldsymbol{x} = (\boldsymbol{A}^T\boldsymbol{y})^T\boldsymbol{x} \geq \boldsymbol{c}^T\boldsymbol{x}$$

となる.□

弱双対定理の主張はもし主問題と双対問題の両方に実行可能解があれば, 最適解がありそうでその際の最適値がある程度予想できるということである. ここで最適解の定義をきちんとしておく.

定義 2.3 (最適解)[1]　$\boldsymbol{x}^* \in X$ が, 任意の $\boldsymbol{x} \in X$ に対し,$\boldsymbol{c}^T\boldsymbol{x}^* \geq \boldsymbol{c}^T\boldsymbol{x}$ を満たすとき,$\boldsymbol{x}^*$ を最大化問題 (P) の最適解 (optimal solution) という. 同様に,$\boldsymbol{y}^* \in Y$ が任意の $\boldsymbol{y} \in Y$ に対し,$\boldsymbol{b}^T\boldsymbol{y} \geq \boldsymbol{b}^T\boldsymbol{y}^*$ を満たすとき,$\boldsymbol{y}^*$ を最小化問題 (D) の最適解という

この最適解について以下の性質が成り立つ.

性質 2.2 (最適解の凸性)[1]　$\boldsymbol{x}^*, \boldsymbol{x}^{**}$ を (P) の異なる 2 つの最適解とする.$\boldsymbol{x}^*, \boldsymbol{x}^{**}$ の凸結合も最適解である.

(証明)　$\boldsymbol{x}^*, \boldsymbol{x}^{**}$ ともに最適解であるので最適値を x_f^* =$\boldsymbol{c}^T\boldsymbol{x}^* = \boldsymbol{c}^T\boldsymbol{x}^{**}$ とする.$\lambda \in [0,1]$ に対して $\boldsymbol{x}(\lambda) := \lambda\boldsymbol{x}^* + (1-\lambda)\boldsymbol{x}^{**}$ とする. そのとき $\boldsymbol{c}^T\boldsymbol{x}(\lambda) = \boldsymbol{c}^T\boldsymbol{x}^* + (1-\lambda)\boldsymbol{c}^T\boldsymbol{x}^{**}$=$x_f^*$ となり示される.□

これらのことと, 弱双対定理より $\boldsymbol{x}, \boldsymbol{y}$ がそれぞれ (P),(D) の最適解であるための十分条件が用意に導かれる.

系 2.1 $\boldsymbol{x}, \boldsymbol{y}$ をそれぞれ (P),(D) の実行可能解とする. このとき

$$\boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{b}^T\boldsymbol{y}$$

ならば $\boldsymbol{x}, \boldsymbol{y}$ はそれぞれ (P),(D) の最適解である

(証明)　$\boldsymbol{x}, \boldsymbol{y}$ はそれぞれ (P),(D) の実行可能解であるので, 弱双対定理より,

$$\boldsymbol{c}^T\boldsymbol{x} \leq \boldsymbol{b}^T\boldsymbol{y}$$

がなりたつ.

$$\boldsymbol{c}^T\boldsymbol{x} \leq \boldsymbol{b}^T\boldsymbol{y} = \boldsymbol{c}^T\boldsymbol{x}$$

より $\boldsymbol{c}^T\boldsymbol{x}$ を上回る (P) の実行可能解も,$\boldsymbol{b}^T\boldsymbol{y}$ を下回る（D）の実行可能かも存在しない. ゆえに $\boldsymbol{x}, \boldsymbol{y}$ はそれぞれ (P),(D) の最適解である.□

この系 2.1 の逆として以下の様な双対定理が存在する.

定理 2.2 (双対定理)[1]　(P) と (D) のどちらも実行可能ならば,(P),(D) のいずれも最適解を持ち最適値は一致する.

ここで, この後の議論を進める前に問題が非有界であるという性質を導入する

定義 2.4 (非有界な問題)[1]　任意の実数 M 対し,$\boldsymbol{c}^T\boldsymbol{x} > M$ となる $\boldsymbol{x} \in X$ が存在するとき最大化問題 (P) は非有界 (unboundd) であるという. 同様に, 任意の実数 M にたいし,$\boldsymbol{b}^T\boldsymbol{y} < M$ となる $\boldsymbol{y} \in Y$ が存在するとき, 最小化問題 (D) は非有界であるという.

次に LP の基本定理 (fundamental theorem) というものを紹介する.

定理 2.3 (基本定理)[1]　(P) が実行可能であり, かつ最適解を持たないならば,(P) は非有界である. 同様に,(D) が実行可能であり, かつ最適解を持たないならば,(D) は非有界である.

次に双対定理, 基本定理, 系 2.1 を組み合わせると以下の定理が得られる.

定理 2.4 (強双対定理)[1]　問題 (P) が最適解を持つための必要十分条件は問題 (D) が最適解を持つことである.

この定理の主張は LP を解くことを最適解を求めることと解釈したときに (P) が解けることと (D) が解けることは同じであるということである.

この節の最後として (P) と (D) に入力されたベクトルがそれぞれ最適解であるための必要十分条件である相補性定理 (complementarity theorem) を紹介する.

定理 2.5 (相補性定理)[1]　不等式標準形の LP(P) の実行可能解 $\boldsymbol{x}^*$ とその双対問題 (D) の実行可能解を $\boldsymbol{y}^*$ とする.$(\boldsymbol{x}^*, \boldsymbol{y}^*)$ がそれぞれ (P),(D) の最適解であるための必要十分条件は,

$$\begin{aligned} x_j^* \cdot (\boldsymbol{A}^T\boldsymbol{y}^* - \boldsymbol{c})_j = 0 \quad (j = 1, 2, \ldots, n) \\ y_i^* \cdot (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*)_i = 0 \quad (i = 1, 2, \ldots, m) \end{aligned}$$

である.

(証明)　双対定理と系 2.1 より,$\boldsymbol{x}^*, \boldsymbol{y}^*$ がそれぞれ (P),(D) の最適解であるための必要十分条件は,$\boldsymbol{c}^T\boldsymbol{x}^* = \boldsymbol{b}^T\boldsymbol{y}^*$ である. よって次の式がなりたつ,

$$0 \quad = \quad \boldsymbol{b}^T\boldsymbol{y}^* - \boldsymbol{c}^T\boldsymbol{x}^*$$

この右辺に $\boldsymbol{y}^{*T}(\boldsymbol{A}\boldsymbol{x}^*)$ を足して引く操作を行う.

$$0 \quad = \quad \boldsymbol{b}^T\boldsymbol{y}^* - \boldsymbol{c}^T\boldsymbol{x}^* + \boldsymbol{y}^{*T}(\boldsymbol{A}\boldsymbol{x}^*) - \boldsymbol{y}^{*T}(\boldsymbol{A}\boldsymbol{x}^*)$$

この式を整理すると以下のようになる,

$$\begin{aligned} 0 &= \boldsymbol{y}^{*T}\boldsymbol{b} - \boldsymbol{y}^{*T}\boldsymbol{A}\boldsymbol{x}^* + \boldsymbol{x}^{*T}\boldsymbol{A}\boldsymbol{y}^* - \boldsymbol{x}^{*T}\boldsymbol{c} \\ &= \boldsymbol{y}^{*T}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*) + \boldsymbol{x}^{*T}(\boldsymbol{A}^T\boldsymbol{y}^* - \boldsymbol{c}) \end{aligned}$$

となる.一方$\boldsymbol{x}^*, \boldsymbol{y}^*$はそれぞれ(P),(D)の実行可能解であることから,$\boldsymbol{y}^* \geq 0, \boldsymbol{x}^* \geq 0$かつ,$\boldsymbol{A}\boldsymbol{x}^* \leq \boldsymbol{b}$より$\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^* \geq 0$,$\boldsymbol{A}^T\boldsymbol{y}^* \geq \boldsymbol{c}$より$\boldsymbol{A}^T\boldsymbol{y}^* - \boldsymbol{c} \geq 0$がなりたつ.

この条件式のもとで,$\boldsymbol{y}^{*T}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*) + \boldsymbol{x}^{*T}(\boldsymbol{A}^T\boldsymbol{y}^* - \boldsymbol{c}) = 0$の式は,

$$\begin{aligned} \boldsymbol{y}^{*T}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*) &= 0 \\ \boldsymbol{x}^{*T}(\boldsymbol{A}^T\boldsymbol{y}^* - \boldsymbol{c}) &= 0 \end{aligned}$$

と等価である.各ベクトル,$\boldsymbol{y}^{*T}, \boldsymbol{x}^{*T}, \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*, \boldsymbol{A}^T\boldsymbol{y}^* - \boldsymbol{c}$は全て非負ベクトルである.よって上の式は,

$$\begin{aligned} x_j^* \cdot (\boldsymbol{A}^T\boldsymbol{y}^* - \boldsymbol{c})_j = 0 \quad (j = 1, 2, \ldots, n) \\ y_i^* \cdot (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*)_i = 0 \quad (i = 1, 2, \ldots, m) \end{aligned}$$

と等価である.よって定理は示された.□

2.2.3 二者択一の定理と双対定理

この節では前節での双対定理を証明するために必要な二者択一の定理について紹介と説明する.また今後の章や説にとっても重要な定理なので証明もここで行う.

まずFarkasの二者択一の定理と呼ばれるものを紹介する

定理 2.6 [1](Farkasの二者択一の定理)$\boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{b} \in \mathbb{R}^m$とする.次のいずれか一方かつ一方のみが非空である.

$$\begin{aligned} X_F(\boldsymbol{A}, \boldsymbol{b}) &:= \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\} \\ Y_F(\boldsymbol{A}, \boldsymbol{b}) &:= \{\boldsymbol{y} \in \mathbb{R}^m | \boldsymbol{A}^T\boldsymbol{y} \geq \boldsymbol{0}, \boldsymbol{b}^T\boldsymbol{y} < 0\} \end{aligned}$$

この定理の幾何学的な意味は[1]参照

この二者択一の定理を証明するために一般形の二者択一の定理を証明する.

定理 2.7 [1](一般形の二者択一の定理)$\boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{b} \in \mathbb{R}^m$とする.次のいずれか一方かつ一方のみが非空である.行列$\boldsymbol{A}$の添字集合を$N = \{1, 2, \ldots, n\}$とし,$\boldsymbol{A}_i$を$\boldsymbol{A}$の第i列のベクトルとする.Nの分割I, J, Kに対し,$X(\boldsymbol{A}, \boldsymbol{b}, I, J, K), Y(\boldsymbol{A}, \boldsymbol{b}, I, J, K)$を以下のように定義する.

$$X(\boldsymbol{A}, \boldsymbol{b}, I, J, K) := \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ x_i \geq 0 (i \in I) \\ x_i \text{は自由} (i \in J) \\ x_i = 0 (i \in K) \end{array} \right.$$

$$Y(\boldsymbol{A}, \boldsymbol{b}, I, J, K) := \left\{ \boldsymbol{y} \in \mathbb{R}^m \;\middle|\; \begin{array}{l} \boldsymbol{A}_i^T\boldsymbol{y} \geq 0 \\ \boldsymbol{A}_i^T\boldsymbol{y} = 0 (i \in I) \\ \boldsymbol{A}_i^T, \boldsymbol{y} \text{は自由} (i \in J) \\ \boldsymbol{b}^T\boldsymbol{y} < 0 (i \in K) \end{array} \right.$$

N の任意の分割 I, J, K に関して X, Y のどちらか一方かつ一方のみが非空である.

この定理における $I = N, J = \emptyset, K = \emptyset$ の場合が Farkas の二者択一の定理である. また $I = \emptyset, J = N, K = \emptyset$ の場合以下の様な Gale の二者択一の定理を得る.

定理 2.8 [1](Gale の二者択一の定理)$\boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{b} \in \mathbb{R}^m$ とする. 次のいずれか一方かつ一方のみが非空である.

$$\begin{aligned} X_F(\boldsymbol{A}, \boldsymbol{b}) &:= \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\} \\ Y_F(\boldsymbol{A}, \boldsymbol{b}) &:= \{\boldsymbol{y} \in \mathbb{R}^m | \boldsymbol{A}^T\boldsymbol{y} = \boldsymbol{0}, \boldsymbol{b}^T\boldsymbol{y} < 0\} \end{aligned}$$

(証明) [1] 参照

これらを用いて一般形の二者択一の定理を証明する.

(証明) [1](一般形の二者択一の定理) 表記を単純にするために $X(\boldsymbol{A}, \boldsymbol{b}, I, J, K) := X(I, J, K)$, $Y(\boldsymbol{A}, \boldsymbol{b}, I, J, K) := Y(I, J, K)$ とする.$\bar{\boldsymbol{x}} \in X(I, J, K)$ かつ $\bar{\boldsymbol{y}} \in Y(I, J, K)$ とすると

$$\begin{aligned} 0 > \boldsymbol{b}^T\boldsymbol{y} &= (\boldsymbol{A}\bar{\boldsymbol{x}})^T\bar{\boldsymbol{y}} = \bar{\boldsymbol{y}}^T\boldsymbol{A}\bar{\boldsymbol{x}} \\ &= \sum_{i \in I} \bar{\boldsymbol{y}}\boldsymbol{A}_i\bar{x}_i + \sum_{i \in J} \bar{\boldsymbol{y}}\boldsymbol{A}_i\bar{x}_i + \sum_{i \in K} \bar{\boldsymbol{y}}\boldsymbol{A}_i\bar{x}_i \\ &= \sum_{i \in I} \bar{\boldsymbol{y}}\boldsymbol{A}_i\bar{x}_i + 0 + 0 \geq 0 \end{aligned}$$

となり矛盾. ゆえに $X(I, J, K), Y(I, J, K)$ のどちらか一方のみしか非空にならない. 以下ではそのことを I に関する帰納法を用いて示す.

(1) $|I| = 0$ の時, すなわち $I = \emptyset$ の時 Gale の二者択一の定理と同様になるので確認済みである.

(2) 正の整数 k に関して $|I| < k$ ならば主張が成り立つと仮定して $|I| = k$ の場合に主張が成り立つことを示す.

I の要素を一つ選び l とし, 次の 4 つの集合を考える.

$$\begin{array}{ll} X(I - l, J + l, K) & X(I - l, J, K + l) \\ Y(I - l, J + l, K) & Y(I - l, J, K + l) \end{array}$$

$X(I - l, J, K + l) \subseteq X(I, J, K)$ は容易に確かめられる. すなわち $X(I - l, J, K + l) \neq \emptyset$ ならば $X(I, J, K) \neq \emptyset$ である. ゆえに $X(I - l, J, K + l) = \emptyset$ の場合のみ考えれば良い. また帰納法の仮定から $Y(I - l, J, K + l)$ が非空の場合のみ議論すれば良い.

Y も同様に考えれば $Y(I - l, J + l, K) = \emptyset$ の場合のみ考えれば良い. また同様に帰納法の仮定から $X(I - l, J + l, K)$ が飛空の場合のみ議論すれば良い.　これまでの議論よい考える必要があるのは以下の場合のみである.

$$\exists \bar{\boldsymbol{x}} \in X(I-l, J+l, K) \text{ かつ } \exists \bar{\boldsymbol{y}} \in Y(I-l, J, K+l)$$

$\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}$ の定義より

$$\begin{aligned} 0 > \boldsymbol{b}^T \boldsymbol{y} &= (\boldsymbol{A}\bar{\boldsymbol{x}})^T \bar{\boldsymbol{y}} = \bar{\boldsymbol{y}}^T \boldsymbol{A} \bar{\boldsymbol{x}} \\ &= \sum_{i \in I-l} \bar{\boldsymbol{y}} \boldsymbol{A}_i \bar{x}_i + \bar{\boldsymbol{y}}^T \boldsymbol{A}_l \bar{x}_l + \sum_{i \in J \cup K} \bar{\boldsymbol{y}} \boldsymbol{A}_i \bar{x}_i \\ &= \sum_{i \in I-l} \bar{\boldsymbol{y}} \boldsymbol{A}_i \bar{x}_i + \bar{\boldsymbol{y}}^T \boldsymbol{A}_l \bar{x}_l \geq \bar{\boldsymbol{y}} \boldsymbol{A}_l \bar{x}_l \end{aligned}$$

となり,$\bar{\boldsymbol{y}}^T \boldsymbol{A}_l > 0$ または $\bar{x}_l > 0$ が成り立つ. これはそれぞれ $Y(I,J,K), X(I,J,K)$ の解となるので主張は証明された.□

次に最も汎用性のある二者択一の定理である Tucker の二者択一の定理を紹介しておく.

定理 2.9 [1](Tucker の二者択一の定理)$\boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{b} \in \mathbb{R}^m$ とする. 次のいずれか一方かつ一方のみが非空である. 行列 $\boldsymbol{A}$ の添字集合を $N = \{1, 2, \ldots, n\}$ とし,$\boldsymbol{A}_i$ を $\boldsymbol{A}$ の第 i 列のベクトルとする.N の分割 I, J, K, L に対し,$X_T(\boldsymbol{A}, \boldsymbol{b}, I, J, K, L), Y_T(\boldsymbol{A}, \boldsymbol{b}, I, J, K, L)$ を以下のように定義する.

$$X_T(\boldsymbol{A}, \boldsymbol{b}, I, J, K, L) := \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ x_i \geq 0 (i \in I) \\ x_i \text{ は自由 } (i \in J) \\ x_i = 0 (i \in K) \\ x_i > 0 (i \in L) \end{array} \right.$$

$$Y_T(\boldsymbol{A}, \boldsymbol{b}, I, J, K, L) := \left\{ \boldsymbol{y} \in \mathbb{R}^m \;\middle|\; \begin{array}{l} \boldsymbol{A}_I^T \boldsymbol{y} \geq 0 \\ \boldsymbol{A}_J^T \boldsymbol{y} = 0 (i \in I) \\ \boldsymbol{A}_K^T, \boldsymbol{y} \text{ は自由 } (i \in J) \\ \boldsymbol{A}_L^T, \boldsymbol{y} \geq \boldsymbol{0} (\neq 0) \end{array} \right.$$

N の任意の分割 I, J, K に関して X_T, Y_T のどちらか一方かつ一方のみが非空である.

これらの定理, 特に Farkas の二者択一の定理を用いることで先の節で説明した双対定理や基本定理を証明できる. 本論文の主題とは大きく外れるので [1] を参照のこと.

2.3 シンプレックス法

この章の最後では LP を解くための最も簡単なアルゴリズムであるシンプレックス法を紹介する. またシンプレックス法の効率をより良くした改訂シンプレックス法についても紹介する. まず先に例題である以下の様な最大化 LP を考える.

$$\text{最大化} \quad 2x_1 + x_2$$

$$\text{条件式} \quad \left\{ \begin{array}{l} x_1 + 2x_2 \leq 10 \\ x_1 + x_2 \leq 6 \\ 3x_1 + x_2 \leq 12 \\ x_1, x_2 \geq 0 \end{array} \right.$$

さらにこの LP にスラック変数を加えて以下の様な等式標準形の形に変形する.

$$\text{最大化}\quad 2x_1 + x_2$$
$$\text{条件式}\quad \begin{cases} x_1 + 2x_2 + x_3 & & & = 10 \\ x_1 + x_2 & +x_4 & & = 6 \\ 3x_1 + x_2 & & +x_5 & = 12 \\ x_1, x_2, x_3, x_4, x_5 \geq 0 & & & \end{cases}$$

つぎにこの式を以下のような辞書と呼ばれる形式に変形する

$$\text{最大化}\quad z = 2x_1 + x_2$$
$$\text{条件式}\quad \begin{cases} x_3 = 10 - x_1 - 2x_2 \\ x_4 = 6 - x_1 - x_2 \\ x_5 = 12 - 3x_1 - x_2 \\ x_1, x_2, x_3, x_4, x_5 \geq 0 \end{cases}$$

この時点では右辺の変数を全て 0 にすれば解は得られるのでこの時点の解候補は以下のように書ける.

$$\begin{array}{rl} & (z, \quad x_1, \quad x_2, \quad x_3, \quad x_4, \quad x_5) \\ = & (0, \quad 0, \quad 0, \quad 10, \quad 6, \quad 12) \end{array}$$

ここから最適解を探っていく.

step1:目的関数の係数が正のものを一つ選ぶ.(列選択) 今回は x_1 とする.
step2:選んだ変数を 0 から α に置き換える.$(x_1 : 0 \to \alpha, x_2 : 0)$ 置き換えて以下のように整理して α が最も小さくなる行を選ぶ (行選択).

$$z : 2\alpha$$
$$x_3 : 10 - \alpha \geq 0 \to \alpha \leq 10$$
$$x_4 : 6 - \alpha \geq 0 \to \alpha \leq 6$$
$$x_5 : 12 - 3\alpha \geq 0 \to \alpha \leq 4$$

今回は $\alpha \leq 4$ が最小なので x_5 の行を選択する.

step3:解候補と辞書更新. まず,step2 より $x_1 = 4, x_5 = 0$ として解候補を更新する.

$$\begin{array}{rl} & (z, \quad x_1, \quad x_2, \quad x_3, \quad x_4, \quad x_5) \\ = & (8, \quad 4, \quad 0, \quad 10, \quad 6, \quad 0) \end{array}$$

さらに x_5 の行について x_5 と x_1 の役割を入れ替えた以下の様な式を他の行にも代入する.

$$x_1 = 4 - \tfrac{1}{3}x_5 - \tfrac{1}{3}x_2$$

この式を他の行にも代入して辞書を更新すると以下の様な更新された辞書が得られる.

$$
\begin{aligned}
&\text{最大化} \quad z = 8 + \tfrac{1}{3}x_2 - \tfrac{2}{3}x_5 \\
&\text{条件式} \quad \begin{cases} x_3 = 6 - \frac{5}{3}x_2 + \frac{1}{3}x_5 \\ x_4 = 2 - \frac{2}{3}x_2 + \frac{1}{3}x_5 \\ x_1 = 4 - \frac{1}{3}x_2 - \frac{1}{3}x_5 \\ x_1, x_2, x_3, x_4, x_5 \geq 0 \end{cases}
\end{aligned}
$$

step4:目的関数の係数の値が全て負ならば終了する. その時の解候補が最適解であり最適値は z の部分になる. 今回は一回のループのみにする.

このアルゴリズムをまとめると以下のように書くことができる. そのために辞書を一般化して以下のように書く.

表 1: シンプレックス辞書

		x_1	x_2	$\cdots$	x_n
x_f	0	c_1	c_2	$\cdots$	c_n
x_{n+1}	b_1	$-a_{11}$	$-a_{12}$	$\cdots$	$-a_{1n}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
x_{n+m}	b_m	$-a_{m1}$	$-a_{m2}$	$\cdots$	$-a_{mn}$

以下アルゴリズム.

[シンプレックス法][1]
入力:不等式標準形 LP, 但し $b_i \geq 0 (i = 1, 2, \ldots, m)$
出力:最適辞書 D
初期化:$N := \{1, 2, \ldots, n\}; B := \{n+1, n+2, \ldots, n+m\}$

D:

		x_1	x_2	$\cdots$	x_n
x_f	0	c_1	c_2	$\cdots$	c_n
x_{n+1}	b_1	$-a_{11}$	$-a_{12}$	$\cdots$	$-a_{1n}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
x_{n+m}	b_m	$-a_{m1}$	$-a_{m2}$	$\cdots$	$-a_{mn}$

Step1(最適性判定):
　$\bar{c}_j \leq 0$ ならば D を出力して終了 (最適解が求まっている)
Step2(ピボット列選択):
　s を $\{j | \bar{c}_j > 0, j \in N\}$ からひとつ選ぶ
Step3(非有界性判定):
　$\bar{a}_{is} \leq 0 (\forall i \in B)$ ならば D を出力して終了 (非有界のため終了)
Step4(ピボット行選択):
　$\bar{a}_{rs} > 0$ かつ $\bar{b}_r / \bar{a}_{rs} = min\{\bar{b}_i / \bar{a}_{is} | \bar{a}_{is} > 0, i \in B\}$ となる r を一つ選ぶ
Step5(選択行, 列に対する辞書更新):
　例のように辞書を更新し, その辞書を新たな辞書とする.
　$N := N - s + r; B := B - r + s;$ として Step1 に戻る

以上が LP を解くためのシンプレックス法になる.

2.3.1 改訂シンプレックス法

ここでは前の節で紹介したシンプレックス法での辞書全体を更新するのでは無く, 辞書の必要な部分のみを計算することで演算回数を少なくしたものである, 改訂シンプレックス法を紹介する. またこの後の章で出てくる LCP のアルゴリズムにも応用できるためアルゴリズムの紹介のみしておく.

[改訂シンプレックス法][1]
入力:不等式標準形 LP, 但し $\boldsymbol{b} \geq \boldsymbol{0}$
出力:最適基底解, あるいは非有界であることの証明
初期化:$N := \{1, 2, \ldots, n\}; B := \{n+1, n+2, \ldots, n+m\}$
Step1:
$\boldsymbol{A}_B\bar{\boldsymbol{b}} = \boldsymbol{b}$ を $\bar{\boldsymbol{b}} \in \mathbb{R}^B$ について解く
$\boldsymbol{y}^T\boldsymbol{A}_B = \boldsymbol{c}_B^T$ を $\boldsymbol{y} \in \mathbb{R}^m$ について解く
$\bar{\boldsymbol{c}}_N^T := \boldsymbol{c}_N^T - \boldsymbol{y}^T\boldsymbol{A}_N$ を計算する
$\bar{\boldsymbol{c}}_N^T \leq 0$ ならば $(\boldsymbol{x}_B, \boldsymbol{x}_N)$=$(\bar{\boldsymbol{b}}, \boldsymbol{0})$ を出力して終了する.
Step2:
s を $\{j|\bar{c}_j > 0, j \in N\}$ からひとつ選ぶ
Step3:
$\boldsymbol{A}_B\boldsymbol{d} = A_s$ を $\boldsymbol{d} \in \mathbb{R}^B$ について解く.
$\boldsymbol{d} \leq 0$ なら終了.
Step4(ピボット行選択):
$d_r > 0$ かつ $\bar{b}_r/d_r = min\{\bar{b}_i/d_i|d_i > 0, i \in B\}$ となる r を一つ選ぶ
Step5(選択行, 列に対する辞書更新):
ピボット演算の代わりに次の操作を行う.
$N := N - s + r; B := B - r + s;$ として Step1 に戻る

アルゴリズムからわかるように行列とベクトルの計算のみから LP を解いている.
このアルゴリズムの長所として辞書全体を更新しないため計算による誤差が小さくなること, 横長の辞書の場合一家の繰り返しにかかる演算回数が少なくなることなどがあげられる.

3　線形相補性問題とP行列

この章では数理計画問題の一つであり, 線形相補性問題と呼ばれ, 均衡問題などに利用されたり, 線形計画問題を特殊パターンとしてもつような問題について述べている. また入力される行列によるときやすさなどに注目している.

3.1　線形相補性問題

まずはじめに線形相補性問題の説明を行う. 線形相補性問題 (Linear complementarity problem, LCP) とは, 与えられたある行列 $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ と,$\boldsymbol{q} \in \mathbb{R}^n$ に対して,$\boldsymbol{y} = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{q}$, $\boldsymbol{x} \geq \boldsymbol{0}$, $\boldsymbol{y} \geq \boldsymbol{0}$, $x_i \cdot y_i = 0$ $(i = 1, 2, \ldots, n)$ を満たすようなベクトルの組 $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ を求める問題である.

このような入力される行列とベクトル $\boldsymbol{M}, \boldsymbol{q}$ で決定される LCP$(\boldsymbol{M}, \boldsymbol{q})$ をまとめて以下のように書く.

$$\mathrm{LCP}(\boldsymbol{M}, \boldsymbol{q}) \left| \begin{array}{lll} \text{find} & \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n & \\ \text{s.t.} & \boldsymbol{y} = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{q} & (\boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{y} \geq \boldsymbol{0}) \\ & x_i \cdot y_i = 0 & (i = 1, 2, \ldots, n) \end{array} \right.$$

このような LCP は入力される行列やベクトルによって解の一意性や解きやすさが大きく変わる. ここでは後述する P 行列を含むような行列である S 行列と呼ばれるものをまず紹介する.

定義 3.1 (S 行列)[1]
$\boldsymbol{M} \in \mathbb{R}^{n\times n}$ とする. 次の条件を満たす $\boldsymbol{x} \in \mathbb{R}^n$ が存在するとき,$\boldsymbol{M}$ は S 行列行列 (S-matrix) であるという.

$$\boldsymbol{M}\boldsymbol{x} > \boldsymbol{0} \quad (\boldsymbol{x} > \boldsymbol{0})$$

この S 行列の判別は線形計画問題を解くのと同じくらいの手間でできるということを行っているのが以下の補助定理である.

補助定理 3.1　二者択一の定理 [1]
$\boldsymbol{M} \in \mathbb{R}^{n\times n}$ について，以下のいずれか一方のみが成り立つ.

$$\begin{array}{ll} \text{(i)} & X_S := \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{M}\boldsymbol{x} > \boldsymbol{0}, \boldsymbol{x} > \boldsymbol{0}\} \neq \emptyset \\ \text{(ii)} & Y_S := \{\boldsymbol{y} \in \mathbb{R}^n | \boldsymbol{M}^{\boldsymbol{T}}\boldsymbol{y} \leq \boldsymbol{0}, \boldsymbol{y} \geq \boldsymbol{0}, \boldsymbol{y} \neq \boldsymbol{0}\} \neq \emptyset \end{array}$$

さらに S 行列は S 行列を入力行列に持つ LCP の解の候補が常に存在するということで特徴付けられる.

性質 3.1 [1]　$\boldsymbol{M} \in \mathbb{R}^{n\times n}$ が S 行列であることの必要十分条件は任意の $\boldsymbol{q} \in \mathbb{R}^n$ に対して

$$\{(\boldsymbol{x}, \boldsymbol{y}) | \boldsymbol{y} = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{q}, \boldsymbol{x}, \boldsymbol{y} \geq \boldsymbol{0}\}$$

が解をもつことである.

(証明) (⇒):S 行列の定義から明らか.
(⇐):$\boldsymbol{q} \in \mathbb{R}^n$ は任意なので,$\boldsymbol{q} < \boldsymbol{0}$ となるように選ぶ.$\{(\boldsymbol{x},\boldsymbol{y})|\boldsymbol{y} = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{q}, \boldsymbol{x}, \boldsymbol{y} \geq \boldsymbol{0}\} \neq \emptyset$ であるから,$(\boldsymbol{x}', \boldsymbol{y}')$ をとると,$\boldsymbol{y}' \geq 0$ より $\boldsymbol{M}\boldsymbol{x}'$ と $\boldsymbol{q}$ を足して正にならなくてはいけないので,$\boldsymbol{M}\boldsymbol{x}' \geq -\boldsymbol{q} > \boldsymbol{0}$ $(\boldsymbol{x}' \geq \boldsymbol{0})$ を満たす. ここで,$\boldsymbol{x} := \boldsymbol{x}' + \epsilon\boldsymbol{e}$ とすれば，$\boldsymbol{M}\boldsymbol{x}' > \boldsymbol{0}$ なので,$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{M}\boldsymbol{x}' + \epsilon\boldsymbol{M}\boldsymbol{e} > 0$ となる $\epsilon > 0$ が存在する.

このとき,$\boldsymbol{x} := \boldsymbol{x}' + \epsilon\boldsymbol{e} > \boldsymbol{0}$ であることから, 任意に入力された行列 $\boldsymbol{M}$ に対して $\boldsymbol{M}\boldsymbol{x} > \boldsymbol{0}(\boldsymbol{x} > \boldsymbol{0})$ となるベクトル $\boldsymbol{x}$ が必ず存在することがわかる. ゆえに,$\boldsymbol{M}$ は S 行列である.□

このことから, 行列 $\boldsymbol{M}$ が S 行列である場合は，LCP$(\boldsymbol{M}, \boldsymbol{q})$ を解くためには, 以下の非線形最適化問題を考えればよい.

$$(\text{式 } 3.1)\left|\begin{array}{ll} \text{最小化} & \boldsymbol{q}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{M}\boldsymbol{x} \\ \text{条　件} & \boldsymbol{M}\boldsymbol{x} + \boldsymbol{q} \geq \boldsymbol{0} \quad (\boldsymbol{x} \geq \boldsymbol{0}) \end{array}\right.$$

上記の問題を考え, 最適値が 0 の場合は元の LCP の解となり, 0 より大きい場合は LCP には解が存在しない.

これらのことより以下の補助定理が言える

補助定理 3.2 $\boldsymbol{M} \in \mathbb{R}^{n\times n}$,$\boldsymbol{q} \in \mathbb{R}^n$ とする.$\{(\boldsymbol{x},\boldsymbol{y})|\boldsymbol{y} = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{q}, \boldsymbol{x}, \boldsymbol{y} \geq \boldsymbol{0}\} \neq \emptyset$ ならば (式 3.1) は最適解 $\boldsymbol{x}^*$ を持つ. 更に最適解 $\boldsymbol{x}^*$ に対し以下の式を満たす $\boldsymbol{u}^*$ が存在する.

$$\begin{array}{ll} \text{(i)} & \boldsymbol{q} + (\boldsymbol{M} + \boldsymbol{M}^T)\boldsymbol{x}^* - \boldsymbol{M}^T\boldsymbol{u}^* \geq 0 \\ \text{(ii)} & (\boldsymbol{x}^*)^T(\boldsymbol{q} + (\boldsymbol{M} + \boldsymbol{M}^T)\boldsymbol{x}^* - \boldsymbol{M}^T\boldsymbol{u}^*) = 0 \\ \text{(iii)} & \boldsymbol{u}^* \geq \boldsymbol{0} \\ \text{(iv)} & (\boldsymbol{u}^*)^T(\boldsymbol{q} + \boldsymbol{M}\boldsymbol{x}^*) = 0 \\ \text{(v)} & (\boldsymbol{x}^* - \boldsymbol{u}^*)_i(\boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{u}^*))_i \leq 0 \ (i = 1, 2, \ldots, n) \end{array}$$

(証明) (i) (iv) は非線形最小化問題 (3.1) の KKT 条件となっているので明らかに成り立つ.(ii) より以下の式が成り立つ.

$$\begin{aligned} 0 &= \boldsymbol{x}^{*T}\{(\boldsymbol{q} + \boldsymbol{M}\boldsymbol{x}^*) + \boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{u}^*)\} \\ &= \sum_{i=1}^{n} x_i^*(\boldsymbol{q} + \boldsymbol{M}\boldsymbol{x}^*)_i + \sum_{i=1}^{n} x_i^*(\boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{u}^*))_i \end{aligned}$$

一方,(i) より $i = 1, 2, \ldots, n$ に対して

$$\boldsymbol{x}_i^*(\boldsymbol{q} + \boldsymbol{M}\boldsymbol{x})_i + x_i^*(\boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{z}^*))_i \geq 0$$

である. よって,$\boldsymbol{x}_i^*(\boldsymbol{q} + \boldsymbol{M}\boldsymbol{x})_i + x_i^*(\boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{z}^*))_i = 0$ でなけらばならない. ゆえに

$$x_i^*(\boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{z}^*))_i \leq 0$$

が成り立つ. 同様に,(i) より,$i = 1, 2, \ldots, n$ に対して

$$u_i^*(\boldsymbol{q} + \boldsymbol{M}\boldsymbol{x})_i + u_i^*(\boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{z}^*))_i \geq 0$$

が得られる.(iv) より,$u_i^*(\boldsymbol{q} + \boldsymbol{M}\boldsymbol{x})_i = 0$ であるので

$$-u_i^*(\boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{z}^*))_i \leq 0$$

$x_i^*(\boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{z}^*))_i \leq 0$,$-u_i^*(\boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{z}^*))_i \leq 0$ の 2 つの式より (v) が得られる.□

3.2 P 行列

この節では P 行列と呼ばれる行列について説明する.

定義 3.2 (P 行列)[1] $n \times n$ 実行列 $\boldsymbol{A}$ が P 行列であるとは, 任意の添字の部分集合 $J \subseteq \{1, 2, \ldots, n\}$ に対し,

$$\det(\boldsymbol{A}_{JJ}) > 0$$

となることである. 但し $\boldsymbol{A}_{JJ}$ は行と列が J で添字付けされた,$\boldsymbol{A}$ の $J \times J$ 部分正方行列でこれを主座小行列と呼ぶ.

以下に実例を示す.

例 3.1 (P 行列ではない例,P 行列である例)
(P 行列ではない例)

$$\boldsymbol{A} = \begin{bmatrix} 2 & 1 & 2 \\ 5 & 3 & 3 \\ 2 & 6 & 4 \end{bmatrix}$$

このような行列は P 行列ではない. なぜなら $J = \{2, 3\}$ のとき,

$$\boldsymbol{A}_{JJ} = \begin{bmatrix} 3 & 3 \\ 6 & 4 \end{bmatrix}$$

となるので $\det(\boldsymbol{A}_{JJ}) = -6 \leq 0$ となるためである.
(P 行列である例)

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix}$$

このような行列は P 行列である. なぜなら $J = \{\{\emptyset\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$ の主座小行列, すなわち,

$$[\,],[1],[1],[1], \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix}$$

これら全ての行列式の値が 0 より大きくなるためである.

ここで大事なことは P 行列で無いことを確かめるのは簡単であるが P 行列であることを確かめるのは難しいことである.P 行列でないことを確かめる際には首座小行列式が負になる添字の組を一つ見つければいいのに対して,P 行列であることを確かめるには 2^n 個の行列式を計算しなければならないためである.

このような P 行列に対して以下の様な定理や性質が成り立っている.

性質 3.2 [1]$\boldsymbol{M} \in \mathbb{R}^{n\times n}$ が P 行列ならば以下の 3 つが成り立つ.

(i) $\boldsymbol{M}$ の転置行列 $\boldsymbol{M}^T$ も P 行列である.
(ii) 任意の正則な対角行列 $\boldsymbol{D}$ について，$\boldsymbol{DMD}$ も P 行列である
(iii) $d_1, d_2, \ldots, d_n \geq 0$ とする。 $\boldsymbol{M} + \mathrm{diag}(d_1, d_2, \ldots, d_n)$ も P 行列である.

(証明) (i) は行列式の性質より $\det(\boldsymbol{M})$=$\det(\boldsymbol{M}^T)$ なので明らかである.(ii) は $\boldsymbol{D}$ が対角行列なので,$\det(\boldsymbol{DMD})_{II}$=$\det(\boldsymbol{D}_{II})\det(\boldsymbol{M}_{II})\det(\boldsymbol{D}_{II})(I \subset N)$ が成り立つ. よって明らかである (iii) は行列式の多重線形性より明らかである.(多重線形性については以下参照)□

(行列式の多重線形性)[1]
$\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_j, \ldots, \boldsymbol{a}_n \in \mathbb{R}^n, \boldsymbol{a}_j^{'} \in \mathbb{R}^n, c \in \mathbb{R}$ とする. これらを列ベクトルとして n 個並べた行列の行列式について以下が成り立つ.

(i) $\det([\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, c\boldsymbol{a}_j, \ldots, \boldsymbol{a}_n])$=$c$·$\det([\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_j, \ldots, \boldsymbol{a}_n])$
(ある列が c 倍されてる時に c を外に出せる)
(ii) $\det([\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_j + \boldsymbol{a}_j^{'}, \ldots, \boldsymbol{a}_n])$=$\det([\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_j, \ldots, \boldsymbol{a}_n])$+$\det([\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_j^{'}, \ldots, \boldsymbol{a}_n])$
(ある列がベクトル同士の足し算になっているときに 2 つの行列式に分解して計算できる)

さらに P 行列についての以下の有名な必要十分条件がある.

定理 3.1 [1]$\boldsymbol{M} \in \mathbb{R}^n \times n$ とする.
(i)$\boldsymbol{M}$ が P 行列である ⇔(ii) 任意の $\boldsymbol{x}(\neq \boldsymbol{0}) \in \mathbb{R}^n$ に対して,$x_i(\boldsymbol{Mx})_i > 0$ となる $i \in N$ が存在する.

(証明) (⇒):帰納法で証明する.$n = 1$ の時,$\boldsymbol{M}$ は P 行列なので $J = 1$ は必ず正になるので明らかに成り立つ.

次に $n > 1$ のとき,$n > n'$ である n' に対して (i)⇒(ii) が成り立つと仮定する.

そのとき $n \times n$ の P 行列 $\boldsymbol{M}$ について,(i)⇒(ii) が成り立たないと仮定する. つまり $x_i(\boldsymbol{Mx})_i \leq 0(\forall i \in N)$ となる $\boldsymbol{x}(\neq \boldsymbol{0}) \in \mathbb{R}^n$ が存在するとする.

このとき x_r=0 となる r が存在するとき,$I := 1, 2, \ldots, n-r$ とし,$\boldsymbol{x}$ から第 r 要素を取り除いた $\boldsymbol{x}' \neq \boldsymbol{0}$ について,$x'_j(\boldsymbol{M}_{II}\boldsymbol{x}')_j \leq 0(\forall j \in I)$ が成り立つ. このとき,$\boldsymbol{M}_{II}, x'$ の次元は $n-1$ になっており n より低くなっている. このことから,$n > n'$ である n' に対して (i)⇒(ii) が成り立つという帰納法の仮定に矛盾する. よって $x_r = 0$ となる r は存在しない. すなわち $\boldsymbol{x}$ の要素に 0 は存在しない.

このことから $d_i := \frac{(\boldsymbol{Mx})_i}{x_i} \leq 0$ が定義できる. この d_i に関して,$(\boldsymbol{Mx}_i) = d_i x_i$ が成り立つ.$\boldsymbol{D} := \mathrm{diag}(d_1, d_2, \ldots, d_n)$ とすれば,$(\boldsymbol{M} - \boldsymbol{D})\boldsymbol{x} = \boldsymbol{0}$ を得る. 性質 3.2 の (iii) より $\boldsymbol{M} - \boldsymbol{D}$ もまた P 行列となるので $\det(\boldsymbol{M} - \boldsymbol{D}) > 0$ となる. よって $\boldsymbol{M} - \boldsymbol{D}$ は正則であり逆行列を持つ. よって,$(\boldsymbol{M} - \boldsymbol{D})\boldsymbol{x} = \boldsymbol{0}$ に左から $(\boldsymbol{M} - \boldsymbol{D})^{-1}$ をかけてやると $\boldsymbol{x} = \boldsymbol{0}$ となってしまい,$\boldsymbol{x}$ に 0 の要素が無いことに矛盾する. したがって (i)⇒(ii) が成り立たないとする仮定は誤りであり (i)⇒(ii) が成り立たつと言える.

(⇐):対偶を証明する. まず $\boldsymbol{M}$ が P 行列ではないとする. すなわち,$\det(\boldsymbol{M}_{II}) \leq 0$ とする.

$\det(\boldsymbol{M}_{II})$=0 となる $I \subseteq N$ が存在する場合:

$\boldsymbol{x}'_I \neq \boldsymbol{0}, \boldsymbol{M}_{II}\boldsymbol{x}'_I$=0 なる $\boldsymbol{x}'_I$ が存在する.$\boldsymbol{x}$ を $x_i = 0(i \notin I), x_i = x'_i(i \in I)$ とすれば $\boldsymbol{x}$ は $x_i(\boldsymbol{M}\boldsymbol{x})_i = 0(\forall i \in N)$ を満たす.

それ以外の場合:

$\det(M_{II}) < 0$ となる $I \subseteq N$ が存在するので, そのような I で極小のものを J とする. つまり $\det(M_{JJ}) < 0$ である. さらに任意の $J' \subset J$ に対して $\det(\boldsymbol{M}_{J'J'}) > 0$ であると置く.$J = j_1, j_2, \ldots, j_k$ であり, かつ $j_1 < j_2 < \cdots < j_k$ であるとする. 線形方程式 $\boldsymbol{M}_{JJ}\boldsymbol{x}'_J = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ を考える. クラメルの公式より $x'_{j_1} = \frac{(-1)^{(j_1+j_1)}\det(\boldsymbol{M}_{J'J'})}{\det(\boldsymbol{M}_{JJ})} < 0$ である. 但し $J' = J - \{j_1\}$ ここで $\boldsymbol{x}$ を $x_i = 0(i \notin J), x_i = x'_i(i \in J)$ とすれば $\boldsymbol{x} \neq 0$ であり,$x_i(\boldsymbol{M}\boldsymbol{x})_< 0(i = j_1)$,$x_i(\boldsymbol{M}\boldsymbol{x})_= 0(i \neq j_1)$ である. これらのことより (i)⇐(ii) の対偶が真であることが証明され,(i)⇐(ii) も証明された □

この定理から次の性質が成り立つ

性質 3.3 [1] 任意の P 行列 $\boldsymbol{M}$ は S 行列である.

(証明) まずこの性質を言い換えると以下のように書くことができる

$$\boldsymbol{M} \text{ が P 行列である} \Rightarrow \boldsymbol{M}\boldsymbol{x} > \boldsymbol{0} \ \ (\boldsymbol{x} > \boldsymbol{0}) \text{ となる } \boldsymbol{x} \text{ が存在する.}$$

ここから対偶を証明する.$\boldsymbol{M}$ が S 行列ではないとする. すると二者択一の定理より $Y_S := \{\boldsymbol{y} \in \mathbb{R}^n \| \boldsymbol{M}^T\boldsymbol{y} \leq \boldsymbol{0}, \boldsymbol{y} \geq \boldsymbol{0}, \boldsymbol{y} \neq \boldsymbol{0}\} \neq \emptyset$ すなわち,$(\boldsymbol{M}^T\boldsymbol{y}) \leq 0, \boldsymbol{y} \geq 0, \boldsymbol{y} \neq 0$ を満たすような $\boldsymbol{y} \in \mathbb{R}^n(\forall i \in N)$ が存在する. よって $y_i \geq 0$ であるので $\boldsymbol{y} = \boldsymbol{x}$ とおける. すると

$$y_i(\boldsymbol{M}^T\boldsymbol{y})_i \leq \boldsymbol{0} \Leftrightarrow x_i(\boldsymbol{M}^T\boldsymbol{x})_i \leq \boldsymbol{0}$$

となるので先の定理より $\boldsymbol{M}^T$ は P 行列ではなく,P 行列の性質から $\boldsymbol{M}$ も P 行列ではない. よって対偶が証明されたのでこの性質は真である.□

3.2.1 P 行列と LCP の関係

これまで紹介してきた P 行列と LCP の解の存在判定には重要な必要十分条件が成り立っている. それが以下の定理である.

定理 3.2 [1] $\boldsymbol{M}$ が P 行列であることの必要十分条件は, 任意の $\boldsymbol{q}$ に対して LCP($\boldsymbol{M}, \boldsymbol{q}$) が唯一解をもつことである.

(証明) (⇒):$\boldsymbol{M}$ を P 行列とする. 先述の性質より,$\boldsymbol{M}$ は S 行列である.S 行列の性質より, 任意のベクトル $\boldsymbol{q}$ に対して,

$$\{\boldsymbol{x} | \boldsymbol{M}\boldsymbol{x} + \boldsymbol{q}, \boldsymbol{x} \geq 0\}$$

は解を持つ. 性質 3.2 の問題は最適解 $\boldsymbol{x}^*$ を持ち, さらに補助定理 3.2 を満たし $\boldsymbol{u}^*$ が存在する. 補助定理 3.2 の (v) から $(\boldsymbol{x}^* - \boldsymbol{u}^*)_i(\boldsymbol{M}^T(\boldsymbol{x}^* - \boldsymbol{u}^*))_i \leq 0$ $(i = 1, 2, \ldots, n)$ が成り立つ. ここで $\boldsymbol{x}^* \neq \boldsymbol{u}^*$ とすると,$\boldsymbol{a} = \boldsymbol{x}^* - \boldsymbol{u}^*$ とおける. これを先の式に代入すると,$a_i(\boldsymbol{M}^T\boldsymbol{a})_i \leq 0$ となる. これは $\boldsymbol{M}$ が P 行列, すなわち $\boldsymbol{M}^T$ が P 行列であるための必要十分条件であるための式,$x_i(\boldsymbol{M}^T\boldsymbol{x})_i > 0$ に矛盾する. よって $\boldsymbol{x}^* = \boldsymbol{u}^*$ である.

$\boldsymbol{u}^* = \boldsymbol{x}^*$ であることと補助定理3.2の (iv) から $(\boldsymbol{x}^*)^T(\boldsymbol{q}+\boldsymbol{M}\boldsymbol{x}^*) = 0$ となり,$\boldsymbol{x}^*$ は LCP$(\boldsymbol{M}, \boldsymbol{q})$ の解となる.

次に唯一解であることを示すために $\boldsymbol{x}^*$ と異なる解 $\boldsymbol{x}'$ が存在するとする.$\boldsymbol{y}^* := \boldsymbol{M}\boldsymbol{x}^*+\boldsymbol{q}, \boldsymbol{y}' := \boldsymbol{M}\boldsymbol{x}' + \boldsymbol{q}$ とすると $\boldsymbol{y}^* - \boldsymbol{y}' = \boldsymbol{M}(\boldsymbol{x}^* - \boldsymbol{x}')$ が得られる.$i = 1, 2, \ldots, n$ に対して.

$$(\boldsymbol{x}^* - \boldsymbol{x}')_i(\boldsymbol{y}^* - \boldsymbol{y}')_i = (\boldsymbol{x}^* - \boldsymbol{x}')_i(\boldsymbol{M}(\boldsymbol{x}^* - \boldsymbol{x}'))_i$$

を考える. まず $\boldsymbol{x}^*, \boldsymbol{x}'$ は LCP の解なので $\boldsymbol{x}^* \cdot \boldsymbol{y}^* = 0, \boldsymbol{x}' \cdot \boldsymbol{y}' = 0$ を満たす. このことは $(\boldsymbol{x}^*, \boldsymbol{x}')(\boldsymbol{y}^*, \boldsymbol{y}')$ がそれぞれどちらも $\boldsymbol{0}$ か, それらから 1 つずつ選んだ $(\boldsymbol{x}^*, \boldsymbol{y})',(\boldsymbol{x}', \boldsymbol{y}^*)$ がそれぞれどちらも $\boldsymbol{0}$ かの 4 つのパターンが考えられる. ここで上記の式の左辺 $(\boldsymbol{x}^* - \boldsymbol{x}')_i(\boldsymbol{y}^* - \boldsymbol{y}')_i$ を考えると

$$(\boldsymbol{x}^* - \boldsymbol{x}')_i(\boldsymbol{y}^* - \boldsymbol{y}')_i = (\boldsymbol{x}^*)_i(\boldsymbol{y}^*)_i + (\boldsymbol{x}^*)_i(-\boldsymbol{y}')_i + (-\boldsymbol{x}')_i(\boldsymbol{y}^*)_i + (-\boldsymbol{x}')_i(-\boldsymbol{y}')_i$$

となり上記の 4 パターンに当てはめて考えると

(i) $\boldsymbol{x}_i^* = 0, \boldsymbol{x}_i' = 0$ の時: $(\boldsymbol{x}^* - \boldsymbol{x}')_i(\boldsymbol{y}^* - \boldsymbol{y}')_i = 0$

(ii) $\boldsymbol{y}_i^* = 0, \boldsymbol{y}_i' = 0$ の時: $(\boldsymbol{x}^* - \boldsymbol{x}')_i(\boldsymbol{y}^* - \boldsymbol{y}')_i = 0$

(iii) $\boldsymbol{x}_i^* = 0, \boldsymbol{y}_i' = 0$ の時: $\boldsymbol{x}_i^*, \boldsymbol{y}_i'$ の要素がある部分は $\boldsymbol{0}$ で $(-\boldsymbol{x}')_i(\boldsymbol{y}^*)_i$ のみが残り $\boldsymbol{x}^*, \boldsymbol{x}'$ は LCP の解より $\boldsymbol{x}', \boldsymbol{y}^* \geq \boldsymbol{0}$ から $(-\boldsymbol{x}')_i \leq 0$ なので, $(-\boldsymbol{x}')_i(\boldsymbol{y}^*)_i \leq 0$ となり $(\boldsymbol{x}^* - \boldsymbol{x}')_i(\boldsymbol{y}^* - \boldsymbol{y}')_i \leq \boldsymbol{0}$

(iv) $\boldsymbol{x}_i' = 0, \boldsymbol{y}_i^* = 0$ の時: (iii) と同様に $(\boldsymbol{x}^*)_i(-\boldsymbol{y}')_i \leq 0$ となるので $(\boldsymbol{x}^* - \boldsymbol{x}')_i(\boldsymbol{y}^* - \boldsymbol{y}')_i \leq \boldsymbol{0}$

これらのことから

$$0 \geq (\boldsymbol{x}^* - \boldsymbol{x}')_i(\boldsymbol{y}^* - \boldsymbol{y}')_i = (\boldsymbol{x}^* - \boldsymbol{x}')_i(\boldsymbol{M}(\boldsymbol{x}^* - \boldsymbol{x}'))_i$$

となるので $(\boldsymbol{x}^* - \boldsymbol{x}') = \boldsymbol{w}$ と置けば,$\boldsymbol{w}_i(\boldsymbol{M}\boldsymbol{w})_i \leq 0$ となりこれは $\boldsymbol{M}$ が P 行列であることの必要十分条件と矛盾する. よって解は唯一しか無い. 次に逆を証明する.

($\Leftarrow$):対偶が真であることを証明する.$\boldsymbol{M}$ が P 行列ではないとする.P 行列の必要十分条件より $x_i(\boldsymbol{M}\boldsymbol{x})_i \leq 0 (i = 1, 2, \ldots, n)$ となる $\boldsymbol{x} \neq 0$ が存在する. ここで $\boldsymbol{x}$ を次のように正の部分と負の部分で分割したものを定義する.$\boldsymbol{x}^+$ を $x_i^+ = \max(0, x_i)$,$\boldsymbol{x}^-$ を $x_i^- = \max(0, -x_i)$ とすれば $\boldsymbol{x}^+ \geq 0, -\boldsymbol{x}^- \leq 0$ となり,$\boldsymbol{x} = \boldsymbol{x}^+ - \boldsymbol{x}^-$ を満たしている.

同様に $\boldsymbol{y}$ も次のように分割する..$\boldsymbol{y}^+$ を $y_i^+ = \max(0, (\boldsymbol{M}\boldsymbol{x})_i)$,$\boldsymbol{y}^-$ を $y_i^- = \max(0, -(\boldsymbol{M}\boldsymbol{x})_i)$ とすれば $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{y}^+ - \boldsymbol{y}^-$ を満たす.

ここで LCP の条件式を変形した次の式 $\boldsymbol{q} = \boldsymbol{y} - \boldsymbol{M}\boldsymbol{x}$ に上記の式を代入して考えると

$$\begin{aligned}\boldsymbol{q} &= \boldsymbol{y} - \boldsymbol{M}\boldsymbol{x} \\ &= (\boldsymbol{y}^+ - \boldsymbol{y}^-) - (\boldsymbol{M}(\boldsymbol{x}^+ - \boldsymbol{x}^-)) \\ &= (\boldsymbol{y}^+ - \boldsymbol{M}\boldsymbol{x}^+) - (\boldsymbol{y}^- - \boldsymbol{M}\boldsymbol{x}^-)\end{aligned}$$

と書けるここで $\bar{\boldsymbol{q}} := (\boldsymbol{y}^+ - \boldsymbol{M}\boldsymbol{x}^+) = (\boldsymbol{y}^- - \boldsymbol{M}\boldsymbol{x}^-)$ と定義すると $\boldsymbol{x}^+, \boldsymbol{y}^+$ について $\boldsymbol{y}^+ = \boldsymbol{M}\boldsymbol{x}^+ - \bar{\boldsymbol{q}}$ が成り立ち,$\boldsymbol{x}^+, \boldsymbol{y}^+ \geq \boldsymbol{0}$ であり $\boldsymbol{M}$ がP行列でないこととP行列の必要十分条件から $x_i^+ \cdot y_i^+ = 0$ が成り立つ.$\boldsymbol{x}^-, \boldsymbol{y}^-$ についても同様のことが言える.

このことと $\boldsymbol{x} \neq 0$ であることから $\boldsymbol{x}^+, \boldsymbol{x}^-$ と,$\boldsymbol{y}^+, \boldsymbol{y}^-$ はそれぞれ異なり $(\boldsymbol{x}^+, \boldsymbol{x}^-)$ と $(\boldsymbol{y}^+, \boldsymbol{y}^-)$ はLCP$(\boldsymbol{M}, \bar{\boldsymbol{q}})$ の異なる2つの解となる. よってP行列でない $\boldsymbol{M}$ と任意のベクトルに対するLCPは異なる2つの解が存在することが言え, 対偶が証明されたので元々も真であることが言える.□

これらの話題を通してP行列とLCPの解の存在判定は密接な関わりがあることがわかる. しかし先に述べたようにP行列かどうかを判別するには 2^n 個の行列式を調べる必要があり実際には困難である. それを言っているのが以下である.

P行列問題
与えられた行列 $A \in \mathbb{R}^{n \times n}$ がP行列かどうかを決定する問題は,*co-NP*-**完全問題である**.
(1994，Coxson)

このことからLCPに解が存在するか判別するにP行列の判別が重要である反面,P行列を判別する効率のいい方法は存在しないだろうという予想がされている. そこで次の章ではP行列に迫るための方法の提案と考察を行っている.

3.3 LCPにおけるPrincipalPivotMethod

この章の最後にLCPを解くためのアルゴリズムであるPrincipalPivotMethod(PPM)の紹介をする. 基本的な流れとしては前章の改訂シンプレックス法と同様である. アルゴリズムを紹介する前にLCPの条件式の変形が必要なので行っておく.

まず以下の様な線形相補性問題

$$\mathrm{LCP}(\boldsymbol{M}, \boldsymbol{q}) \left| \begin{array}{lll} \text{find} & \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n & \\ \text{s.t.} & \boldsymbol{y} = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{q} & (\boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{y} \geq \boldsymbol{0}) \\ & x_i \cdot y_i = 0 & (i = 1, 2, \ldots, n) \end{array} \right.$$

の条件式である $\boldsymbol{y} = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{q}$ を $-\boldsymbol{M}\boldsymbol{x} + \boldsymbol{y} = \boldsymbol{q}$ と変形する.

次に $\boldsymbol{y}$ は $n \times n$ の単位ベクトル $\boldsymbol{I}$ との積,$\boldsymbol{I}\boldsymbol{y}$ と表現することができるので,

$$-\boldsymbol{M}\boldsymbol{x} + \boldsymbol{I}\boldsymbol{y} = \boldsymbol{q}$$

と表せる. さらに $\boldsymbol{x}' = [\boldsymbol{x}, \boldsymbol{y}]$($\boldsymbol{x}$ の下に $\boldsymbol{y}$ をくっつけたベクトル) として左辺を整理すると

$$[-\boldsymbol{M}, \boldsymbol{I}]\boldsymbol{x}' = \boldsymbol{q}$$

と書くことができる. このように置けば改訂シンプレックス法における添字集合 $B = n+1, n+2, \ldots, 2n$ で分割される行列は必ず単位ベクトルとなる.

つまりこの問題は目的関数が無く,$n \times 2n$ の行列 $[-\boldsymbol{M}, \boldsymbol{I}]$ と n 次元ベクトル $\boldsymbol{q}$ に関する線形計画問題とみなすことができる. このことを利用して以下のPrincipalPivotMethodが存在する.

[PrincipalPivotMethod:PPM]
入力:行列 $\boldsymbol{M}$ とベクトル $\boldsymbol{q}$
出力:最適基底解
初期化:$R := \{1, 2, \ldots, n\}, N := \{1, 2, \ldots, n\}; B := \{n+1, n+2, \ldots, 2n\}$
$\boldsymbol{A} := [-\boldsymbol{M}, \boldsymbol{I}]$
Step1:
$\boldsymbol{A}_B\bar{\boldsymbol{q}} = \boldsymbol{q}$ を $\bar{\boldsymbol{q}} \in \mathbb{R}^B$ について解く
Step2:
s を $\{j|\bar{q}_j < 0, j \in R\}$ からひとつ選ぶ. もし選べなかった場合,
すなわち $\bar{\boldsymbol{q}}$ の要素が全て正なら最適基底解を出力して終了.
Step3:
$\boldsymbol{A}_B\boldsymbol{d} = A_s$ を $\boldsymbol{d} \in \mathbb{R}^B$ について解く.
$\boldsymbol{d} \leq 0$ なら終了.
Step4(添字交換による辞書更新):
ピボット演算の代わりに次の操作を行う.
N の s 番目の要素と B の s 番目の要素を入れ替えたものを新たに N, B として Step1 に戻る

この PPM の計算量は $\boldsymbol{A}_B\bar{\boldsymbol{q}} = \boldsymbol{q}$ を $\bar{\boldsymbol{q}} \in \mathbb{R}^B$ について解くという, 線形方程式を解く程度で収まっており,$O(n^3 \sim n^4)$ 程度である.

前節で見たように入力された行列が P 行列ならばこのアルゴリズムは必ず解を持つが少ない時間で終わる保証はされていない.

4　隠れ行列を用いたP行列判別

この章では$co-NP$完全問題であり解くのが難しいとされているP行列に迫るための手法の提案と考察を行っている. ここでは先行研究であるPrdd行列という行列の各定理や一般化などの紹介をするとともに, 特に本論文の主題としてはZ^o行列と言うものに着目している.

4.1　P行列サンドイッチ

前章で紹介したようにP行列判定はco-NP完全問題であり, 解くのは難しいだろうと予想されている. 本研究ではそのようなP行列判別へのアプローチに以下のような上位クラスと部分クラスでP行列クラスを挟みこむようなP行列サンドイッチ (Morris and Namiki, 2007) というものに着目している.

$$L \subseteq \text{P行列クラス} \subseteq U$$

Lが部分クラスでUが上位クラスとしてP行列クラスを挟み込んでいるような形である. それらの説明をする前にまず基本行列クラスと呼ばれる行列を各種定義しておく.

4.2　基本行列クラス

この説ではP行列や近いP行列に近い行列クラスを探るために必要な各行列を定義する. まずはじめに本研究と関わりが深い行列として先行研究で用いられたPrdd行列を紹介する. 次に本研究で着目したZ^o行列と呼ばれるものを紹介する.

4.2.1　Prdd行列

ここではまず先行研究としてPrdd行列というものを紹介する

定義 4.1 (基本行列:Prdd-1行列,Prdd-∞行列)[1]

1. $\boldsymbol{C} \in \mathbb{R}^{n\times n}$が次の式を満たすとき$\boldsymbol{C}$をPrdd-1行列という

$$C_{ii} > \sum_{j\neq i} |C_{ij}| \quad (i \in N = \{1, 2, \ldots, n\})$$

2. $\boldsymbol{C} \in \mathbb{R}^{n\times n}$が次の式を満たすとき$\boldsymbol{C}$をPrdd-$\infty$行列という

$$C_{ii} > |C_{ij}| \quad (i \in N = \{1, 2, \ldots, n\})$$

次にこれらの具体例を示す

例 4.1 (Prdd-1の例)

$$\boldsymbol{C} = \begin{bmatrix} 5 & 1 & -2 \\ 1 & 4 & 2 \\ 0 & 1 & 3 \end{bmatrix}$$

このような行列は Prdd-1 行列である. なぜなら各行ベクトルの対角成分を除いた成分の絶対値の和が対角成分より小さくなっているからである.
(Prdd-∞ の例)

$$\boldsymbol{C} = \begin{bmatrix} 5 & 4 & 3 \\ 5 & 8 & -4 \\ 6 & 3 & 7 \end{bmatrix}$$

このような行列は Prdd-∞ 行列である. なぜなら各行ベクトルの対角成分を除いた成分の絶対値が対角成分より小さくなっているからである.

4.2.2 Z^o 行列

次に本研究で着目している Z^o 行列と呼ばれる行列を紹介する.

定義 4.2 (基本行列:Z^o 行列,Z^o+ 行列,Z^o- 行列)

1. $\boldsymbol{C} \in \mathbb{R}^{n\times n}$ が次の式を満たすとき $\boldsymbol{C}$ を Z^o 行列という

$$C_{ii} > 0, C_{ij} < 0 \ \ (i \in N = \{1, 2, \ldots, n\}, i \neq j)$$

2. $\boldsymbol{C} \in \mathbb{R}^{n\times n}$ が次の式を満たすとき $\boldsymbol{C}$ を Z^o+ 行列という

$$C_{ii} > 0, C_{ij} < 0 \ \ (i \in N = \{1, 2, \ldots, n\}, i \neq j), \sum_{j=1}^{n} C_{ij} < 0$$

3. $\boldsymbol{C} \in \mathbb{R}^{n\times n}$ が次の式を満たすとき $\boldsymbol{C}$ を Z^o- 行列という

$$C_{ii} > 0, C_{ij} < 0 (i \in N = \{1, 2, \ldots, n\}, i \neq j), \sum_{j=1}^{n} C_{ij} > 0$$

これら行列についても以下例を示す.

例 4.2 (Z^o 行列の例)

$$\boldsymbol{C} = \begin{bmatrix} 3 & -5 & -2 \\ -9 & 6 & -5 \\ -3 & -1 & 7 \end{bmatrix}$$

このような行列は Z^o 行列である. なぜなら対角成分が正で非対角成分が負だからである.
(Z^o+ 行列の例)

$$\boldsymbol{C} = \begin{bmatrix} 1 & -1 & -2 \\ -2 & 3 & -5 \\ -3 & -1 & 2 \end{bmatrix}$$

このような行列は Z^o+ 行列である. なぜなら対角成分が正で非対角成分が負であり, 行ベクトルごとの和がどれも負になるからである.
(Z^o- 行列の例)

$$\boldsymbol{C} = \begin{bmatrix} 4 & -1 & -2 \\ -2 & 5 & -1 \\ -1 & -1 & 6 \end{bmatrix}$$

このような行列は Z^o+ 行列である. なぜなら対角成分が正で非対角成分が負であり, 行ベクトルごとの和がどれも正になるからである.

ここで今後の為に各基本行列全体の集合を $K_{\mathrm{prdd}-1}, K_{\mathrm{prdd}-\infty}, K_{Z^o+}, K_{Z^o-}$ とする.

4.2.3 基本行列クラスに関する各種定理や定義

これら基本行列に対して以下の様な補助定理が成り立つ.

補助定理 4.1 [1] $\boldsymbol{x} \in \mathbb{R}^n, i \in N = \{1, 2, \ldots, n\}$ とする. 次が成り立つ. ただし,$\boldsymbol{e_j}$ は第 j 成分が 1 でそれ以外は 0 であるような n 次元ベクトルである.

1. $x_i > \sum_{j \neq i} |x_j|$ が成り立つための必要十分条件は, 任意の $j(\neq i)$ に対して,$\alpha_j, \beta_j > 0$ が存在し,$\boldsymbol{x} = \sum_{j \neq i} \{\alpha_j(\boldsymbol{e_i} + \boldsymbol{e_j}) + \beta_j(\boldsymbol{e_i} - \boldsymbol{e_j})\}$ と表せることである.

2. $x_i > |x_j| (i \neq j)$ が成り立つための必要十分条件は,$J \subseteq N \backslash \{i\}$ を満たす J に対して α_J が存在し,$\boldsymbol{x} = \sum_J \{\alpha_J(\sum_{j \notin J} \boldsymbol{e_j} - \sum_{j \in J} \boldsymbol{e_j})\}$ と表せることである.

3. $x_i > 0, x_j < 0 (j \neq i), \sum_{j=1}^{n} x_j < 0$ が成り立つための必要十分条件は, 任意の $i, j \in N (i \neq j)$ に対して $\alpha_j, \beta_j > 0$ が存在し,$\boldsymbol{x} = \sum_{j \neq i} \{\alpha_j(\boldsymbol{e_i} - \boldsymbol{e_j}) + \beta_j(-\boldsymbol{e_j})\}$ と表せることである.

4. $x_i > 0, x_j < 0 (j \neq i), \sum_{j=1}^{n} x_j > 0$ が成り立つための必要十分条件は, 任意の $i, j \in N (i \neq j)$ に対して $\alpha_j, \beta_i > 0$ が存在し,$\boldsymbol{x} = \sum_{j \neq i} \{\alpha_j(\boldsymbol{e_i} - \boldsymbol{e_j}) + \beta_i \boldsymbol{e_i}\}$ と表せることである.

(証明) 一つ証明すればいずれも同じ方法で証明できるので 1 のみ証明する.
($\Leftarrow$):第 i 成分の係数は $\sum_{j\neq i}(\alpha_j+\beta_j)$ となり, 第 j 成分の係数は $(\beta_j-\alpha_j)$ となり,$\alpha_j,\beta_j>0$ なので明らかに成り立つ.
($\Rightarrow$):n に関する帰納法で証明する.$n=2$ の時 $i=1$ としてよい.$x_1>\sum_{j\neq i}|x_j|$ なので $n=2$ より $x_1>|x_2|$ ならば,$x_2\leq 0$ と $x_2\geq 0$ の時を合わせて考えることで

$$\boldsymbol{e_1}+\boldsymbol{e_2}=\begin{bmatrix}1\\1\end{bmatrix},\ \boldsymbol{e_1}-\boldsymbol{e_2}=\begin{bmatrix}1\\-1\end{bmatrix}$$

の非負の線形結合で表される. つまり,

$$\begin{bmatrix}x_1\\x_2\end{bmatrix}=\sum_{j\neq i}\{\alpha_j(\boldsymbol{e_i}+\boldsymbol{e_j})+\beta_j(\boldsymbol{e_i}-\boldsymbol{e_j})\},\alpha_j,\beta_j>0(j\neq i)$$

となりことは明らかである.

次に $n>n'$ である n' に対して主張が成り立っていると仮定する. さらに $\boldsymbol{x}\in\mathbb{R}^n$ で $x_i>\sum_{j\neq i}|x_j|$ を満たすと仮定する.$x_i>\sum_{j\neq i}|x_j|$ なので, 十分小さな $\epsilon>0$ を取れば,$x_i-\epsilon>\sum_{j\neq i}|x_j|$ を満たす. さらに $r\in N$ について $|x_r|=\min\{|x_j||j\in N\}$,$J:=\{j\in N|x_j<0\}$ とする. また $\boldsymbol{d}\in\mathbb{R}^n$ を,

$$d_j:=\begin{cases}n-1 & (j=i)\\ 1 & (j\notin J, j\neq i)\\ -1 & (j\in J)\end{cases}$$

とし,$\boldsymbol{x}':=\boldsymbol{x}-\epsilon\boldsymbol{e}_i-|x_r|\boldsymbol{d}$ とする. この時 $x'_r=0$ となる. また j が J に含まれるかどうかで場合分けして考えると,$|x_r|$ の定義より

$$|x'_j|=\begin{cases}-(x_j+|x_r|) & (j\in J)\\ x_j-|x_r| & (j\notin J)\end{cases}$$

となる. よって

$$\sum_{j\neq i,r}|x'_j|=\sum_{j\in J}\{-(x_j+|x_r|)\}+\sum_{j\notin J}(x_j-|x_r|)$$

ただし $j\in J$ の時 $x_j<0$ より $-x_j>0$ なので $-x_j=|x_j|$ より,

$$\sum_{j\neq i,r}|x'_j|=\sum_{j\in J}(|x_j|-|x_r|)+\sum_{j\notin J}(|x_j|-|x_r|)$$

これらのことから

$$\begin{aligned}x'_i-\sum_{j\neq i,r}|x'_j| &= x_i-\epsilon-|x_r|(n-1)-\{\sum_{j\in J}(|x_j|-|x_r|)+\sum_{j\notin J}(|x_j|-|x_r|)\}\\ &= x_i-\sum_{j\neq i}|x_j|-\epsilon\end{aligned}$$

$x_i - \epsilon > \sum_{j\neq i}|x_j|$ の仮定から $x_i - \sum_{j\neq i}|x_j| - \epsilon > 0$ すなわち $x'_i > \sum_{j\neq i,r}|x'_j|$ であることがわかる.

次に x'' を x' から第 r 成分を取り除いたものとすれば, 次元は下がるので帰納法の仮定より

$$x'' = \sum_{j\neq i,r}\{\alpha'_j(\boldsymbol{e'_i} + \boldsymbol{e'_j}) + \beta'_j(\boldsymbol{e'_i} - \boldsymbol{e'_j})\}$$

と表せる. ただし $\boldsymbol{e'_i}, \boldsymbol{e'_j}$ は $\boldsymbol{e_i}, \boldsymbol{e_j}$ から第 r 成分を取り除いたものである.$x_r = 0$ なので

$$\boldsymbol{x'} = \boldsymbol{x} - \epsilon\boldsymbol{e_i} - |x_r|\boldsymbol{d} = \sum_{j\neq i,r}\{\alpha'_j(\boldsymbol{e_i} + \boldsymbol{e_j}) + \beta'_j(\boldsymbol{e_i} - \boldsymbol{e_j})\}$$

がなりたつ. また $\boldsymbol{e_i} = \frac{1}{2}\{(\boldsymbol{e_i} + \boldsymbol{e}_r) + (\boldsymbol{e_i} - \boldsymbol{e_r})\}, \boldsymbol{d} = \sum_{j\notin J\cup\{i\}}(\boldsymbol{e_i} + \boldsymbol{e_j}) + \sum_{j\in J}(\boldsymbol{e_i} - \boldsymbol{e_j})$ とかける ので

$$\alpha_r := \tfrac{\epsilon}{2}, \beta_r := \tfrac{\epsilon}{2}, \alpha_j := \alpha'_j + |x_r|(j \notin J\cup\{i\}), \beta_j := \beta'_j + 0(j \notin J\cup\{i\}),$$
$$\alpha_j := \alpha'_j + 0(j \in J), \beta_j := \beta'_j + |x_r|(j \in J)$$

とおけば,

$$\begin{aligned}
\boldsymbol{x} &= \boldsymbol{x'} + \epsilon\boldsymbol{e_i} + |x_r|\boldsymbol{d} \\
&= \sum_{j\neq i,r}\{\alpha'_j(\boldsymbol{e_i} + \boldsymbol{e_j}) + \beta'_j(\boldsymbol{e_i} - \boldsymbol{e_j})\} + \frac{\epsilon}{2}\{(\boldsymbol{e_i} + \boldsymbol{e}_r) + (\boldsymbol{e_i} - \boldsymbol{e_r})\} \\
&\quad + |x_r|\sum_{j\notin J\cup\{i\}}(\boldsymbol{e_i} + \boldsymbol{e_j}) + |x_r|\sum_{j\in J}(\boldsymbol{e_i} - \boldsymbol{e_j}) \\
&= \sum_{j\notin J\cup\{i\}}\{(\alpha'_j + |x_r|)(\boldsymbol{e_i} + \boldsymbol{e_j})\} + \sum_{j\in J}\{(\alpha'_j + 0)(\boldsymbol{e_i} + \boldsymbol{e_j}) \\
&\quad + \sum_{j\in J}\{(\beta'_j + |x_r|)(\boldsymbol{e_i} - \boldsymbol{e_j})\} + \sum_{j\notin J\cup\{i\}}\{(\beta'_j + 0)(\boldsymbol{e_i} - \boldsymbol{e_j}) \\
&\quad + \tfrac{\epsilon}{2}\{(\boldsymbol{e_i} + \boldsymbol{e}_r) + (\boldsymbol{e_i} - \boldsymbol{e_r})\} + \sum_{j\notin J\cup\{i\}}|x_r|(\boldsymbol{e_i} + \boldsymbol{e_j}) + \sum_{j\in J}|x_r|(\boldsymbol{e_i} - \boldsymbol{e_j}) \\
&= \sum_{j\neq i,r}\{\alpha_j(\boldsymbol{e_i} + \boldsymbol{e_j})\} + \sum_{j\neq i,r}\{\beta_j(\boldsymbol{e_i} - \boldsymbol{e_j})\} + \alpha_r(\boldsymbol{e_i} + \boldsymbol{e_j}) + \beta_r(\boldsymbol{e_i} - \boldsymbol{e_j}) \\
&= \sum_{j\neq i}\{\alpha_j(\boldsymbol{e_i} + \boldsymbol{e_j}) + \beta_j(\boldsymbol{e_i} - \boldsymbol{e_j})\}
\end{aligned}$$

が成り立つ. よって題意は示された.□

この補助定理より各基本行列の端線行列表示である以下の性質が得られる. ただし $E_{ij} \in \mathbb{R}^{n\times n}$ は (i,j) 成分が 1 でそれ以外が 0 の行列である.

命題 4.1 $\boldsymbol{C} \in \mathbb{R}^{n\times n}, N = \{1, 2, \ldots, n\}$ とする.

1. $\boldsymbol{C} \in K_{\mathrm{prdd}-1}$ が成り立つための必要十分条件は, 任意の $i, j(i, j \in N, i \neq j)$ に対して,$\alpha_{ij}, \beta_{ij} > 0$ が存在し,$\boldsymbol{C} = \sum_{j\neq i}\{\alpha_{ij}(\boldsymbol{E_{ii}} - \boldsymbol{E_{ij}}) + \beta_{ij}(\boldsymbol{E_{ii}} + \boldsymbol{E_{ij}})\}$ と表せることである.

2. $\boldsymbol{C} \in K_{\mathrm{prdd}-\infty}$ が成り立つための必要十分条件は,$J \subseteq N\backslash\{i\}$ を満たす任意のペア (i, J) に対して $\alpha_{(i,J)}$ が存在し,$\boldsymbol{C} = \sum_{i\notin J}\{\alpha_{(i,J)}(\sum_{j\notin J}\boldsymbol{E_{ij}} - \sum_{j\in J}\boldsymbol{E_{ij}})\}$ と表せることである.

3. $\boldsymbol{C} \in K_{Z^o+}$ が成り立つための必要十分条件は, 任意の $i, j \in N(i \neq j)$ に対して $\alpha_{ij}, \beta_{ij} > 0$ が存在し,$\boldsymbol{C} = \sum_{j\neq i}\{\alpha_{ij}(\boldsymbol{E_{ii}} - \boldsymbol{E_{ij}}) + \beta_{ij}(-\boldsymbol{E_{ij}})\}$ と表せることである.

4. $\boldsymbol{C} \in K_{Z^o-}$ が成り立つための必要十分条件は, 任意の $i, j \in N(i \neq j)$ に対して $\alpha_{ij}, \beta_i > 0$ が存在し,$\boldsymbol{C} = \sum_{j\neq i}\{\alpha_{ij}(\boldsymbol{E_{ii}} - \boldsymbol{E_{ij}}) + \beta_i\boldsymbol{E_{ii}}\}$ と表せることである.

(証明) 補助定理 4.1 よりあきらかである

4.3 基本行列の隠れ行列

続いて各基本行列の隠れ行列と呼ばれるものを導入する. これらを用いて P 行列サンドイッチを作る.

定義 4.3 [1](隠れ Prdd-1 行列, 隠れ Prdd-∞ 行列)
行列 $\boldsymbol{A}$ が隠れ (hidden)prdd-1(prdd-∞) 行列であるとは,$\boldsymbol{AC} = \boldsymbol{B}$ となる
$\boldsymbol{C}, \boldsymbol{B} \in K_{\mathrm{prdd}-1}(K_{\mathrm{prdd}\infty})$,$\boldsymbol{C}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ が存在することである.

定義 4.4 [1](隠れ Z^o- 行列, 隠れ Z^o+ 行列)
行列 $\boldsymbol{A}$ が隠れ $Z^o-(Z^o+)$ 行列であるとは,$\boldsymbol{AC} = \boldsymbol{B}$ となる $\boldsymbol{C}, \boldsymbol{B} \in K_{Z^o-}(K_{Z^o+})$,
$\boldsymbol{C}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ が存在することである.

さらにこれら隠れ行列について以下のような二者択一の定理が成り立つ.

定理 4.1 [1](隠れ prdd-1 の二者択一の定理)
$\boldsymbol{A} \in \mathbb{R}^{n\times n}$ に対して次のいずれか一方かつ一方のみが成り立つ.

(1) $\boldsymbol{AC} = \boldsymbol{B}$ となる $\boldsymbol{C}, \boldsymbol{B} \in K_{\mathrm{prdd}-1}$,$\boldsymbol{C}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ が存在する

(2) $\boldsymbol{R} + \boldsymbol{A}^T\boldsymbol{S} = \boldsymbol{O}$ かつ $R_{ii} \geq |R_{ij}|, S_{ii} \geq |S_{ij}|$ となる $(\boldsymbol{R}, \boldsymbol{S})(\neq (\boldsymbol{O}, \boldsymbol{O}))$,$\boldsymbol{R}, \boldsymbol{S} \in \mathbb{R}^{n\times n}$ が存在する

定理 4.2 [1](隠れ prdd-∞ の二者択一の定理)
$\boldsymbol{A} \in \mathbb{R}^{n\times n}$ に対して次のいずれか一方かつ一方のみが成り立つ.

(1) $\boldsymbol{AC} = \boldsymbol{B}$ となる $\boldsymbol{C}, \boldsymbol{B} \in K_{\mathrm{prdd}-1}$,$\boldsymbol{C}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ が存在する

(2) $\boldsymbol{R} + \boldsymbol{A}^T\boldsymbol{S} = \boldsymbol{O}$ かつ $G \subseteq N\backslash\{i\}$ を満たす (i, G) について $\sum_{j\notin G}R_{ij} - \sum_{j\in G}R_{ij} \geq 0$, $\sum_{j\notin G}S_{ij} - \sum_{j\in G}S_{ij} \geq 0$ となる $(\boldsymbol{R}, \boldsymbol{S})(\neq (\boldsymbol{O}, \boldsymbol{O}))$,$\boldsymbol{R}, \boldsymbol{S} \in \mathbb{R}^{n\times n}$ が存在する

定理 4.3 (隠れ Z^o- の二者択一の定理)
$\boldsymbol{A} \in \mathbb{R}^{n\times n}$ に対して次のいずれか一方かつ一方のみが成り立つ.

(1) $\boldsymbol{AC} = \boldsymbol{B}$ となる $\boldsymbol{C}, \boldsymbol{B} \in K_{Z^o-}$,$\boldsymbol{C}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ が存在する

(2) $\boldsymbol{R} + \boldsymbol{A}^T\boldsymbol{S} = \boldsymbol{O}$ かつ $R_{ii} \geq R_{ij}, R_{ii} \geq 0, S_{ii} \geq S_{ij}, S_{ii} \geq 0$ となる $(\boldsymbol{R}, \boldsymbol{S})(\neq (\boldsymbol{O}, \boldsymbol{O}))$,$\boldsymbol{R}, \boldsymbol{S} \in \mathbb{R}^{n\times n}$ が存在する

定理 4.4 (隠れ Z^o+ の二者択一の定理)
$\boldsymbol{A} \in \mathbb{R}^{n\times n}$ に対して次のいずれか一方かつ一方のみが成り立つ.

(1) $\boldsymbol{AC} = \boldsymbol{B}$ となる $\boldsymbol{C}, \boldsymbol{B} \in K_{Z^o+}$,$\boldsymbol{C}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ が存在する

(2) $\boldsymbol{R} + \boldsymbol{A}^T\boldsymbol{S} = \boldsymbol{O}$ かつ $R_{ii} \geq R_{ij}, R_{ij} \leq 0, S_{ii} \geq S_{ij}, S_{ij} \leq 0$ となる $(\boldsymbol{R}, \boldsymbol{S})(\neq (\boldsymbol{O}, \boldsymbol{O}))$,$\boldsymbol{R}, \boldsymbol{S} \in \mathbb{R}^{n\times n}$ が存在する

(証明) $\boldsymbol{C}, \boldsymbol{B}$ を行列では無く n^2 次元のベクトルとして考え, 制約条件 $\boldsymbol{AC} = \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{B} \in K_{\mathrm{prdd}-1}(K_{\mathrm{prdd}\infty}, K_{Z^o-}, K_{Z^o+})$ を線形等式や線形不等式で表し, そこに Tucker の二者択一の定理を適用すると主張が得られる.□

以下に隠れ prdd-1 の時の例を示す

例 4.3 例として $n = 2$ の時を考える.$\boldsymbol{A}$ が隠れ prdd-1 行列であるための条件 $\boldsymbol{AC} = \boldsymbol{B}$ について,$\hat{\boldsymbol{A}}$ を次のように定める.

$$\hat{\boldsymbol{A}} := \begin{bmatrix} a_{11} & 0 & a_{12} & 0 \\ 0 & a_{11} & 0 & a_{12} \\ a_{21} & 0 & a_{22} & 0 \\ 0 & a_{21} & 0 & a_{22} \end{bmatrix}, \hat{\boldsymbol{c}} := \begin{bmatrix} c_{11} \\ c_{12} \\ c_{21} \\ c_{22} \end{bmatrix}, \hat{\boldsymbol{b}} := \begin{bmatrix} b_{11} \\ b_{12} \\ b_{21} \\ b_{22} \end{bmatrix}$$

のように $\boldsymbol{c}, \boldsymbol{B}$ をベクトルとして考えれば,$\hat{\boldsymbol{A}}\boldsymbol{c} = \hat{\boldsymbol{b}}$ という線形方程式として表すことができる. さらに $\boldsymbol{C}, \boldsymbol{B}$ が prdd-1 行列であるという条件についても

$$\hat{\boldsymbol{X}} := \begin{bmatrix} 1 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \hat{\boldsymbol{w}}_C := \begin{bmatrix} c \\ y_{12} \\ c \\ y_{21} \\ c \\ z_{12} \\ c \\ z_{21} \end{bmatrix}, \hat{\boldsymbol{w}}_B := \begin{bmatrix} b \\ y_{12} \\ b \\ y_{21} \\ b \\ z_{12} \\ b \\ z_{21} \end{bmatrix}$$

とすることで,prdd-1 の条件は $\hat{\boldsymbol{c}} = \hat{\boldsymbol{X}}\hat{\boldsymbol{w}}_C, \hat{\boldsymbol{w}}_C > \boldsymbol{0}, \hat{\boldsymbol{b}} = \hat{\boldsymbol{X}}\hat{\boldsymbol{w}}_B, \hat{\boldsymbol{w}}_B > \boldsymbol{0}$ と表すことができる. ただし $\hat{\boldsymbol{X}}$ は prdd-1 の単線行列である $\boldsymbol{E}_{ii} - \boldsymbol{E}_{ij}, \boldsymbol{E}_{ii} + \boldsymbol{E}_{ij}$ をベクトルとして考え, 順番に並べたものである. これらをまとめて

$$\bar{\boldsymbol{A}} := \begin{bmatrix} \hat{\boldsymbol{A}} & -\boldsymbol{I} & \boldsymbol{O} & \boldsymbol{O} \\ \boldsymbol{I} & \boldsymbol{O} & -\hat{\boldsymbol{X}} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} & \boldsymbol{O} & -\hat{\boldsymbol{X}} \end{bmatrix}, \bar{\boldsymbol{x}} = [\hat{\boldsymbol{c}}, \hat{\boldsymbol{b}}, \hat{\boldsymbol{w}}_C, \hat{\boldsymbol{w}}_B]^T$$

とすれば $\boldsymbol{A}$ が隠れ prdd-1 行列であるかどうかは $\bar{\boldsymbol{A}}\bar{\boldsymbol{x}} = \boldsymbol{0}, \hat{\boldsymbol{w}}_C > 0, \hat{\boldsymbol{w}}_B > 0$ が解を持つかどうかで判別可能であり, これに Tucker の二者択一の定理を適用すると主張を得る.

この二者択一の定理で重要なことは隠れ行列の判別は多項式時間で可能であるということを示しているということである.P 行列が co-NP 困難なのに対してその周辺の行列クラスは多項式時間で判別可能だということが重要である.

次にこれら隠れ行列に関して重要な定理を幾つか紹介する. まず準備のために P 行列と同値な条件である以下の補助定理を証明する.

補助定理 4.2 [1] $\boldsymbol{A} \in \mathbb{R}^{n\times n}, N = \{1, 2, \ldots, n\}$ とする. 次の 3 つは等価である.

(1) $\boldsymbol{A}$ が P 行列である.

(2) 任意の $\boldsymbol{x}(\neq \boldsymbol{0}) \in \mathbb{R}^n$ に対して $x_i(\boldsymbol{A}\boldsymbol{x})_i > 0$ となる $i \in N$ が存在する.

(3) 任意の $G \subseteq N$ に対して, 以下を満たすような $\boldsymbol{x}^G \in \mathbb{R}^n$ が存在する:$x_i^G < 0 \text{and} (\boldsymbol{A}\boldsymbol{x}^G)_i < 0 (i \in G), x_i^G > 0 \text{and} (\boldsymbol{A}\boldsymbol{x}^G)_i > 0 (i \notin G)$

(証明) (1) と (2) の等価性は定理 3.1 ですでに示してあるのでここでは (2) と (3) の等価性を示す.

(2)⇒(3):(2) が成り立つと仮定し,$G \subseteq N$ に対して $\boldsymbol{D}^G$ を $D_{ii}^G = -1 (i \in G), D_{ii}^G = 1 (i \notin G)$ を満たす対角行列とする.P 行列の性質より $\boldsymbol{D}^G\boldsymbol{A}\boldsymbol{D}^G$ は P 行列であり,P 行列は S 行列であるという性質から S 行列でもある. したがって正のベクトル $\boldsymbol{x}^G > \boldsymbol{0}$ が存在し,$\boldsymbol{D}^G\boldsymbol{A}\boldsymbol{D}^G\boldsymbol{x}^G > \boldsymbol{0}$ である. このことから $(\boldsymbol{D}^G\boldsymbol{x}^G)_i < 0, (\boldsymbol{A}\boldsymbol{D}^G\boldsymbol{x}^G)_i < 0 (i \in G)$,$(\boldsymbol{D}^G\boldsymbol{x}^G)_i > 0, (\boldsymbol{A}\boldsymbol{D}^G\boldsymbol{x}^G)_i > 0 (i \notin G)$ となるので (3) が成り立っている.

(3)⇒(2):(2) が成り立たないとして背理法で証明する.(2) と (1) が等価なことはすでに証明済みであるので,$\boldsymbol{A}$ は P 行列ではない. さらに $\boldsymbol{A}^T$ も P 行列ではないし,(2) は成り立たない.

$\boldsymbol{x} \neq \boldsymbol{0}$ を $x_i(\boldsymbol{A}^T\boldsymbol{x})_i \leq 0 (i \in N)$ を満たすベクトルとし,$G := \{i | x_i < 0\}$ とする. ここでこの G に対して (3) が成り立つとすると,$y_i^G < 0$,$(\boldsymbol{A}\boldsymbol{y}^G)_i < 0 (i \in G)$ と,$y_i^G > 0 (\boldsymbol{A}\boldsymbol{y}^G)_i > 0 (i \notin G)$ を満たすような $\boldsymbol{y}$ が存在する. すると $\boldsymbol{x}^T(\boldsymbol{A}\boldsymbol{y}) > 0$ と $\boldsymbol{y}^T(\boldsymbol{A}^T\boldsymbol{x}) \leq 0$ を同時に得るがこれは矛盾である. よって仮定が間違っていたことになり背理法より題意は示された.□

この補助定理と二者択一の定理より次の包含関係が示される.

定理 4.5 [1] 任意の隠れ prdd-1 行列は P 行列である.

(証明) $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ を隠れ prdd-1 行列であるとする. 二者択一の定理より,$\boldsymbol{A}\boldsymbol{C} = \boldsymbol{B}$ をとなるような prdd-1 行列 $\boldsymbol{C}, \boldsymbol{B}$ が存在する. ここで,$G \subseteq N$ に対して,$\boldsymbol{e}^G$ を $e_i^G = -1 \ (i \in G)$,$e_i^G = 1 \ (i \notin G)$ を満たすベクトルとすると,$\boldsymbol{C}$ と $\boldsymbol{B}$ は prdd-1 行列であることより,$(\boldsymbol{C}\boldsymbol{e}^G)_i < 0 \ (i \in G)$, $(\boldsymbol{C}\boldsymbol{e}^G)_i > 0 \ (i \notin G)$ を満たす. さらに,$(\boldsymbol{A}\boldsymbol{C}\boldsymbol{e}^G)_i = (\boldsymbol{B}\boldsymbol{e}^G)_i < 0 \ (i \in G)$, $(\boldsymbol{A}\boldsymbol{C}\boldsymbol{e}^G)_i = (\boldsymbol{B}\boldsymbol{e}^G)_i > 0 \ (i \notin G)$ を満たす. このとき, ベクトル $(\boldsymbol{C}\boldsymbol{e}^G)$ は補助定理 4.2 の (3) における $\boldsymbol{x}^G$ の役割を果たしていることがわかる. したがって,(1) と等価であるので, 行列 $\boldsymbol{A}$ は P 行列である.□

定理 4.6 [1] 任意の P 行列は隠れ prdd-∞ 行列である.

(証明) $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ が prdd-∞ 行列でないとして対偶を示す. 二者択一の定理のより $\boldsymbol{R}+\boldsymbol{A}^T\boldsymbol{S} = \boldsymbol{0}$ かつ,$G \subseteq N\backslash\{i\}$ を満たす (i,G) について $\sum_{j\notin G} R_{ij} - \sum_{j\in G} R_{ij} \geq 0$, $\sum_{j\notin G} S_{ij} - \sum_{j\in G} S_{ij} \geq 0$ を満たす行列 $\boldsymbol{R},\boldsymbol{S}$ が存在する. 但し,$(\boldsymbol{R},\boldsymbol{S}) \neq (\boldsymbol{0},\boldsymbol{0})$ である.

$\boldsymbol{R},\boldsymbol{S}$ がともに $\boldsymbol{0}$ ではないという条件と,$\boldsymbol{R} = -\boldsymbol{A}^T\boldsymbol{S}$ より,$\boldsymbol{S} = \boldsymbol{0}$ とすると $\boldsymbol{R},\boldsymbol{S} = (\boldsymbol{0},\boldsymbol{0})$ となるので $\boldsymbol{S} \neq \boldsymbol{0}$ であることがわかる.

$\boldsymbol{S} \neq \boldsymbol{0}$ より,$\sum_{j\notin G} S_{ij} - \sum_{j\in G} S_{ij} \geq 0$ のうち, 少なくとも一つの不等式において, 等号を満たさないことがわかる, その場合の G を G^* とする. 即ち, $\sum_{j\notin G^*} S_{ij} - \sum_{j\in G^*} S_{ij} > 0$ が成り立つとする. ここで,$\boldsymbol{x} = \boldsymbol{S}\boldsymbol{e}^{G^*}$ とする. ただし, $\boldsymbol{e}^{G^*}$ は, $e_i^{G^*} = -1\ (i \in G^*)$,$e_i^{G^*} = 1\ (i \notin G^*)$ を満たすベクトルとする. すると $i \notin G^*$ に対しては,$x_i = (\boldsymbol{S}\boldsymbol{e}^{G^*})_i \geq 0$, $(\boldsymbol{A}^T\boldsymbol{x})_i = (\boldsymbol{A}^T\boldsymbol{S}\boldsymbol{e}^{G^*})_i = -(\boldsymbol{R}\boldsymbol{e}^{G^*}) \leq 0$ が成り立ち, $i \in G^*$ に対しては, $x_i = (\boldsymbol{S}\boldsymbol{e}^{G^*})_i \leq 0$, $(\boldsymbol{A}^T\boldsymbol{x})_i = (\boldsymbol{A}^T\boldsymbol{S}\boldsymbol{e}^{G^*})_i = -(\boldsymbol{R}\boldsymbol{e}^{G^*}) \geq 0$ が成り立つ. このことは, 補助定理 4.2 の (2) を満たさず,$\boldsymbol{A}^T$ が P 行列であることに矛盾する.□

ここまでの定理から prdd 行列に関する P 行列サンドイッチは以下のようになっていることがわかる.

隠れ prd-1 行列 $\subseteq$P 行列 $\subseteq$ 隠れ prdd-∞ 行列

また Z^o- 行列について以下のような性質が成り立つ.

性質 4.1 [2] $\boldsymbol{C}$ が Z^o- 行列ならば $\boldsymbol{C}$ は prdd-1 行列である.

(証明) $\boldsymbol{C}$ は Z^o- 行列より,$C_{ij} < 0\ (\forall i,j(i \neq j))$,$\sum_{j=1}^{n} C_{ij} > 0$ を満たしている.

$C_{ij} < 0\ (\forall i,j(i \neq j))$ より,$|C_{ij}| = -C_{ij}$ となる. したがって

$$\begin{aligned} C_{ii} - \sum_{j\neq i} |C_{ij}| &= C_{ii} - \sum_{j\neq i} (-C_{ij}) \\ &= C_{ii} + \sum_{j\neq i} C_{ij} \\ &= \sum_{j=1}^{n} C_{ij} > 0 \end{aligned}$$

となる prdd-1 行列の条件をみたすので,$\boldsymbol{C}$ は prdd-1 行列であることがわかる.□

このことと各隠れ行列の定義や参考文献 [2] と [3] から Z^o 行列のサンドイッチは以下のようになる. このサンドイッチの利点としては P 行列の部分クラスである隠れ Z^o- 行列が入力された LCP は多項式時間で解が求まることがわかっているということがあげられる.

隠れ Z^o- 行列 $\subseteq$P 行列 $\subseteq$ 隠れ Z^o+ 行列

更に prdd 行列と Z^o 行列のサンドイッチを合わせると以下の様な包含関係が成り立っている.

隠れ Z^o- 行列 $\subseteq$ 隠れ prdd-1 行列 $\subseteq$P 行列 $\subseteq$ 隠れ Z^o+ 行列 $\subseteq$ 隠れ prdd∞ 行列

4.4 隠れ行列のパラメータを用いた一般化

今まで紹介したような行列はP行列に近い行列クラスとして判別が容易なのは紹介したとおりだが, より近い行列を発見するのは難しい. そこで各基本行列を単純な実数の変化で移動できるようなパラメータを用いた一般が行われている. まず先行研究として行われているprdd行列のパラメータ化を紹介する.

定義 4.5 [3](k-prdd-in 行列) 実数パラメータ $(1 \leq k \leq n-1)$, 行列 $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ が以下の条件をみたすとき,k-prdd-in 行列であるという.

$$
\begin{aligned}
&C_{ii} > d|C_{ij'}| + \sum_{j\in G} C_{ij}| \\
&\forall(i,j',G)\begin{cases} G \subseteq 1,2,\ldots,n\backslash\{i\} \\ |G| = k' \\ j' \notin G \end{cases} \\
&\text{ただし,} k' = [k]\,, d = k - k'
\end{aligned}
$$

このような k-prdd-in 行列について以下の様な性質が成り立っている.

性質 4.2 [3]

(1) $k = 1$ の時:$k' = 1, d = 0, |G| = 1$ より $C_{ii} > |C_{ij}|, \forall i, j(i \neq j)$ となり 1-prdd-in 行列=prdd-∞ 行列となる

(2) $k = n-1$ の時:$k' = n-1, d = 0, |G| = n-1$ より,$C_{ii} > \sum_{j\neq i} |C_{ij}|$ となり $n-1$-prdd-in 行列=prdd-1 行列となる.

定義 4.6 [3](k-prdd-out 行列) 実数パラメータ $(1 \leq k \leq n-1)$, 行列 $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ が以下の条件をみたすとき,k-prdd-out 行列であるという.

$$
\begin{cases}
C_{ii} > |C_{ij}| & \forall i, j(i \neq j) \in \{1,2,\ldots,n\} \\
C_{ii} > \frac{1}{k}\sum_{j\neq i} |C_{ij}| & \forall i \in \{1,2,\ldots,n\}
\end{cases}
$$

性質 4.3 [3]

(1) $k = 1$ の時:$C_{ii} > \sum_{j\neq i} |C_{ij}|$ となり 1-prdd-out 行列=prdd-1 行列となる

(2) $k = n-1$ の時:$C_{ii} > |C_{ij}|$ となり $n-1$-prdd-out 行列=prdd-∞ 行列となる.

これら行列についても端線表現が可能であるが [3] 参照のこと.

このパタメータ行列についても基本行列と同様に隠れ行列が存在し, 二者択一の定理が存在している. 但し以降 $K_{\text{in}}, K_{\text{out}}$ はそれぞれ k-prdd-in 行列,k-prdd-out 行列の集合とする.

定義 4.7 [3](隠れ行列)
行列 $\boldsymbol{A}$ が隠れ k-prdd-in(k-prdd-out) 行列であるとは,$\boldsymbol{AC} = \boldsymbol{B}$ となる $\boldsymbol{C}, \boldsymbol{B} \in K_{\mathrm{in}}(K_{\mathrm{out}})$, $\boldsymbol{C}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ が存在することである.

定理 4.7 [3](隠れ k-prdd-in 行列の二者択一の定理)
$\boldsymbol{A} \in \mathbb{R}^{n\times n}$ に対して次のいずれか一方かつ一方のみが成り立つ.

(1) $\boldsymbol{AC} = \boldsymbol{B}$ となる $\boldsymbol{C}, \boldsymbol{B} \in K_{\mathrm{in}}$,$\boldsymbol{C}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ が存在する

(2) $\boldsymbol{R} + \boldsymbol{A}^T\boldsymbol{S} = \boldsymbol{O}$ かつ $R_{ii} \geq |R_{ij}|(\forall i \neq j), R_{ij} \geq \frac{1}{k}\sum_{j\neq i}|R_{ij}|(\forall i \in \{1,2,\ldots,n\}), S_{ii} \geq |S_{ij}|(\forall i \neq j), S_{ij} \geq \frac{1}{k}\sum_{j\neq i}|S_{ij}|(\forall i \in \{1,2,\ldots,n\})$ となる $(\boldsymbol{R}, \boldsymbol{S})(\neq (\boldsymbol{O}, \boldsymbol{O}))$,$\boldsymbol{R}, \boldsymbol{S} \in \mathbb{R}^{n\times n}$ が存在する

定理 4.8 [3](隠れ k-prdd-out 行列の二者択一の定理)
$\boldsymbol{A} \in \mathbb{R}^{n\times n}$ に対して次のいずれか一方かつ一方のみが成り立つ.

(1) $\boldsymbol{AC} = \boldsymbol{B}$ となる $\boldsymbol{C}, \boldsymbol{B} \in K_{\mathrm{out}}$,$\boldsymbol{C}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ が存在する

(2) $\boldsymbol{R} + \boldsymbol{A}^T\boldsymbol{S} = \boldsymbol{O}$ かつ $R_{ii} \geq d|R_{ij'}| + \sum_{j\in G}|R_{ij}|, S_{ii} \geq d|S_{ij'}| + \sum_{j\in G}|S_{ij}|$, $\forall(i, j', G)$, ただし $G \subseteq 1,2,\ldots,n\backslash\{i\}, |G| = k'$, $j' \notin G$ となる $(\boldsymbol{R}, \boldsymbol{S})(\neq (\boldsymbol{O}, \boldsymbol{O}))$,$\boldsymbol{R}, \boldsymbol{S} \in \mathbb{R}^{n\times n}$ が存在する

これらの二者択一の定理により P 行列を判別するよりは prdd のパラメータ化行列を判別するほうが効率がいいことがわかる. 一方隠れ prdd-1 行列や隠れ prdd-∞ 行列の二者択一の定理とは違い, 解くべき線形計画問題が非常に大きくなるため容易に求まるとは限らないという注意点がある.

4.4.1 パラメータ Z^o 行列

本研究では先行研究を受けて, もう一方の P 行列サンドイッチである Z^o 行列サンドイッチのパラメータ化に取り組んだ.

パラメータ化するにあたってまずはじめに以下のような各基本行列の生成行列表現に着目した.

$\boldsymbol{E}_{ij}$ を (i,j) 成分が 1 でそれ以外が 0 の $n \times n$ 行列とする.

命題 4.2 (prdd-1 行列の生成行列表現)
$\boldsymbol{C} \in K_{\mathrm{prdd}-1}$ が成り立つための必要十分条件は, 任意の $i, j(i, j \in N, i \neq j)$ に対して,$\alpha_{ij}, \beta ij > 0$ が存在し,$\boldsymbol{C} = \sum_{j\neq i}\{\alpha_{ij}(\boldsymbol{E}_{ii} - \boldsymbol{E}_{ij}) + \beta_{ij}(\boldsymbol{E}_{ii} + \boldsymbol{E}_{ij})\}$ と表せることである.

命題 4.3 (Z^o+ 行列の生成行列表現)
$\boldsymbol{C} \in K_{Z^o+}$ が成り立つための必要十分条件は, 任意の $i, j(i, j \in N, i \neq j)$ に対して,$\alpha_{ij}, \beta ij > 0$ が存在し,$\boldsymbol{C} = \sum_{j\neq i}\{\alpha_{ij}(\boldsymbol{E}_{ii} - \boldsymbol{E}_{ij}) + \beta_{ij}(-\boldsymbol{E}_{ij})\}$ と表せることである.

命題 4.4 (Z^o- 行列の生成行列表現)
$\boldsymbol{C} \in K_{Z^o-}$ が成り立つための必要十分条件は, 任意の $i, j (i, j \in N, i \neq j)$ に対して,$\alpha_{ij}, \beta i > 0$ が存在し,$\boldsymbol{C} = \sum_{j \neq i} \{\alpha_{ij}(\boldsymbol{E}_{ii} - \boldsymbol{E}_{ij}) + \beta_i \boldsymbol{E}_{ii}\}$ と表せることである.

これがどのような意味か代表例として Z^o+ 行列の例を以下に示す.

例 4.4 $\boldsymbol{C} = \begin{bmatrix} 5 & -3 & -4 \\ -1 & 1 & -2 \\ -4 & -6 & 8 \end{bmatrix}$ の時,$i = 1$ の場合を例に考える.$i = 1$ なので $\boldsymbol{E}_{ii}, \boldsymbol{E}_{ij}$ は次のようになる.

$$\boldsymbol{E}_{11} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{E}_{12} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{E}_{13} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

これらを Z^o+ の生成行列表現に当てはめると,

$$\boldsymbol{C} = \begin{bmatrix} \alpha_{12} + \alpha_{13} & -\alpha_{12} - \beta_{12} & -\alpha_{13} - \beta_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

となるような $\alpha, \beta > 0$ が存在するかどうか確かめる. 今回の場合は $\alpha_{12} = 2, \alpha_{13} = 3, \beta_{12} = 1, \beta_{13} = 1$ で $\boldsymbol{C}$ の 1 行目となるので α, β がともに存在している.$i = 2, 3$ の時も同様に考えれば良い.

この生成行列表現で注目すべき部分は各行列が正の結合で表されているということである. 更に係数として α を持つ部分は各行列で共通しており, 異なる部分は β を係数として持つ部分であるということに着目する.

上記の部分に着目し,P 行列サンドイッチの包含関係の両端に当たる Z^o+ と Z^o- の異なる部分の凸結合を考えることで以下のような $Z^o(t)$ 行列を定義した.

定義 4.8 [fujiwara]($Z^o(t)$ 行列の定義) 実数パラメータ t, 行列 $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ が以下の条件をみたすとき $Z^o(t)$ 行列であるという.

$$\boldsymbol{C} = \sum_{j \neq i} \alpha_{ij}(\boldsymbol{E}_{ii} - \boldsymbol{E}_{ij}) + \sum_{j \neq i} \beta_{ij} \boldsymbol{E}(t)_{ij}$$

但し

$$\begin{aligned} \boldsymbol{E}(t)_{ij} &:= (1-t)(\boldsymbol{E}_{ii} + \boldsymbol{E}_{ij}) + t(-\boldsymbol{E}_{ij}) \\ &= (1-t)\boldsymbol{E}_{ii} + (1-2t)\boldsymbol{E}_{ij} \end{aligned}$$

である.

この $Z^o(t)$ 行列は以下の様な特徴を持っている.

特徴

- $t=0$ の時,prdd-1 の生成行列表現と同様. すなわち $Z^o(0)$ =prdd-1 行列クラス
- $t=\frac{1}{2}$ の時,Z^o- の生成行列表現と同様. すなわち $Z^o(0)=Z^o-$ 行列クラス
- $t=1$ の時,Z^o+ の生成行列表現と同様. すなわち $Z^o(0)=Z^o+$ 行列クラス
- t が $0\leq t\leq t'<\frac{2}{3}$ の時, 包含関係は $Z^o(t')$ クラス $\subseteq Z^o(t)$ クラス
- t が $\frac{2}{3}<t\leq t'\leq 1$ の時, 包含関係は $Z^o(t)$ クラス $\subseteq Z^o(t')$ クラス

上 3 つの特徴から下の特徴は明らかである.

ここからわかることは明らかに $Z^o(t)$ 行列は拡張になっているということと,$\frac{2}{3}$ を堺に包含関係が逆転するということである.

この行列についても以下のような隠れ行列が存在し, 二者択一の定理が成り立つ.

定義 4.9 (隠れ行列)
行列 $\boldsymbol{A}$ が隠れ $Z^o(t)$ 行列であるとは,$\boldsymbol{AC}=\boldsymbol{B}$ となる $Z^o(t)$ 行列 $\boldsymbol{C},\boldsymbol{B}\in\mathbb{R}^{n\times n}$ が存在することである.

定理 4.9 (隠れ $Z^o(t)$ 行列の二者択一の定理)
$\boldsymbol{A}\in\mathbb{R}^{n\times n}$ に対して次のいずれか一方かつ一方のみが成り立つ.

(1) $\boldsymbol{AC}=\boldsymbol{B}$ となる $Z^o(t)$ 行列 $\boldsymbol{C},\boldsymbol{B}\in\mathbb{R}^{n\times n}$ が存在する

(2) $\boldsymbol{R}+\boldsymbol{A}^T\boldsymbol{S}=\boldsymbol{O}$ かつ $\langle\boldsymbol{E_{ii}}-\boldsymbol{E_{ij}},\boldsymbol{R}\rangle\geq 0$,$\langle\boldsymbol{E}(t)_{ij},\boldsymbol{R}\rangle\geq 0$,$\langle\boldsymbol{E_{ii}}-\boldsymbol{E}_{ij},\boldsymbol{S}\rangle\geq 0$,$\langle\boldsymbol{E}(t)_{ij},\boldsymbol{S}\rangle\geq 0$.
となる $(\boldsymbol{R},\boldsymbol{S})(\neq(\boldsymbol{O},\boldsymbol{O}))$,$\boldsymbol{R},\boldsymbol{S}\in\mathbb{R}^{n\times n}$ が存在する.
但し $\langle\boldsymbol{C},\boldsymbol{R}\rangle$ は $\boldsymbol{C}$ と $\boldsymbol{R}$ のスカラー積を表す, すなわち,$\langle\boldsymbol{C},\boldsymbol{R}\rangle=\sum_{i,j}C_{ij}R_{ij}$.

このことから $Z^o(t)$ 行列の判別も多項式時間であることがわかる.

4.5 パラメーター化行列と P 行列の包含関係

最後にいくつかの包含関係を紹介する.

例 4.5 [3] 以下の行列 $\overline{\boldsymbol{A}}(\boldsymbol{t})\in\mathbb{R}^{n\times n}$ $(n\geq 4)$ は,P 行列であるが, 隠れ k-prdd-in 行列 $(\forall k(1<k\leq n-1))$ でない行列である.

$$\overline{\boldsymbol{A}}(\boldsymbol{t})=\begin{bmatrix}\boldsymbol{A}(\boldsymbol{t}) & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}\quad \text{ただし, } \boldsymbol{A}(\boldsymbol{t})=\frac{1}{2+t^2}\begin{bmatrix}t^2 & 0 & 2 & 2t\\ 0 & t^2 & -2t & 2\\ -2 & 2t & t^2 & 0\\ -2t & -2 & 0 & t^2\end{bmatrix}$$

この行列 $\boldsymbol{A}$ は $t>0$ の範囲で,P 行列である.

(証明) [3] を参照

例 4.6 [3]**3.2** 以下の行列 $\boldsymbol{A} \in \quad \mathbb{R}^{n\times n}$ $(n > 3)$ は，隠れ k-prdd-in 行列 $(\forall k(1 \leq k < n-1))$ であるが,P 行列でない行列の例である.

$$\boldsymbol{A} = \begin{bmatrix} n-1 & -1 & \cdots & -1 \\ -1 & n-1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ -1 & \cdots & -1 & n-1 \end{bmatrix}$$

この行列 $\boldsymbol{A}$ は $1 \leq k < n-1$ の範囲で隠れ k-prdd-in 行列である.

(証明) [3] を参照

この例からわかることは隠れ k-prdd-in 行列はタイトな部分クラスになっており, 広げることが出来ないということである. 次に先行研究でわかっている包含関係を紹介する.

定理 4.10 [3]P 行列 $\subseteq$ 隠れ $(n-2)$-prdd-out である.

(証明) [3] を参照

これらを受けて今現在わかっている包含関係は以下である.

$$\text{隠れ } Z^o(\tfrac{1}{2}) \subseteq \text{隠れ } Z^o(0) \subseteq \text{P 行列} \subseteq \text{隠れ } Z^o(1) \subseteq \text{隠れ (n-2)-prdd-out}$$

これで今現在わかっている包含関係を紹介したが $Z^o(t)$ 行列に関する包含関係をどこまで狭くできるかは, 未だにわかっていない. 次の章ではその包含関係を見出すために計算機実験を行っている.

5　計算機実験

前章でP行列サンドイッチに用いる行列をパラメータ化したことで簡単な実数操作で互いの行列に行き来でき,二者択一の定理でそれらの判別も容易にできることを説明した.この章ではそれらパラメータ化行列のパラメーターとして用いた実数を減らす,もしくは増やすことでどこまでP行列に迫れるか計算機実験を行った.またLCPとの関係を見るためにP行列に含まれる行列について,LCPのPPMを何回で終わることができるかの計算機実験も行っている.

5.1　計算機実験によるP行列へのアプローチ

まずはじめにP行列の代表例である例4.5について隠れ$Z^o(t)$行列のtをどこまで狭められるかの実験を行ったところ結果は以下のようになった.

[実験方法]

(1) 例4.5の行列を$\boldsymbol{A}$として隠れ$Z^o(t)$行列の判別にかける

(2) tを二部探索でどこまで狭められるか確かめる

[結果]

$s \diagdown n$	4	5	6	7
0.1	0.99975	0.99975	0.99975	0.99975
0.2	0.99763	0.99763	0.99763	0.99763
0.3	0.99112	0.99112	0.99112	0.99112
0.4	0.97753	0.97753	0.97753	0.97753
0.5	0.95458	0.95458	0.95458	0.95458
0.6	0.92154	0.92154	0.92154	0.92154
0.7	0.88037	0.88037	0.88037	0.88037
0.8	0.83512	0.83512	0.83512	0.83512

[予想]

この結果からわかることは$s = 0.1$の時tの値がほぼ1になってしまっているので$\frac{2}{3} < t \leq 1$の範囲では隠れ$Z^o(t)$行列は1から少しでも小さくするとはみ出すP行列があるので,P行列に近づくことは出来ないということである.

次に$t < 0$の範囲でどこまでtを下げることができるかという実験とともに,[3]で行った実験を拡張したような以下の実験を行った.

[実験方法]

(1) $A = BC^{-1}$となる行列Aをランダムに生成する.ただしBとCはk-prdd-out行列であり,$k \in [1, 6]$として0.5刻みで増加させる

(2) A が P 行列かどうか調べる (全ての主座小行列式を計算する)

(3) もし A が P 行列なら, A が隠れ k-prdd-out 行列となる最小の k,A が $t' \leq 0$ で $Z^o(t')$ 行列となる最小の t' を調べる. (二分探索).

先行研究での実験との違いは,$k \in [1,6]$ で $\boldsymbol{C}, \boldsymbol{B}$ を作っていることである. 理由はそれを超えても P 行列となる行列が作られる可能性があるからである. 以下が結果.

各 n に対して,B を k-prdd-out,C を k'-prod-out で $A = BC^{-1}$ を十分な数生成する.

[結果]

n	% of P-matrix	max. k.	min. t
4	13.23	1.32	-1.47
5	7.62	1.35	-1.65
6	5.95	1.26	-1.90
7	5.90	1.27	-1.94
8	6.58	1.33	-1.92
10	8.42	1.33	-1.96
12	10.37	1.42	-1.999
14	12.20	1.43	-1.996

[予想]

この結果から新しく予想できることは t は負の範囲ではどんどん拡大していってるので $t < 0$ の範囲では隠れ $Z^o(t)$ 行列は全て P 行列に含まれており,P 行列に近づくことは出来ないのではないかという予想と, 先行研究と同様に隠れ k-prdd-out 行列は $k = 2$ あるいは $k = \sqrt{3}$ くらいまで下げられるのではないかということである.k-prdd-out の予想に関しては [3] も参考のこと.

5.2 LCP の PPM を用いた実験

次に LCP にのピボットアルゴリズムである PPM に P 行列の各部分クラスに所属する行列を入力して最大何回のループで終わるかの実験をした. なぜこのような実験をしたかというと隠れ Z^o- 行列が入力されるような LCP は多項式時間で計算可能であることがわかっていること, なぜ PPM で実験したかについては先行研究 [4] で LCP と PPM について以下のような関係がわかっているからである.K 行列については [4] 参照.

命題 5.1 [4]P 行列の部分クラスである K 行列について,K 行列が入力された LCP の PPM は最大 $2n$ 回のループで終了する

(証明) [4] 参照

このような命題が存在するため P 行列の部分クラスである隠れ Z^o- と隠れ Z^o- を含まないような隠れ prdd-1 行列について実験を行った.

[実験方法:隠れ Z^o- 行列]

(1) $A = BC^{-1}$ となる行列 A をランダムに生成する. ただし B と C は Z^o- 行列である.

(2) $A \in \mathbb{R}^{n\times n}$ とすべての要素が-10000 であるようなベクトル $q \in \mathbb{R}^n$ を PPM に入力する. 返り値はループ回数, 基底解とし, ループ回数が n^2 以上になったら失敗とする.

(3) $4 \leq n \leq 500$ で次々に作られる新たな A を対角に並べて行列の次元を順次増やす.$n = 20$ までは 4 刻みそれ以降は 20 刻みとする.

なぜ (3) のような処理をするかというと 1 つずつ次元を増やしていくような行列の作り方をすると実験可能な時間で実験が終わらなくなってしまうためである. さらに多量の $\boldsymbol{B}, \boldsymbol{C}$ を生成しても n が増えるに連れて最大ループ数の $\boldsymbol{A}$ を見つけにくくなってしまうためである.

[結果]

n	4	8	12	16	20	24	40	60	80	100	120	140	160
ループ数	10	19	28	37	46	55	91	136	181	226	271	316	361
n	180	200	220	240	260	280	300	320	340	360	380	400	420
ループ数	406	451	496	541	586	631	676	721	766	811	856	901	946
n	440	460	480	500									
ループ数	991	1036	1081	1126									

[実験方法:隠れ prdd-1 行列]

(1) $A = BC^{-1}$ となる行列 A をランダムに生成する. ただし B と C は (1.7)-prdd-out 行列である.

(2) $\boldsymbol{A}$ が隠れ prdd-1 行列かつ隠れ Z^o- 行列でないことを調べる.

(3) $A \in \mathbb{R}^{n\times n}$ とすべての要素が-10000 であるようなベクトル $q \in \mathbb{R}^n$ を PPM に入力する. 返り値はループ回数, 基底解とし, ループ回数が n^2 以上になったら失敗とする.

(4) $4 \leq n \leq 500$ で次々に作られる新たな A を対角に並べて行列の次元を順次増やす.$n = 20$ までは 4 刻みそれ以降は 20 刻みとする.

なぜ (1.7)-prdd-out で作っているかという説明をしておくと直接 $\boldsymbol{B}, \boldsymbol{C}$ を prdd-1 行列として作ると (2) の判別で Z^o- 行列でない行列が少なくなってしまい, 実験が終わらなくなるためである. そのために k-prdd-out 行列を使って幅広く行列を作成している.

[結果]

n	4	8	12	16	20	40	60	80	100	120	140	160
ループ数	16	31	46	61	76	151	226	301	376	451	526	601
n	180	200	220	240	260	280	300	320	340	360	380	400
ループ数	676	751	826	901	976	1051	1126	1201	1276	1351	1426	1501
n	420	440	460	480	500							
ループ数	1576	1651	1726	1801	1876							

[予想]

まず一つ目の予想として隠れ Z^o- は PPM でも多項式時間で解けるのではないかと予想ができる. 根拠としては実験結果が余り膨れ上がらないのとともに, もともと隠れ Z^o- には効率的なアルゴリズムが存在するからである.

また前にも述べたように PPM の計算量は線形方程式を解く程度で収まっており,$O(n^3 \sim n^4)$ 程度である. このことと, 隠れ prdd-1 行列のループ数はより増加することが予想できることより, 隠れ prdd-1 行列に対して PPM は有用なアルゴリズムではないという予想をした. それに伴って隠れ prdd-1 行列を含まないような P 行列でも PPM は有用なアルゴリズムではないと考えている.

6　まとめ, 考察

本研究ではP行列判別のため, よりP行列に近い行列クラスを見つけることが目的であり先行研究であるprdd行列の拡張を受けて,Z^o行列を拡張した$Z^o(t)$行列を定義, 導入した. どのように導入したかというとZ^o+行列とZ^o-行列の生成行列表現と呼ばれるものに着目し, それの凸結合を取る形で定義することが出来た. またこの$Z^o(t)$行列についてもtに$0, \frac{1}{2}, 1$を入れた時のどのような行列になるかや,$\frac{2}{3}$より大きいか小さいかで包含関係が逆転することがわかっている..

この他に計算機実験によってパラメータであるtは$\frac{2}{3} < t \leq 1$の範囲ではP行列に近づくことが出来ず,$t \leq 0$の範囲でもP行列に全て含まれるので近づくことは出来ないという考察をした. さらにP行列サンドイッチにおけるLCPとの関係について,LCPのピボットアルゴリズムであるPPMにおいては隠れprdd-1行列およびP行列には適さないという予想をした.

今後の課題としては$Z^o(t)$行列を用いてP行列に近づくためのより理論的な部分の考察が必要である. 特に$Z^o(t)$行列のパラメータ付けの方法ではprdd-∞行列が含まれておらず，より良いパラメータ化が存在する可能性がある. また隠れZ^o-行列ではない隠れprdd-1行列や, 隠れZ^o-行列でもなく隠れprdd-1行列でもないP行列に関するLCPの解法アルゴリズムについての考察が必要であると考える.

謝辞

本論文を作成するにあたって, 定理の証明や数々の先行研究の紹介にとどまらず, 計算機実験に於けるプログラミング作成や実験方法などについて丁寧なご指導ご鞭撻を頂きました東邦大学　理学研究科　情報科学専攻　並木研究室　並木誠准教授, また, 研究の節目でご多忙の中至らないところを指導, 提案していだいた同学副査の白柳潔教授, 高田英行講師に深く感謝の気持と御礼を申し上げます.

またともに研究に励むとともに先輩として指導していただいた修了生の塚崎茂章先輩, 吉宮博人先輩に感謝の気持と御礼を申し上げます.

参考文献

[1] 並木誠,『応用最適化シリーズ 1:線形計画法』, 朝倉書店,2008 年

[2] Walter Morris and Makoto Namiki,『Good Hidden P-matrix Sandwiches』,2007

[3] 吉宮博人,『隠れ行列を用いた P 行列判別と計算機実験』, 東邦大学 大学院 理学研究科 情報科学専攻 並木研究室,2014 年

[4] WJon Foniok,Komei Fukuda,Bernd Gartner,Hans-Jakob Kuthi,『Pivoting in Linear Complementarity:Two Polynomial-Time Cases』,2009

付録

以下に付録するソースコードは実験で使ってたものである. 使用言語は Python, 仕様モジュールは各付録のインポート文を参照.

(1:prdd_and_znat.py)P 行列やそれぞれ隠れではない基本行列, パラメータ行列の判別のためのソース. 主な返り値は boolean.

```
#! usr/bin/env python
# coding:utf-8

from numpy import *
from math import *
import itertools
MEPS=1.0e-8

#添字のすべての部分集合を作成
#形式はリスト
def all_nonempty_subsets(ls):
    return [list(x) for i in range(1,len(ls)+1)
           for x in itertools.combinations(ls,i)]

def all_subsets(ls):
    return [list(x) for i in range(0,len(ls)+1)
           for x in itertools.combinations(ls,i)]

#P 行列判別
#定義通りなので教科書参照
def pmatrixQ(matrix):
    indexsets = all_nonempty_subsets(range(len(matrix)))
    for i in range(len(indexsets)):
        if linalg.det(matrix[ix_(indexsets[i],indexsets[i])])<=1.0e-10:
            return False
    return True

#prdd-1 判別
#こちらも定義通り
def prdd_1Q(matrix):
    for i in range(len(matrix)):
        offdiag=0
        for j in range(len(matrix)):
            if i!=j:
                offdiag+=abs(matrix[i][j])
        if matrix[i][i]<=offdiag+MEPS:
            return False
    return True

#prdd-inf 判別
#これも定義通り
def prdd_infQ(matrix):
    for i in range(len(matrix)):
```

```
        for j in range(len(matrix)):
            if matrix[i][i]<=abs(matrix[i][j]) and i!=j:
                    return False
    return True

#Znat 作成
#定義通りだが plus とかより短い
def znat(matrix):
    n=len(matrix)
    for i in range(n):
        if matrix[i,i]<=0:
            return False
        J=range(n)
        J.remove(i)
        for j in J:
            if matrix[i,j]>=0:
                return False
    return True

#Z^o plus 行列
#定義通り
def zplusQ(matrix):
    n=len(matrix)
    for i in range(n):
        offdiag=0
        if matrix[i][i]<=0:
            return False
        for j in range(n):
            offdiag+=matrix[i][j]
            if matrix[i][j]>=0 and i!=j:
                    return False
        if offdiag>=0:
            return False
    return True

#Z^o minus 行列
#定義通り
def zminusQ(matrix):
    n=len(matrix)
    for i in range(n):
        offdiag=0
        if matrix[i][i]<=0:
            return False
        for j in range(n):
            offdiag+=matrix[i][j]
            if matrix[i][j]>=0 and i!=j:
                    return False
        if offdiag<=0:
            return False
```

```
    return True

#k-prdd-out 判別
#定義通り
def k_prdd_out(matrix,k):
    n=len(matrix)
    for i in range(n):
        offdiag=0
        for j in range(n):
             if i!=j:
                offdiag+=abs(matrix[i][j])
        if matrix[i][i]<=(offdiag/k)+MEPS:
            return False
    return True

#k-prdd-in 判別
#定義通り
def k_prdd_in(matrix,k):
    n=len(matrix)
    N=range(n)
    kk=int(floor(k))
    d=k-kk
    for i in N:
        J=range(n)
        J.remove(i)
        GG=list(itertools.combinations(J,kk))
        for g in GG:
            G=list(g)
            offdiag=0
            for j in G:
                offdiag+=abs(matrix[i,j])
            H=list(set(J)-set(G))
            if H != []:
                for jj in H:
                    offdiag+=d*abs(matrix[i,jj])
                if matrix[i,i]<=offdiag+MEPS:
                    return False
            else:
                if matrix[i,i] <= offdiag+MEPS:
                    return False
    return True
```

(2:hidden_matrix2.py) 主に隠れ行列を線形計画問題を解くことで判別するプログラム. 仕様モジュールは Gurobi Optimaizer. これは数理計画用ソルバーであり, シャアウェアである.

```
#! usr/bin/env python
# coding:utf-8

#今回修正分 5/28
#Znat と S を作成,S かつ Znat も一応作成
#Znatplus の R,S 条件の非ゼロを調整

from numpy import *
from prdd_and_znat import *
from gurobipy import *
MEPS=1.0e-8

def s_matrix(M,mfrag=True):
    model=Model("s_matrix")
    N=range(len(M))
    #変数宣言
    x={}
    for i in N:
        x[i]=model.addVar(vtype="C",lb=1.0,name="x(%s)" % (i))
    model.update()
    #制約条件入力
    for i in N:
        model.addConstr(quicksum(M[i,j]*x[j] for j in N)>=1,name="Smatrix-Const(%s)"%(i))
    model.update()
    #目的関数入力
    model.setObjective(quicksum(x[i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #グロビオブジェクトからリストへ
    #フラグが False ならば計算不要
    if model.status == GRB.Status.OPTIMAL:
        if mfrag==False:
            return True,[]
        else:
            xx=[]
            for i in N:
                rowx=[]
                rowx.append(x[i].X)
                xx.append(rowx)
            return True,array(xx)
#-------------------------------------------------
    ##R,S
    #フラグが False の場合は表示しないので計算せずに False と空のリストを返せば良い
    if mfrag==False:
        return False,[]

    #変数宣言
```

```
    model=Model("s_matrix")
    MT=transpose(M)
    y={}
    for i in N:
        y[i]=model.addVar(vtype="C",lb=1.0,name="y(%s)" % (i))
    model.update()

    #制約条件
    for i in N:
        model.addConstr(quicksum(MT[i,j]*y[j] for j in N)<=0,name="Smatrix-Const(%s)"%(i))
    model.update()

    #目的関数入力
    model.setObjective(quicksum(y[i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #表示
    if model.status == GRB.Status.OPTIMAL:
        yy=[]
        for i in N:
            rowy=[]
            rowy.append(y[i].X)
            yy.append(rowy)
        return False,array(yy)

def hidden_znat(A,mfrag=True):
    model=Model("hiddenZnat")
    N=range(len(A))

    #変数宣言
    C={}
    B={}
    for i in N:
        for j in N:
            if i==j:
                C[i,j]=model.addVar(vtype="C",lb=1.0,name="C(%s,%s)" % (i,j))
                B[i,j]=model.addVar(vtype="C",lb=1.0,name="B(%s,%s)" % (i,j))
            else:
                C[i,j]=model.addVar(vtype="C",ub=-1.0,lb=-GRB.INFINITY,name="C(%s,%s)"
                    % (i,j))
                B[i,j]=model.addVar(vtype="C",ub=-1.0,lb=-GRB.INFINITY,name="B(%s,%s)"
                    % (i,j))
    model.update()

    #制約条件入力
    for i in N:
        for j in N:
            model.addConstr(quicksum(A[i,k]*C[k,j] for k in N)==B[i,j],
                name="Zplus-Const(%s,%s)"%(i,j))
```

```
    model.update()

    #目的関数入力
    model.setObjective(quicksum(C[i,i] for i in N)+quicksum(B[i,i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #グロビオブジェクトからリストへ
    #フラグがFalseならば計算不要
    if model.status == GRB.Status.OPTIMAL:
        if mfrag==False:
            return True,True,[],[]
        else:
            CC=[]
            BB=[]
            for i in N:
                rowc=[]
                rowb=[]
                for j in N:
                    rowc.append(C[i,j].X)
                    rowb.append(B[i,j].X)
                    CC.append(rowc)
                    BB.append(rowb)
                return znat(array(CC)),znat(array(BB)),array(BB),array(CC)
#-------------------------------------------
    #フラグがFalseのときはR,Sのを求めない
    if mfrag==False:
        return False,False,[],[]

    model=Model("hiddenZnat")
    model=Model("hiddenZnat")
    AT=transpose(A)

    #変数宣言
    R={}
    S={}
    for i in N:
        for j in N:
            R[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="R(%s,%s)" % (i,j))
            S[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="S(%s,%s)" % (i,j))
    model.update()

    #制約条件入力
    for i in N:
        J=range(len(N))
        J.remove(i)
        for j in N:
            model.addConstr(R[i,j]+quicksum(AT[i,k]*S[k,j] for k in N)==0,
                name="Prdd-Const(%s,%s)"%(i,j))
        for j in J:
```

```
            model.addConstr(R[i,i] >= R[i,j],name= "R-offDiag-Const(%s,%s)"%(i,j))
            model.addConstr(S[i,i] >= S[i,j],name= "S-offDiag-Const(%s,%s)"%(i,j))
    model.addConstr(quicksum(S[i,i] for i in N) >= len(A),name= "S-Diag-Const(%s,%s)"%(i,j))

    #目的関数入力
    model.setObjective(quicksum(S[i,i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #表示
    if model.status == GRB.Status.OPTIMAL:
        RR=[]
        SS=[]
        for i in N:
            rowr=[]
            rows=[]
            for j in N:
                rowr.append(R[i,j].X)
                rows.append(S[i,j].X)
            RR.append(rowr)
            SS.append(rows)
        return znat(array(RR)),znat(array(SS)),array(RR),array(SS)

def parameter_zmat(C,t):
    """
    returns True or False
    """
    #Z^o(t) 行列の判別は不等式ではない為このような形である
    model=Model("Parametric_Zmatrix")

    N=range(len(C))
    a={}
    b={}
    for i in N:
        for j in N:
            a[i,j]=model.addVar(vtype="C",ub=GRB.INFINITY,lb=1,name="a(%s,%s)" % (i,j))
            b[i,j]=model.addVar(vtype="C",ub=GRB.INFINITY,lb=1,name="b(%s,%s)" % (i,j))
    model.update()

    for i in N:
        J=range(len(N))
        J.remove(i)
        model.addConstr(quicksum(a[i,j] for j in J)+(1-t)*(quicksum(b[i,j] for j in
            J))==C[i,i],name="PZ-Diag-Const(%s,%s)"%(i,i))
    for i in N:
        J=range(len(N))
        J.remove(i)
        for j in J:
                model.addConstr(-a[i,j]+(1-2*t)*(b[i,j])==C[i,j],name=
                    "PZ-Const(%s,%s)"%(i,j))
```

```
    model.update()


    model.setObjective(quicksum(quicksum(a[i,j]+b[i,j] for j in N) for i in N),GRB.MINIMIZE)
    model.optimize()
    model.write("test2.lp")

    if model.status == GRB.Status.OPTIMAL:
        return True
    else:
        return False

def hidden_prdd1Q(A,mfrag=True):
    """
    retruns (True,C,B) if A is hidden prdd-1 otherwise (False,R,S).
    if mfrag is False, matrices C,B,R,S will be [].
    """
    model=Model("prdd")
    N=range(len(A))

    #変数宣言
    C={}
    B={}
    for i in N:
        for j in N:
            C[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="C(%s,%s)" % (i,j))
            B[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="B(%s,%s)" % (i,j))
    cy={}
    cz={}
    by={}
    bz={}
    for i in N:
        J=range(len(N))
        J.remove(i)
        for j in J:
            cy[i,j]=model.addVar(vtype="C",lb=1,name="cy(%s,%s)" % (i,j))
            cz[i,j]=model.addVar(vtype="C",lb=1,name="cz(%s,%s)" % (i,j))
            by[i,j]=model.addVar(vtype="C",lb=1,name="by(%s,%s)" % (i,j))
            bz[i,j]=model.addVar(vtype="C",lb=1,name="bz(%s,%s)" % (i,j))
    model.update()


    #制約条件入力
    for i in N:
        for j in N:
            model.addConstr(quicksum(A[i,k]*C[k,j] for k in N)==B[i,j],
                name="Prdd-Const(%s,%s)"%(i,j))
    for i in N:
        J=range(len(N))
```

```
        J.remove(i)
        model.addConstr(quicksum(cy[i,j] for j in J)+quicksum(cz[i,j] for j in
             J)==C[i,i],name="C-Const(%s,%s)" %(i,i))
    for i in N:
        J=range(len(N))
        J.remove(i)
        for j in J:
            model.addConstr(C[i,j] == -cy[i,j]+cz[i,j],
                name= "C-OffDiag-Const(%s,%s)"%(i,j))
    for i in N:
        J=range(len(N))
        J.remove(i)
        model.addConstr(quicksum(by[i,j] for j in J)+quicksum(bz[i,j] for j in
             J)==B[i,i],name="B-Diag-Const(%s,%s)" %(i,i))
    for i in N:
        J=range(len(N))
        J.remove(i)
        for j in J:
            model.addConstr(B[i,j] == -by[i,j]+bz[i,j],name= "B-OffDiag-Const(%s,%s)"%(i,j))
    model.update()

    #目的関数入力
    model.setObjective(quicksum(cy[i,j] for (i,j) in cy)
        +quicksum(cz[i,j] for (i,j) in cz)+
                       quicksum(by[i,j] for (i,j) in by)+quicksum(bz[i,j]
                           for (i,j) in bz),GRB.MINIMIZE)
    model.optimize()

    #グロビオブジェクトからリストへ
    #フラグがFalseならば計算不要
    if model.status == GRB.Status.OPTIMAL:
        if mfrag==False:
            return True,True,[],[]
        else:
            CC=[]
            BB=[]
            for i in N:
                rowc=[]
                rowb=[]
                for j in N:
                    rowc.append(C[i,j].X)
                    rowb.append(B[i,j].X)
                    CC.append(rowc)
                    BB.append(rowb)
                    return prdd_1Q(array(CC)),prdd_1Q(array(BB)),array(BB),array(CC)
#-----------------------------------------
    ##R,S
    #フラグがFalseの場合は表示しないので計算せずにFalseと空のリストを返せば良い
    if mfrag==False:
```

```
        return False,False,[],[]

    #変数宣言
    model=Model("hidden_prdd")
    AT=transpose(A)
    R={}
    S={}
    for i in N:
        for j in N:
            R[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="R(%s,%s)" % (i,j))
            S[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="S(%s,%s)" % (i,j))
    model.update()

    #制約条件
    for i in N:
        J=range(len(N))
        J.remove(i)
        for j in N:
            model.addConstr(R[i,j]+quicksum(AT[i,k]*S[k,j] for k in N)==0,
                name="Prdd-Const(%s,%s)"%(i,j))
        for j in J:
            model.addConstr(R[i,i] >= R[i,j],name= "R-OffDiag-Const(%s,%s)"%(i,j))
            model.addConstr(R[i,i] >= -R[i,j],name= "-R-OffDiag-Const(%s,%s)"%(i,j))
            model.addConstr(S[i,i] >= S[i,j],name= "S-OffDiag-Const(%s,%s)"%(i,j))
            model.addConstr(S[i,i] >= -S[i,j],name= "-S-OffDiag-Const(%s,%s)"%(i,j))
    model.addConstr(quicksum(S[i,i] for i in N) >= len(A),name= "S-Diag-Const(%s,%s)"%(i,j))

    #目的関数入力
    model.setObjective(quicksum(S[i,i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #表示
    if model.status == GRB.Status.OPTIMAL:
        RR=[]
        SS=[]
        for i in N:
            rowr=[]
            rows=[]
            for j in N:
                rowr.append(R[i,j].X)
                rows.append(S[i,j].X)
            RR.append(rowr)
            SS.append(rows)
        return prdd_1Q(array(RR)),prdd_1Q(array(SS)),array(RR),array(SS)

def hidden_prddinfQ(A,mfrag=True):
    """
    retruns (True,C,B) if A is hidden prdd-inf otherwise (False,R,S).
    if mfrag is False, matrices C,B,R,S will be [].
```

```
    """
    model=Model("hidden_prdd_inf")
    N=range(len(A))

    #変数宣言
    C={}
    B={}
    for i in N:
        for j in N:
            C[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="C(%s,%s)" % (i,j))
            B[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="B(%s,%s)" % (i,j))
    model.update()

    #制約条件入力
    for i in N:
        for j in N:
            model.addConstr(quicksum(A[i,k]*C[k,j] for k in N)==B[i,j],
                name="Prdd-inf-Const(%s,%s)"%(i,j))
        J=range(len(N))
        J.remove(i)
        for j in J:
            model.addConstr(C[i,i] >= C[i,j]+1,name= "C-OffDiag-Const(%s,%s)"%(i,j))
            model.addConstr(C[i,i] >= -C[i,j]+1,name= "-C-OffDiag-Const(%s,%s)"%(i,j))
            model.addConstr(B[i,i] >= B[i,j]+1,name= "B-OffDiag-Const(%s,%s)"%(i,j))
            model.addConstr(B[i,i] >= -B[i,j]+1,name= "-B-OffDiag-Const(%s,%s)"%(i,j))
    model.update()

    #目的関数入力
    #なんでもいいので対角成分同士を足したものの最小化
    model.setObjective(quicksum(C[i,i] for i in N)+quicksum(B[i,i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #表示
    #フラグがFalseの場合は計算不要
    if model.status == GRB.Status.OPTIMAL:
        if mfrag==False:
            return True,True,[],[]
        CC=[]
        BB=[]
        for i in N:
            rowc=[]
            rowb=[]
            for j in N:
                rowc.append(C[i,j].X)
                rowb.append(B[i,j].X)
            CC.append(rowc)
            BB.append(rowb)
        return prdd_infQ(array(CC)),prdd_infQ(array(BB)),array(BB),array(CC)
#-----------------------------------------------------
```

```
#R,S
#同じようにフラグがFalseならばR,Sの計算不要
if mfrag==False:
    return False,False,[],[]

model=Model("hidden_prddinf")
AT=transpose(A)

#変数宣言
R={}
S = {}
for i in N:
    for j in N:
        R[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="R(%s,%s)" % (i,j))
        S[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="S(%s,%s)" % (i,j))
ry={}
rz={}
sy={}
sz={}
for i in N:
    J=range(len(N))
    J.remove(i)
    for j in J:
        ry[i,j]=model.addVar(vtype="C",lb=0,name="ry(%s,%s)" % (i,j))
        rz[i,j]=model.addVar(vtype="C",lb=0,name="rz(%s,%s)" % (i,j))
        sy[i,j]=model.addVar(vtype="C",lb=0,name="sy(%s,%s)" % (i,j))
        sz[i,j]=model.addVar(vtype="C",lb=0,name="sz(%s,%s)" % (i,j))
model.update()

#制約条件作成
for i in N:
    for j in N:
        model.addConstr(R[i,j]+quicksum(AT[i,k]*S[k,j] for k in N)==0,
            name="Prdd-Const(%s,%s)"%(i,j))
for i in N:
    J=range(len(N))
    J.remove(i)
    model.addConstr(quicksum(ry[i,j] for j in J)
        +quicksum(rz[i,j] for j in J)==R[i,i],name="R-Const(%s,%s)" %(i,i))
for i in N:
    J=range(len(N))
    J.remove(i)
    for j in J:
        model.addConstr(R[i,j] == -ry[i,j]+rz[i,j],name= "R-OffDiag-Const(%s,%s)"%(i,j))
for i in N:
    J=range(len(N))
    J.remove(i)
    model.addConstr(quicksum(sy[i,j] for j in J)
        +quicksum(sz[i,j] for j in J)==S[i,i],name="S-Diag-Const(%s,%s)" %(i,i))
```

```
    for i in N:
        J=range(len(N))
        J.remove(i)
        for j in J:
            model.addConstr(S[i,j] == -sy[i,j]+sz[i,j],name= "S-OffDiag-Const(%s,%s)"%(i,j))
    model.addConstr(quicksum(S[i,i] for i in N) >= len(A),name= "S-Diag-Const(%s,%s)"%(i,j))
    model.update()

    #目的関数作成
    #目的関数はなんでもいいので対角成分の最小化
    model.setObjective(quicksum(S[i,i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #表示
    #グロビ形式からリストに直して返す
    if model.status == GRB.Status.OPTIMAL:
        RR=[]
        SS=[]
        for i in N:
            rowr=[]
            rows=[]
            for j in N:
                rowr.append(R[i,j].X)
                rows.append(S[i,j].X)
            RR.append(rowr)
            SS.append(rows)
        return prdd_1Q(array(RR)),prdd_1Q(array(SS)),array(RR),array(SS)

def hidden_k_prdd_out(A,k,mfrag=True):
    """
    retruns (True,C,B) if A is hidden k-prdd-out otherwise (False,R,S).
    if mfrag is False, matrices C,B,R,S will be [].
    """
    model=Model("hidden_k_prddout")
    N=range(len(A))

    #変数宣言
    C={}
    B={}
    for i in N:
        for j in N:
            C[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="C(%s,%s)" % (i,j))
            B[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="B(%s,%s)" % (i,j))
    model.update()

    #制約条件入力
    for i in N:
        for j in N:
            model.addConstr(quicksum(A[i,k]*C[k,j] for k in N)==B[i,j],
```

```
                name="hidden-k-Prdd-out-Const(%s,%s)"%(i,j))
        J=range(len(A))
        J.remove(i)
        for j in J:
            model.addConstr(C[i,i] >= C[i,j]+1,name= "C-OffDiag-Const(%s,%s)"%(i,j))
            model.addConstr(C[i,i] >= -C[i,j]+1,name= "-C-OffDiag-Const(%s,%s)"%(i,j))
            model.addConstr(B[i,i] >= B[i,j]+1,name= "B-OffDiag-Const(%s,%s)"%(i,j))
            model.addConstr(B[i,i] >= -B[i,j]+1,name= "-B-OffDiag-Const(%s,%s)"%(i,j))
        GG=all_subsets(J)
        for G in GG:
            H=list(set(J)-set(G))
            model.addConstr(k*C[i,i]>=(quicksum(C[i,j] for j in G)
                -quicksum(C[i,j] for j in H))+k+MEPS,name="C-Const(%s,%s)"%(i,G))
            model.addConstr(k*B[i,i]>=(quicksum(B[i,j] for j in G)
                -quicksum(B[i,j] for j in H))+k+MEPS,name="B-Const(%s,%s)"%(i,G))
    model.update()

    #目的関数入力
    #ここも対角成分の最小化である
    model.setObjective(quicksum(C[i,i] for i in N)+quicksum(B[i,i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #表示
    #フラグが False の時は計算不要
    if model.status == GRB.Status.OPTIMAL:
        if mfrag==False:
            return True,[],[]
        CC=[]
        BB=[]
        for i in N:
            rowc=[]
            rowb=[]
            for j in N:
                rowc.append(C[i,j].X)
                rowb.append(B[i,j].X)
            CC.append(rowc)
            BB.append(rowb)
        return True,array(BB),array(CC)
#----------------------------------------
    #R,S
    #フラグが False の時は False と空のリストを返す
    if mfrag==False:
        return False,[],[]
    model=Model("hidden_k_prdd_out")
    AT=transpose(A)
    n=len(A)
    N=range(n)
    kk=int(floor(k))
    d=k-kk
```

```
#変数宣言
R={}
S={}
for i in N:
    for j in N:
        R[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="R(%s,%s)" % (i,j))
        S[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="S(%s,%s)" % (i,j))
model.update()

#制約条件入力
for i in N:
    for j in N:
        model.addConstr(R[i,j]+quicksum(AT[i,k]*S[k,j] for k in N)==0,
            name="KPrdd-Out-Const(%s,%s)"%(i,j))
    J=range(n)
    J.remove(i)
    GG=list(itertools.combinations(J,kk))
    for g in GG:
        G=list(g)
        H=list(set(J)-set(G))
        if H == [] or abs(d)<=MEPS:
            II=all_subsets(G)
            for I in II:
                L=list(set(G)-set(I))
                model.addConstr(R[i,i]>=quicksum(R[i,j] for j in I)
                    -quicksum(R[i,j] for j in L),name="R-Diag-Const(%s,%s)"%(i,G))
                model.addConstr(S[i,i]>=quicksum(S[i,j] for j in I)
                    -quicksum(S[i,j] for j in L),name="S-Diag-Const(%s,%s)"%(i,G))
        else:
            for jj in H:
                II=all_subsets(G)
                for I in II:
                    L=list(set(G)-set(I))
                    model.addConstr(R[i,i]>=d*R[i,jj]+quicksum(R[i,j] for j in I)
                        -quicksum(R[i,j] for j in L),
                            name="R-Diag-plus-Const(%s,%s)"%(i,G))
                    model.addConstr(R[i,i]>=d*-R[i,jj]+quicksum(R[i,j] for j in I)
                        -quicksum(R[i,j] for j in L),
                            name="R-Diag-minus-Const(%s,%s)"%(i,G))
                    model.addConstr(S[i,i]>=d*S[i,jj]+quicksum(S[i,j] for j in I)
                        -quicksum(S[i,j] for j in L),
                            name="S-Diag-plus-Const(%s,%s)"%(i,G))
                    model.addConstr(S[i,i]>=d*-S[i,jj]+quicksum(S[i,j] for j in I)
                        -quicksum(S[i,j] for j in L),
                            name="S-Diag-minus-Const(%s,%s)"%(i,G))
model.addConstr(quicksum(S[i,i] for i in N) >= len(A),
    name= "KPrdd-S-Diag-Const(%s,%s)"%(i,j))
model.update()
```

```
    #目的関数入力
    #S の対角成分の最小化
    model.setObjective(quicksum(S[i,i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #表示
    #グロビオブジェクトをリストに変換
    if model.status == GRB.Status.OPTIMAL:
        RR=[]
        SS=[]
        for i in N:
            rowr=[]
            rows=[]
            for j in N:
                rowr.append(R[i,j].X)
                rows.append(S[i,j].X)
            RR.append(rowr)
            SS.append(rows)
        return False,array(RR),array(SS)

def hidden_znatplus(A,mfrag=True):
    """
    retruns (True,C,B) if A is hidden znat+matrix otherwise (False,R,S).
    if mfrag is False, matrices C,B,R,S will be [].
    """
    model=Model("hidden_Zplus")
    N=range(len(A))

    #変数宣言
    C={}
    B={}
    for i in N:
        C[i,i]=model.addVar(vtype="C",lb=1.0,name="C(%s,%s)" % (i,i))
        B[i,i]=model.addVar(vtype="C",lb=1.0,name="B(%s,%s)" % (i,i))
        J=range(len(N))
        J.remove(i)
        for j in J:
            C[i,j]=model.addVar(vtype="C",ub=-1.0,lb=-GRB.INFINITY,name="C(%s,%s)" % (i,j))
            B[i,j]=model.addVar(vtype="C",ub=-1.0,lb=-GRB.INFINITY,name="B(%s,%s)" % (i,j))
    model.update()

    #制約条件入力
    for i in N:
        for j in N:
            model.addConstr(quicksum(A[i,k]*C[k,j] for k in N)==B[i,j],
                name="Zplus-Const(%s,%s)"%(i,j))
    for i in N:
        model.addConstr(quicksum(B[i,j] for j in N) <=-1,
```

```
          name= "B-OffDiag-Const(%s,%s)"%(i,i))
    model.addConstr(quicksum(C[i,j] for j in N) <=-1,
        name= "C-OffDiag-Const(%s,%s)"%(i,i))
model.update()

#目的関数入力
model.setObjective(quicksum(C[i,i] for i in N)+quicksum(B[i,i] for i in N),GRB.MINIMIZE)
model.optimize()
model.write("test.lp")

#表示
#フラグがFalseなら計算不要
if model.status == GRB.Status.OPTIMAL:
    if mfrag==False:
        return True,True,[],[]
    CC=[]
    BB=[]
    for i in N:
        rowc=[]
        rowb=[]
        for j in N:
            rowc.append(C[i,j].X)
            rowb.append(B[i,j].X)
        CC.append(rowc)
        BB.append(rowb)
    return zplusQ(array(BB)),zplusQ(array(CC)),array(BB),array(CC)
#-----------------------------------------------
##R,S
#フラグがFalseならR,Sのを求める必要はない
if mfrag==False:
    return False,False,[],[]
model=Model("hidden_Zplus")
AT=transpose(A)

#変数宣言
R={}
S={}
for i in N:
    for j in N:
        if i!=j:
            R[i,j]=model.addVar(vtype="C",ub=0,lb=-GRB.INFINITY,name="R(%s,%s)" % (i,j))
            S[i,j]=model.addVar(vtype="C",ub=0,lb=-GRB.INFINITY,name="S(%s,%s)" % (i,j))
        elif i==j:
            R[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="R(%s,%s)" % (i,j))
            S[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="S(%s,%s)" % (i,j))
model.update()

#制約条件入力
for i in N:
```

```
        J=range(len(N))
        J.remove(i)
        for j in N:
            model.addConstr(R[i,j]+quicksum(AT[i,k]*S[k,j] for k in N)==0,
                name="Prdd-Const(%s,%s)"%(i,j))
        for j in J:
            model.addConstr(R[i,i] >= R[i,j],name= "R-Diag-Const(%s,%s)"%(i,j))
            model.addConstr(S[i,i] >= S[i,j],name= "S-Diag-Const(%s,%s)"%(i,j))
    for i in N:
        J=range(len(N))
        J.remove(i)
        model.addConstr(quicksum(S[i,i]-S[i,j] for j in J)+quicksum(-S[i,j] for j in J)>=1,
            name= "S-Diag-Const(%s,%s)"%(i,j))
    model.update()

    #目的関数
    model.setObjective(0*quicksum(S[i,i] for i in N),GRB.MAXIMIZE)
    model.optimize()
    model.write("test.lp")

    #表示
    if model.status == GRB.Status.OPTIMAL:
        RR=[]
        SS=[]
        for i in N:
            rowr=[]
            rows=[]
            for j in N:
                rowr.append(R[i,j].X)
                rows.append(S[i,j].X)
            RR.append(rowr)
            SS.append(rows)
        return zplusQ(array(RR)),zplusQ(array(SS)),array(RR),array(SS)

def hidden_znatminus(A,mfrag=True):
    """
    returns (True,C,B) if A is hidden znat-matrix otherwise (False,R,S).
    if mfrag is False, matrices C,B,R,S will be [].
    """
    model=Model("hidden_Zminus")
    N=range(len(A))

     #変数宣言
    C={}
    B={}
    for i in N:
        for j in N:
            if i==j:
                C[i,j]=model.addVar(vtype="C",lb=1,name="C(%s,%s)" % (i,j))
```

```
                B[i,j]=model.addVar(vtype="C",lb=1,name="B(%s,%s)" % (i,j))
            else:
                C[i,j]=model.addVar(vtype="C",ub=-1,lb=-GRB.INFINITY,name="C(%s,%s)" % (i,j))
                B[i,j]=model.addVar(vtype="C",ub=-1,lb=-GRB.INFINITY,name="B(%s,%s)" % (i,j))
    model.update()

    #制約条件入力
    for i in N:
        for j in N:
            model.addConstr(quicksum(A[i,k]*C[k,j] for k in N)==B[i,j],
                name="Zplus-Const(%s,%s)"%(i,j))
    for i in N:
        model.addConstr(quicksum(B[i,j] for j in N) >=1 ,
            name= "B-OffDiag-Const(%s,%s)"%(i,j))
        model.addConstr(quicksum(C[i,j] for j in N) >=1 ,
            name= "C-OffDiag-Const(%s,%s)"%(i,j))
    model.update()

    #目的関数入力
    #なんでも良い
    model.setObjective(quicksum(C[i,i] for i in N)+quicksum(B[i,i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #フラグがFalseの場合は表示せず
    if model.status == GRB.Status.OPTIMAL:
        if mfrag==False:
            return True,True,[],[]
        CC=[]
        BB=[]
        for i in N:
            rowc=[]
            rowb=[]
            for j in N:
                rowc.append(C[i,j].X)
                rowb.append(B[i,j].X)
            CC.append(rowc)
            BB.append(rowb)
        return zminusQ(array(BB)),zminusQ(array(CC)),array(BB),array(CC)
#-------------------------------------------
    ##R,S
    #フラグがFalseのときはR,Sのを求めない
    if mfrag==False:
        return False,False,[],[]

    model=Model("hidden_Zminus")
    AT=transpose(A)

    #変数宣言
    R={}
```

```
    S={}
    for i in N:
        for j in N:
            if i!=j:
                R[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="R(%s,%s)" % (i,j))
                S[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="S(%s,%s)" % (i,j))
            else:
                R[i,j]=model.addVar(vtype="C",lb=0,name="R(%s,%s)" % (i,j))
                S[i,j]=model.addVar(vtype="C",lb=0,name="S(%s,%s)" % (i,j))
    model.update()

    #制約条件入力
    for i in N:
        J=range(len(N))
        J.remove(i)
        for j in N:
            model.addConstr(R[i,j]+quicksum(AT[i,k]*S[k,j] for k in N)==0,
                name="Prdd-Const(%s,%s)"%(i,j))
        for j in J:
            model.addConstr(R[i,i] >= R[i,j],
                name= "R-offDiag-Const(%s,%s)"%(i,j))
            model.addConstr(S[i,i] >= S[i,j],
                name= "S-offDiag-Const(%s,%s)"%(i,j))
    model.addConstr(quicksum(S[i,i] for i in N) >= len(A),
        name= "S-Diag-Const(%s,%s)"%(i,j))

    #目的関数入力
    model.setObjective(quicksum(S[i,i] for i in N),GRB.MINIMIZE)
    model.optimize()

    #表示
    if model.status == GRB.Status.OPTIMAL:
        RR=[]
        SS=[]
        for i in N:
            rowr=[]
            rows=[]
            for j in N:
                rowr.append(R[i,j].X)
                rows.append(S[i,j].X)
            RR.append(rowr)
            SS.append(rows)
        return zminusQ(array(RR)),zminusQ(array(SS)),array(RR),array(SS)

def hidden_parameter_zmat(A,t,mfrag=True):
    """
    retruns (True,C,B) if A is hidden znat-t-matrix otherwise (False,R,S).
    if mfrag is False, matrices C,B,R,S will be [].
    """
```

```
model=Model("h_Parametric_Zmatrix")

#変数宣言
#Z^o(t) が不等式ではない為 C と B それぞれの α , β が必要
N=range(len(A))
n=len(N)
B={}
C={}
Ba={}
Bb={}
Ca={}
Cb={}
for i in N:
    for j in N:
        B[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="B(%s,%s)" % (i,j))
        C[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="C(%s,%s)" % (i,j))

        Ba[i,j]=model.addVar(vtype="C",ub=GRB.INFINITY,lb=1.0,name="Ba(%s,%s)" % (i,j))
        Ca[i,j]=model.addVar(vtype="C",ub=GRB.INFINITY,lb=1.0,name="Ca(%s,%s)" % (i,j))

        Bb[i,j]=model.addVar(vtype="C",ub=GRB.INFINITY,lb=1.0,name="Bb(%s,%s)" % (i,j))
        Cb[i,j]=model.addVar(vtype="C",ub=GRB.INFINITY,lb=1.0,name="Cb(%s,%s)" % (i,j))
model.update()

#制約条件
#Z^o(t) の定義にそっている
for i in N:
    for j in N:
        model.addConstr(quicksum(A[i,k]*C[k,j] for k in N)==B[i,j],
            name="h_pZ-Const(%s,%s)"%(i,j))
for i in N:
    J=range(len(N))
    J.remove(i)
    model.addConstr(quicksum(Ba[i,j] for j in J)+(1-t)
        *quicksum(Bb[i,j] for j in J)==B[i,i],name="h-PZB-Diag-Const(%s,%s)"%(i,i))
    model.addConstr(quicksum(Ca[i,j] for j in J)+(1-t)
        *quicksum(Cb[i,j] for j in J)==C[i,i],name="h-PZC-Diag-Const(%s,%s)"%(i,i))
for i in N:
    J=range(len(N))
    J.remove(i)
    for j in J:
            model.addConstr(-Ba[i,j]+(1-2*t)*Bb[i,j]==B[i,j],
                name="PZ-B-Const(%s,%s)"%(i,j))
            model.addConstr(-Ca[i,j]+(1-2*t)*Cb[i,j]==C[i,j],
                name="PZ-C-Const(%s,%s)"%(i,j))
model.update()

#目的関数入力
#C,B を最初化すると負の部分が出るので良くない
```

```
    #そのためC,Bのα, βを最小化
    model.setObjective(quicksum(Ba[i,i] for i in N)
        +quicksum(Bb[i,i] for i in N)+quicksum(Ca[i,i] for i in N)
            +quicksum(Cb[i,i] for i in N),GRB.MINIMIZE)
    model.optimize()

    if model.status == GRB.Status.OPTIMAL:
        if mfrag==False:
            return True,[],[]
        BB=[]
        CC=[]
        for i in N:
            rowb=[]
            rowc=[]
            for j in N:
                rowb.append(B[i,j].X)
                rowc.append(C[i,j].X)
            BB.append(rowb)
            CC.append(rowc)
        return True,array(BB),array(CC)
#-------------------------------------------
    ##R,S
    #フラグがFalseのときはR,Sを求めない
    if mfrag==False:
        return False,[],[]

    model=Model("h_Parametric_Zmatrix")
    AT=transpose(A)

    #変数宣言
    R={}
    S={}
    for i in N:
        for j in N:
            R[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="R(%s,%s)" % (i,j))
            S[i,j]=model.addVar(vtype="C",lb=-GRB.INFINITY,name="S(%s,%s)" % (i,j))
    model.update()

    #制約条件入力
    for i in N:
        J=range(len(N))
        J.remove(i)
        for j in N:
            model.addConstr(R[i,j]+quicksum(AT[i,k]*S[k,j] for k in N)==0,
                name="Prdd-Const(%s,%s)"%(i,j))
        for j in J:
            model.addConstr(R[i,i]-R[i,j]>=0,name= "R-Diag-Const(%s,%s)"%(i,j))
            model.addConstr((1-t)*R[i,i] + (1-(2*t))*R[i,j] >=0,
                name= "R-offDiag-Const(%s,%s)"%(i,j))
```

```
            model.addConstr(S[i,i]-S[i,j] >=0,name= "S-Diag-Const(%s,%s)"%(i,j))
            model.addConstr((1-t)*S[i,i] + (1-(2*t))*S[i,j] >=0,
                name= "S-offDiag-Const(%s,%s)"%(i,j))

    #Non-Zero 条件
    model.addConstr(quicksum(quicksum((2-t)*S[i,i]-2*t*S[i,j] for j in range(0,i))
        + quicksum((2-t)*S[i,i]-2*t*S[i,j] for j in range(i+1,n))
            for i in range(n)) >= n*(n-1)*2,name= "S-nonzero-Const")
    model.update()
    #目的関数入力
    model.setObjective(0,GRB.MINIMIZE)
    model.optimize()
    #表示
    if model.status == GRB.Status.OPTIMAL:
        RR=[]
        SS=[]
        for i in N:
            rowr=[]
            rows=[]
            for j in N:
                rowr.append(R[i,j].X)
                rows.append(S[i,j].X)
            RR.append(rowr)
            SS.append(rows)
        return False,array(RR),array(SS)

def s_and_znat(A):
    model=Model("smatrix_and_znat")
    N=range(len(A))

    #変数宣言
    C={}
    B={}
    for i in N:
        for j in N:
            if i==j:
                C[i,j]=model.addVar(vtype="C",lb=1,name="C(%s,%s)" % (i,j))
                B[i,j]=model.addVar(vtype="C",lb=1,name="B(%s,%s)" % (i,j))
            else:
                C[i,j]=model.addVar(vtype="C",ub=-1,lb=-GRB.INFINITY,name="C(%s,%s)" % (i,j))
                B[i,j]=model.addVar(vtype="C",ub=-1,lb=-GRB.INFINITY,name="B(%s,%s)" % (i,j))
    x={}
    for i in N:
        x[i]=model.addVar(vtype="C",lb=1,name="x(%s)" % (i))
    model.update()

    #制約入力
    for i in N:
        model.addConstr(quicksum(A[i,j]*x[j] for j in N)>=1,name="Smatrix-Const(%s)"%(i))
```

```
    for i in N:
        for j in N:
            model.addConstr(quicksum(A[i,k]*C[k,j] for k in N)==B[i,j],
                name="Zplus-Const(%s,%s)"%(i,j))

    #目的関数入力
    model.setObjective(quicksum(C[i,i] for i in N)+quicksum(B[i,i] for i in N),GRB.MINIMIZE)
    model.setObjective(quicksum(x[i] for i in N),GRB.MINIMIZE)
    model.optimize()

    if model.status == GRB.Status.OPTIMAL:
        return True
    else:
        return False

def matrix_create(n):
    ##行列生成対角成分がn-1でそれ以外が-1の行列を作る。
    ##前研究の特集な例.k-prdd-inだがPではない
    A=[[0 for j in range(n)] for i in range(n)]
    for i in range(n):
        for j in range(n):
            if j==i:
                A[i][j]=n-1
            else:
                A[i][j]=-1
    return array(A)

def matrix_create2(n,t):
    #ただしtは0<t<=1
    At=(1.0/(2+t*t))*(array([[t*t,0,2,2*t],[0,t*t,-2*t,2],[-2,2*t,t*t,0],[-2*t,-2,0,t*t]]))
    A=array([[0.0 for j in range(n)] for i in range(n)])
    for i in range(n):
        for j in range(n):
            if i<=3 and j<=3:
                A[i,j]=At[i,j]
            else:
                if i==j:
                    A[i,j]=1.0
    return A
```

(3:jikkenn22.py) 主にパラメータの t や k を狭める実験のために用いたソースコードである.

```
#! usr/bin/env python
# coding:utf-8
from numpy import *
from prdd_and_znat import *
from gurobipy import *
import hidden_matrix2 as hm2
import time
from pylab import *
import datetime
import random
import csv
MEPS=1.0e-8

##k の最小値を二部探索で見つける
def find_min_t(A):
    #正の範囲での二部探索
    mint=2.0/3.0
    maxt=1.0

    ##最小値、最大値どちらも入るような場合は除外する
    f,C,B=hm2.hidden_parameter_zmat(A,mint,False)
    ff,C,B=hm2.hidden_parameter_zmat(A,maxt,False)
    if f==True and ff==True:
        return 0
    if f==True:
        return mint
    else:
        if not(ff)==True:
            print '!!!!'
            return 0
        while abs(maxt-mint)>0.001:
            mid=(maxt+mint)/2.0
            f,C,B=hm2.hidden_parameter_zmat(A,mid,False)
            if f==True:
                maxt=mid
            else:
                mint=mid
    return mid

def find_min_k(A):
    #hidden-k-prdd-out について
    mink=1
    maxk=2

    #どちらの探索にも引っかかる場合は意味が無いので弾く
    f,C,B=hm2.hidden_k_prdd_out(A,mink,False)
    ff,C,B=hm2.hidden_k_prdd_out(A,maxk,False)
    if f==True and ff==True:
```

```
        return 1

    if f==True:
        return mink
    else:
        #どちらにも入らない場合はなにかおかしい
        f2,C,B=hm2.hidden_k_prdd_out(A,maxk,False)
        if not(f2)==True:
            print '!!!!'
            return 1
        while abs(maxk-mink)>0.001:
            mid=(maxk+mink)/2.0
            f,C,B=hm2.hidden_k_prdd_out(A,mid,False)
            if f==True:
                maxk=mid
            else:
                mink=mid
    return mid

def find_min_t2(A):
    #0<t<0.6 の範囲での二部探索
    mint=0
    maxt=2.0/3.0

    f,C,B=hm2.hidden_parameter_zmat(A,mint,False)
    ff,C,B=hm2.hidden_parameter_zmat(A,maxt,False)
    if f==True and ff==True:
        return 1
    elif ff==True:
        return maxt
    else:
        f,C,B=hm2.hidden_parameter_zmat(A,mint,False)
        if not(f)==True:
            print '!!!!'
            return 1
        while abs(maxt-mint)>0.001:
            mid=(maxt+mint)/2.0
            f,C,B=hm2.hidden_parameter_zmat(A,mid,False)
            if f==True:
                mint=mid
            else:
                maxt=mid
    return mid

def find_min_t3(A):
    #-2<t<-1 の範囲での二部探索
    mint=-2
    maxt=-1
```

```
    f,C,B=hm2.hidden_parameter_zmat(A,mint,False)
    ff,C,B=hm2.hidden_parameter_zmat(A,maxt,False)
    if f==True and ff==True:
        return 1
    elif ff==True:
        return maxt
    else:
        f,C,B=hm2.hidden_parameter_zmat(A,mint,False)
        if not(f)==True:
            print '!!!!'
            return 1
        while abs(maxt-mint)>0.001:
            mid=(maxt+mint)/2.0
            f,C,B=hm2.hidden_parameter_zmat(A,mid,False)
            if f==True:
                mint=mid
            else:
                maxt=mid
    return mid

def random_kprddout(n,k):
    X=zeros((n,n))
    for i in range(n):
        J=range(n)
        J.remove(i)
        for j in J:
            X[i][j]=random.uniform(-1,1)
        X[i][i]=max(sum(abs(X[i]))/k,max(abs(X[i])))
    return X

def random_hidden_kprddout(n,k1,k2):
    while True:
        C = random_kprddout(n,k1)
        if abs(linalg.det(C)) > MEPS:
            break
    B = random_kprddout(n,k2)
    A = dot(B,linalg.inv(C))
    A *= 10000
    for i in range(len(A)):
        for j in range(len(A[i])):
            A[i,j] = int(A[i,j])
    return A

def jikken_znat(n):
    #パラメーター Z 行列の t の最大値を求める
    #範囲は 0.6666666<=t<=1, 生成行列数は 10000
    #シードは行列生成時の k1,k2 の値が変わるときに日時より作成
    #戻り値の形式は辞書で (パラメーターの結果,P 行列の割合, シード)
    repeats=1000
```

```
    B={}
    for i in [n1*0.5 for n1 in range(2,10)]:
        for j in [n2*0.5 for n2 in range(2,10)]:
            day = datetime.datetime.now()
            seed = int(long(date2num(day)*10000000000)%1000000)
            random.seed(seed)
            count=0.0
            mint2=0
            for ii in range(repeats):
                A=random_hidden_kprddout(n,i,j)
                if pmatrixQ(A)==True:
                    count+=1
                    x=find_min_t(A)
                    #最大値に更新
                    if x>mint2:
                        mint2=x
            B[i,j]=(mint2,(count/repeats)*100.0,seed)
    return B

def jikken_znat2(n):
    #パラメーターZ行列のtの最小を求める
    #範囲は0.6666666<=t<=1, 生成行列数は10000
    #シードは行列生成時のk1,k2の値が変わるときに日時より作成
    #戻り値の形式は辞書で(パラメーターの結果,P行列の割合, シード)
    repeats=1000
    B={}
    for i in [n1*0.5 for n1 in range(2,10)]:
        for j in [n2*0.5 for n2 in range(2,10)]:
            day = datetime.datetime.now()
            seed = int(long(date2num(day)*10000000000)%1000000)
            random.seed(seed)
            count=0.0
            mint2=2.0/3.0
            for ii in range(repeats):
                A=random_hidden_kprddout(n,i,j)
                if pmatrixQ(A)==True:
                    count+=1
                    x=find_min_t2(A)
                    #最小値に更新
                    if mint2>x:
                        mint2=x
            B[i,j]=(mint2,(count/repeats)*100.0,seed)
    return B

def jikken_znat3(n):
    #パラメーターZ行列のtの最小を求める
    #範囲は0.6666666<=t<=1, 生成行列数は10000
    #シードは行列生成時のk1,k2の値が変わるときに日時より作成
    #戻り値の形式は辞書で(パラメーターの結果,P行列の割合, シード)
```

```
    repeats=1000
    B={}
    for i in [n1*0.5 for n1 in range(2,10)]:
        for j in [n2*0.5 for n2 in range(2,10)]:
            day = datetime.datetime.now()
            seed = int(long(date2num(day)*10000000000)%1000000)
            random.seed(seed)
            count=0.0
            mint2=-1
            for ii in range(repeats):
                A=random_hidden_kprddout(n,i,j)
                if pmatrixQ(A)==True:
                    count+=1
                    x=find_min_t2(A)
                    #最小値に更新
                    if mint2>x:
                        mint2=x
            B[i,j]=(mint2,(count/repeats)*100.0,seed)
    return B

def jikken_kprddout(n):
    #kprddout 行列の k の最大値を求める
    #範囲は 1<=k<=2, 生成行列数は 1000
    #シードは行列生成時の k1,k2 の値が変わるときに日時より作成
    #戻り値の形式は辞書で (パラメーターの結果,P 行列の割合, シード)
    B={}
    repeats=1000
    for i in [n1*0.5 for n1 in range(2,10)]:
        for j in [n2*0.5 for n2 in range(2,10)]:
            day = datetime.datetime.now()
            seed = int(long(date2num(day)*10000000000)%1000000)
            random.seed(seed)
            maxk3=1
            count=0.0
            for ii in range(repeats):
                A=random_hidden_kprddout(n,i,j)
                if pmatrixQ(A)==True:
                    count+=1
                    tempk=find_min_k(A)
                    if tempk>maxk3:
                        maxk3=tempk
            B[i,j]=(maxk3,(count/repeats)*100,seed)
    return B

def jikken_kprddout2(n):
    #kprddout 行列の k の最大値を求める
    #範囲は 1<=k<=2, 生成行列数は 1000
    #シードは行列生成時の k1,k2 の値が変わるときに日時より作成
    #戻り値の形式は辞書で (パラメーターの結果,P 行列の割合, シード)
```

```
    B={}
    repeats=1000
    for i in [n1*0.5 for n1 in range(2,10)]:
        for j in [n2*0.5 for n2 in range(2,10)]:
            day = datetime.datetime.now()
            seed = int(long(date2num(day)*10000000000)%1000000)
            random.seed(seed)
            maxk3=1
            count=0.0
            for ii in range(repeats):
                A=random_hidden_kprddout(n,i,j)
                kekka1,kekka2=hidden_znatplus(A,mfrag=True)
                if kekka1==True and kekka2==True:
                    count+=1
                    tempk=find_min_k(A)
                    if tempk>maxk3:
                        maxk3=tempk
            B[i,j]=(maxk3,(count/repeats)*100,seed)
    return B

def writepara(results,n):
    #結果を表としてcsv形式で出力
    #書き込みは行ごとにしかできない
    FILE="parameter_result.csv"
    f=open(FILE,'ab')
    c = csv.writer(f)
    c.writerow(['n=',n])
    title=[]
    title.append("k1/k2")
    se=[]
    print 'k1/k2\t',
    for x in [n1*0.5 for n1 in range(2,10)]:
        #一番上は実験結果は入らないので個別に行書き込み
        title.append(str(x))
        print x,'\t',
    print ''
    c.writerow(title)
    for i in [n2*0.5 for n2 in range(2,10)]:
        #行の先頭にkの値を入れてそれ以降に実験結果を書き込み
        csvfile=[]
        csvfile.append(str(i))
        print i,'\t',
        for j in [n2*0.5 for n2 in range(2,10)]:
            z=results[i,j][0]
            se.append(results[i,j][2])
            csvfile.append(z)
            print results[i,j][0],'\t',
        print ''
        c.writerow(csvfile)
```

```
    c.writerow('')
    f.close()

    g = open('seedmemo.txt', 'ab') # 書き込みモードで開く
    nn=("n=",n)
    g.write(str(nn))
    g.write("")
    g.write(str(se)) # 引数の文字列をファイルに書き込む
    g.write("")
    g.close()

def writepara2(results,n):
    #結果を表として csv 形式で出力
    #書き込みは行ごとにしかできない
    FILE="parameter_result2.csv"
    f=open(FILE,'ab')
    c = csv.writer(f)
    c.writerow(['n=',n])
    title=[]
    title.append("k1/k2")
    se=[]
    print 'k1/k2\t',
    for x in [n1*0.5 for n1 in range(2,13)]:
        #一番上は実験結果は入らないので個別に行書き込み
        title.append(str(x))
        print x,'\t',
    print ''
    c.writerow(title)
    for i in [n2*0.5 for n2 in range(2,13)]:
        #行の先頭に k の値を入れてそれ以降に実験結果を書き込み
        csvfile=[]
        csvfile.append(str(i))
        print i,'\t',
        for j in [n2*0.5 for n2 in range(2,13)]:
            z=results[i,j][0]
            se.append(results[i,j][2])
            csvfile.append(z)
            print results[i,j][0],'\t',
        print ''
        c.writerow(csvfile)
    c.writerow('')
    f.close()

    g = open('seedmemo.txt', 'ab') # 書き込みモードで開く
    nn=("n=",n)
    g.write(str(nn))
    g.write("")
    g.write(str(se)) # 引数の文字列をファイルに書き込む
    g.write("")
```

```
    g.close()

def writeprdd(results,n):
    #結果を表として csv 形式で出力
    #書き込みは行ごとにしかできない
    FILE="prdd_result.csv"
    f=open(FILE,'ab')
    c = csv.writer(f)
    c.writerow(['n=',n])
    title=[]
    title.append("k1/k2")
    se=[]
    print 'k1/k2\t',
    for x in [n1*0.5 for n1 in range(2,10)]:
        #一番上は実験結果は入らないので個別に行書き込み
        title.append(str(x))
        print x,'\t',
    print ''
    c.writerow(title)
    for i in [n2*0.5 for n2 in range(2,10)]:
        #行の先頭に k の値を入れてそれ以降に実験結果を書き込み
        csvfile=[]
        csvfile.append(str(i))
        print i,'\t',
        for j in [n2*0.5 for n2 in range(2,10)]:
            z=results[i,j][0]
            se.append(results[i,j][2])
            csvfile.append(z)
            print results[i,j][0],'\t',
        print ''
        c.writerow(csvfile)
    c.writerow('')
    f.close()

    g = open('seedmemo.txt', 'ab') # 書き込みモードで開く
    nn=("n=",n)
    g.write(str(nn))
    g.write(" ")
    g.write(str(se)) # 引数の文字列をファイルに書き込む
    g.write(" ")
    g.close()

def writepmat(results,n):
    #結果を表として csv 形式で出力
    #書き込みは行ごとにしかできない
    FILE="pmat.csv"
    f=open(FILE,'ab')
    c = csv.writer(f)
    c.writerow(['n=',n])
```

```
    title=[]
    title.append("k1/k2")

    print 'k1/k2\t',
    for x in [n1*0.5 for n1 in range(2,10)]:
        #一番上は実験結果は入らないので個別に行書き込み
        title.append(str(x))
        print x,'\t',
    print ''
    c.writerow(title)
    for i in [n1*0.5 for n1 in range(2,10)]:
        #行の先頭にkの値を入れてそれ以降に実験結果を書き込み
        csvfile=[]
        csvfile.append(str(i))
        print i,'\t',
        for j in [n2*0.5 for n2 in range(2,10)]:
            z=results[i,j][1]
            csvfile.append(z)
            print results[i,j][1],'\t',
        print ''
        c.writerow(csvfile)

    c.writerow('')
    f.close()

def writepmat2(results,n):
    #結果を表としてcsv形式で出力
    #書き込みは行ごとにしかできない
    FILE="pmat2.csv"
    f=open(FILE,'ab')
    c = csv.writer(f)
    c.writerow(['n=',n])
    title=[]
    title.append("k1/k2")

    print 'k1/k2\t',
    for x in [n1*0.5 for n1 in range(2,10)]:
        #一番上は実験結果は入らないので個別に行書き込み
        title.append(str(x))
        print x,'\t',
    print ''
    c.writerow(title)
    for i in [n1*0.5 for n1 in range(2,10)]:
        #行の先頭にkの値を入れてそれ以降に実験結果を書き込み
        csvfile=[]
        csvfile.append(str(i))
        print i,'\t',
        for j in [n2*0.5 for n2 in range(2,10)]:
            z=results[i,j][1]
```

```
            csvfile.append(z)
            print results[i,j][1],'\t',
        print ''
        c.writerow(csvfile)

    c.writerow('')
    f.close()

def writepmat3(results,n):
    #結果を表として csv 形式で出力
    #書き込みは行ごとにしかできない
    FILE="pmat3.csv"
    f=open(FILE,'ab')
    c = csv.writer(f)
    c.writerow(['n=',n])
    title=[]
    title.append("k1/k2")

    print 'k1/k2\t',
    for x in [n1*0.5 for n1 in range(2,10)]:
        #一番上は実験結果は入らないので個別に行書き込み
        title.append(str(x))
        print x,'\t',
    print ''
    c.writerow(title)
    for i in [n1*0.5 for n1 in range(2,10)]:
        #行の先頭に k の値を入れてそれ以降に実験結果を書き込み
        csvfile=[]
        csvfile.append(str(i))
        print i,'\t',
        for j in [n2*0.5 for n2 in range(2,10)]:
            z=results[i,j][1]
            csvfile.append(z)
            print results[i,j][1],'\t',
        print ''
        c.writerow(csvfile)

    c.writerow('')
    f.close()
```

(4:PrincipalPivotMethod.py)PPM の実験用である.

```
#! usr/bin/env python
# coding:utf-8


#PrincipalPivotMethod:PPM

from prdd_and_znat import *
import hidden_matrix2 as hm2
import numpy as np
import time
from pylab import *
import datetime
import random
import csv

MEPS=1.0e-8

def create_vector(n):
    #負の範囲での任意のベクトル生成
    q=np.array([random.uniform(-1,-MEPS) for i in range(n)])
    q=10000*q
    for i in range(n):
        q[i]=int(q[i])
    return q

def create_vector2(n):
    #すべての範囲での任意のベクトル生成
    q=np.array([random.uniform(-1,1) for i in range(n)])
    q=10000*q
    for i in range(n):
        q[i]=int(q[i])
    return q

def create_vector3(n):
    return -10000*np.ones(n)

def PPM(M,q):
    #線形相補性問題のための PPM.y=Mx+q を-Mx+Iy=q すなわち [-M,I]x=q として改訂シンプレックス法
    #準備:B は初期基底,N は非基底,R は添字集合,A=[-M,I]
    n=len(M)
    A=np.c_[-M,np.identity(n)]
    R=range(n)
    B=range(n,2*n)
    N=range(n)
    roop=0

    #アルゴリズム本体
```

```
    while 1:
        #基底部分の行列(ベクトル)とそれ以外で分ける
        roop+=1
        Ab=A[np.ix_(R,B)]
        An=A[np.ix_(R,N)]
        #Abの逆行列とqを計算
        qb=np.linalg.solve(Ab,q)
        frag=0
        #終了判定:qのすべての要素が正ならx=0とすればいいので終わり
        for i in R:
            if qb[i]<0:
                frag =1
                break
        #基底更新の代わりに添字交換
        if frag ==1:
            B[i],N[i]=N[i],B[i]
        #終了:ループ回数と基底解を返す
        else:
            x=np.zeros(2*n)
            x[np.ix_(B)]=qb
            return True,x[np.ix_(range(n))],roop
        if roop>=n**2:
            return False,[0],0

def ZnatMinus(n):
    #対角成分を大きくし過ぎると単位行列を同じような働きをしてしまうのでダメ
    #作り方を考えなおす
    #Z^o-
    M=np.zeros((n,n))
    for i in range(n):
        J=list(range(n))
        J.remove(i)
        for j in J:
             M[i][j]=random.uniform(-1,-MEPS)
        M[i][i]=random.uniform(abs(sum(M[i]))+MEPS,abs(sum(M[i]))+1)
    return M

def ZnatPlus(n):
    #対角成分を大きくし過ぎると単位行列を同じような働きをしてしまうのでダメ
    #Z^o+
    M=np.zeros((n,n))
    for i in range(n):
        J=list(range(n))
        J.remove(i)
        for j in J:
             M[i][j]=random.randint(-2,-1)
        M[i][i]=random.randint(0,abs(sum(M[i]))-1)
    return M
```

```
def prddone(n,k):
    X=zeros((n,n))
    for i in range(n):
        J=range(n)
        J.remove(i)
        for j in J:
            X[i][j]=random.uniform(-1,1)
        X[i][i]=max(sum(abs(X[i]))/k,max(abs(X[i])))
    return X

def two_thirds(n):
    #t=2/3 のときの Z^o(t)
    M=np.zeros((n,n))
    for i in range(n):
        J=list(range(n))
        J.remove(i)
        for j in J:
             M[i][j]=random.uniform(-2,0)
        M[i][i]=abs(sum(M[i]))
    M *= 100
    for i in range(len(M)):
        for j in range(len(M[i])):
            M[i,j] = int(M[i,j])
    return M

def HZMinusCreate(n):
    while True:
        C = ZnatMinus(n)
        if abs(np.linalg.det(C)) > MEPS:
            break
    B=np.array(ZnatMinus(n))
    CC=np.linalg.inv(C)
    A=np.dot(B,CC)
    A *= 10000
    for i in range(len(A)):
        for j in range(len(A[i])):
            A[i,j] = int(A[i,j])
    return A

def HZPlusCreate(n):
    while True:
        C = ZnatMinus(n)
        if abs(np.linalg.det(C)) > MEPS:
            break
    B=np.array(ZnatPlus(n))
    CC=np.linalg.inv(C)
    A=np.dot(B,CC)
    A *= 10000
    for i in range(len(A)):
```

```
        for j in range(len(A[i])):
            A[i,j] = int(A[i,j])
    return A

def HPrddOne(n,k1,k2):
    while 1:
        while True:
            C = prddone(n,k1)
            if abs(np.linalg.det(C)) > MEPS:
                break
        B=np.array(prddone(n,k2))
        CC=np.linalg.inv(C)
        A=np.dot(B,CC)
        A *= 10000
        for i in range(len(A)):
            for j in range(len(A[i])):
                A[i,j] = int(A[i,j])
        r1,r2,m1,m2=hm2.hidden_znatminus(A,False)
        a,b,c,d=hm2.hidden_prdd1Q(A,False)
        if r1==False and a==True:
            return A
    #return A

def algorithm_exp1(n,m):
    #4 × 4 の最大となる seed を 25 個求めるだけ
    #4 × 4 以外も別に可能
    #最大回数の目星は付けておくこと . でないと seed 追加の判別が困難
    seeds=[]
    roopmax={}
    matrix=[]
    while 1:
        #繰り返し回数最大となる seed 取り出し
        day = datetime.datetime.now()
        seed = int(long(date2num(day)*10000000000)%1000000)
        random.seed(seed)
        M=HZMinusCreate(n)
        q=create_vector3(n)
        result,x,roop=PPM(M,q)
        if roop>=2*n+2:
            seeds.append(seed)
            #matrix.append(M)
        if len(seeds)>=m:
            break
    for i in range(1,m+1):
        #取り出した seed を使い over_create で n × n 行列を対角に並べた行列作成
        #その行列を使い n から m*n まで n 飛ばしで PPM 実行
        #理論的には最大のループ回数を対角に持ってきているので他のループ回数も最大なはず
        print i*n
        A=over_create(n,seeds[:i])
```

```
        print len(A[1])
        print seeds[:i]
        results,xx,roop2=PPM(A,create_vector3(i*n))
        roopmax[i*n]=roop2
    return roopmax

def search_maxk(n,m):
    #k-prdd-out のちょうどいい k の組を見つける
    #平均取って一番デカイので良い
    #seeds=[]
    roopavg={}
    for k1 in [1.2,1.3,1.4,1.5,1.6,1.7]:
        temp=0.0
        for k2 in [1.2,1.3,1.4,1.5,1.6,1.7]:
            for i in range(m):
                #繰り返し回数最大となる seed 取り出
                day = datetime.datetime.now()
                seed = int(long(date2num(day)*10000000000)%1000000)
                random.seed(seed)
                M=HPrddOne(n,k1,k2)
                q=create_vector3(n)
                result,x,roop=PPM(M,q)
                temp+=roop
                #if roop>=3*n:
                #    seeds.append(seed)
                # #if len(seeds)>=m:
                #    break
        roopavg[k1,k2]=temp/m
    return list(max(roopavg))

def algorithm_exp2(n,k1,k2,m):
    #4 × 4 の最大となる seed を 25 個求めるだけ
    #4 × 4 以外も別に可能
    #最大回数の目星は付けておくこと . でないと seed 追加の判別が困難
    seeds=[]
    roopmax={}
    matrix=[]
    while 1:
        #繰り返し回数最大となる seed 取り出し
        day = datetime.datetime.now()
        seed = int(long(date2num(day)*10000000000)%1000000)
        random.seed(seed)
        M=HPrddOne(n,k1,k2)
        q=create_vector3(n)
        result,x,roop=PPM(M,q)
        if roop>=13:
            seeds.append(seed)
            matrix.append(M)
        if len(seeds)>=m:
```

```
            break
    for i in range(1,m+1):
        #取り出したseedを使いover_createでn×n行列を対角に並べた行列作成
        #その行列を使いnからm*nまでn飛ばしでPPM実行
        #理論的には最大のループ回数を対角に持ってきているので他のループ回数も最大なはず
        print i*n
        A=over_create2(n,k1,k2,seeds[:i])
        print len(A[1])
        print seeds[:i]
        results,xx,roop2=PPM(A,create_vector3(i*n))
        roopmax[i*n]=roop2
    return roopmax,matrix


def algorithm_exp3(n,k1,k2,m):
    roop={}
    roopmax={}
    avg={}
    infinity={}
    seeds={}
    see=[]
    for i in n:
        temp=[]
        tempsum=0.0
        for j in range(m):

            day = datetime.datetime.now()
            seed = int(long(date2num(day)*10000000000)%1000000)
            random.seed(seed)

            M=HPrddOne(i,k1,k2)
            q=create_vector3(i)

            result,x,roop2=PPM(M,q)
            if result==True:
                #最大値を求めるため一時保存
                temp.append(roop2)
                #平均を求めるため合計を一時保存
                tempsum+=roop2
                #すべてのループ回数保存
                roop[i,j]=roop2
            if result==False:
                infinity[i,j]=M
                seeds[i,j]=seed
            if roop2>3*i:
                see.append(seed)
        #合計数をtempの要素数(繰り返し数)で割ることで平均を求める
        avg[i]=tempsum/len(temp)
        roopmax[i]=max(temp)
```

```
    return infinity,roop,roopmax,avg,see

def over_create(n,seed):
    #n × n 列のループ回数最大となる seed から nlen(seed) × nlen(seed) 行列を作り実験する関
数
    A=np.zeros((len(seed)*n,len(seed)*n))
    frag=0
    for se in seed:
        random.seed(se)

        M=HZMinusCreate(n)
        frag+=1
        for i in range(n*frag-n,frag*n):
            for j in range(n*frag-n,frag*n):
                if i<=frag*n-1 and j<=frag*n-1:
                    A[i,j]=M[i-(frag-1)*n,j-(frag-1)*n]
    return A

def over_create2(n,k1,k2,seed):
    #n × n 列のループ回数最大となる seed から nlen(seed) × nlen(seed) 行列を作り実験する関
数
    A=np.zeros((len(seed)*n,len(seed)*n))
    frag=0
    for se in seed:
        random.seed(se)

        M=HPrddOne(n,k1,k2)
        frag+=1
        for i in range(n*frag-n,frag*n):
            for j in range(n*frag-n,frag*n):
                if i<=frag*n-1 and j<=frag*n-1:
                    A[i,j]=M[i-(frag-1)*n,j-(frag-1)*n]
    return A

def write_algomax(roopmax,n):
    #結果を表として csv 形式で出力
    #書き込みは行ごとにしかできない
    FILE="PPM.csv"
    f=open(FILE,'ab')
    c = csv.writer(f)
    c.writerow(['PPM'])
    title=[]
    title.append("n")

    print 'n\t',
    for x in n:
        #一番上は実験結果は入らないので個別に行書き込み
        title.append(str(x))
```

```
        print x,'\t',
    print ''
    c.writerow(title)
    csvfile=[]
    csvfile.append(" ")
    print " ",'\t',
    for i in n:
        #行の先頭にkの値を入れてそれ以降に実験結果を書き込み
        z=roopmax[i]
        csvfile.append(z)
        print roopmax[i],'\t',
        #print ''
    c.writerow(csvfile)
    c.writerow('')
    f.close()

def write_algoavg(avg,n):
    #結果を表としてcsv形式で出力
    #書き込みは行ごとにしかできない
    FILE="PPM.csv"
    f=open(FILE,'ab')
    c = csv.writer(f)
    c.writerow(['PPM'])
    title=[]
    title.append("n")

    print 'n\t',
    for x in n:
        #一番上は実験結果は入らないので個別に行書き込み
        title.append(str(x))
        print x,'\t',
    print ''
    c.writerow(title)

    for i in n:
        #行の先頭にkの値を入れてそれ以降に実験結果を書き込み
        csvfile=[]
        csvfile.append([" "])
        print " ",'\t',
        z=avg[i]
        csvfile.append(z)
        print avg[i],'\t',
        print ''
        c.writerow(csvfile)
    c.writerow('')
    f.close()

def extra_exp(A,n):
    #seedではなく行列を入力して実験するタイプの実験
```

```
#ネスト多すぎるので改良のよりあり
result={}
for i in range(1,(n/len(A))+1):
    AA=np.zeros((i*len(A),i*len(A)))
    for m in range(1,i+1):
        for j in range(len(AA)):
            for k in range(len(AA)):
                if len(A)*(m-1)<=j<len(A)*m and len(A)*(m-1)<=k<len(A)*m:
                    AA[j,k]=A[j-len(A)*m,k-len(A)*m]
    result1,vec,roop=PPM(AA,create_vector3(len(AA)))
    result[len(AA)]=roop
return result,AA
```