

Maxima による Nash 均衡の計算

中 川 義 行
山 下 章 夫

キーワード

2 人非ゼロ和行列ゲーム

Maxima

Nash 均衡

線形相補方程式

1 はじめに

本稿は 2 人非ゼロ和行列ゲームの Nash 均衡について考察したものである。2 人非ゼロ和行列ゲームの Nash 均衡を求める方法は McKelvey-McLennan [3] の展望論文によれば, Liapunov 関数を使用する方法や線形相補方程式の解を利用する方法など, 幾つかの方法が提案されている。以下では, 線形相補方程式を解くことによって Nash 均衡を求める。

次節では 2 人非ゼロ和行列ゲームにおける Nash 均衡の定義を述べる。Nash 均衡とは, 各プレイヤーが相手の戦略を所与として自己の利得を最大化することによって与えられる均衡である。このことは両方のプレイヤーが各々, 線形計画問題を解くことを意味する。

第 3 節では, 特にプレイヤーの持つ純粋戦略が 2 個の場合の Nash 均衡の存在を, シミュレーションによって図示する。そこでは, プレイヤーの最適反応が, 利得行列の成分の組み合わせに依存して様々に変化することを示す。

第 4 節では, 一般の 2 人非ゼロ和行列ゲームの Nash 均衡を求めるために, 各プレイヤーの線形計画問題の双対問題を定式化することによって, 線形相補方程式を導く。

第 5 節では, この線形相補方程式から Nash 均衡を Maxima [5] を用いて求めるためのアルゴリズムを述べる。Nash 均衡は線形相補方程式の非負解に対応するが, 線形相補方程式の解のなかには不定解も存在する。この節では, 不定解以外の非負解を取り出す方法を示す。

第 6 節では, 残された課題について述べる。

付録では, Maxima のインストール方法と Maxima プログラム, およびその実行例を掲載する。

本研究¹⁾では, 利用するソフトウェアとして Maxima というフリーの数式処理システムを採用した。これは Microsoft Excel などの数値計算ソフトウェアと異なり, 数式を記号的に処理するため, 未定係数を含んだまま計算を実行できるので, 誤差のない厳密解を得ることが可能である。そのため, 理論的な研究を行うのに適していると判断した。また, 類似する商用ソフトウェアとしては *Mathematica* や *Maple* が有名である。だが, 入手価格は他の教育用ソフトウェアと比較して相対的に高価²⁾である。それゆえ, 外観や操作性などの瑣末な点には問題があっても, 計算結果は商用

1) この研究は, 龍谷大学経営学会の研究部会「社会科学の研究や教育における Maxima の応用」の共同研究の一部である。

2) 例えば, 以下の URL に *Mathematica* や *Maple* の最新価格が公表されている。

のものと何ら遜色ないフリーウェアを採用した。

2 定 義

以下では、2 人のプレイヤーを、それぞれプレイヤー I とプレイヤー II とする。プレイヤー I は m 個の純粋戦略を持ち、プレイヤー II は n 個の純粋戦略を持つものとする。各プレイヤーの利得行列を、それぞれ A, B とする。各プレイヤーの混合戦略の集合を

$$X = \left\{ \vec{x} \in R^m \mid \sum_{i=1}^m x_i = 1, x_i \geq 0, i = 1, \dots, m \right\} \quad (1)$$

$$Y = \left\{ \vec{y} \in R^n \mid \sum_{i=1}^n y_i = 1, y_i \geq 0, i = 1, \dots, n \right\} \quad (2)$$

で表す。所与の混合戦略 $\vec{x}^* \in X, \vec{y}^* \in Y$ に対して、プレイヤー I, II の期待利得は、それぞれ $\vec{x}^{*'} A \vec{y}^*, \vec{x}^{*'} B \vec{y}^*$ である³⁾。

このとき、混合戦略 Nash 均衡は、すべての $\vec{x} \in X$, および、すべての $\vec{y} \in Y$ に対して、不等式

$$\vec{x}^{*'} A \vec{y}^* \geq \vec{x}' A \vec{y}^* \quad (3)$$

$$\vec{x}^{*'} B \vec{y}^* \geq \vec{x}^{*'} B \vec{y} \quad (4)$$

を満たす組, $(\vec{x}^*, \vec{y}^*)$ として定義される。(3)はプレイヤー II の混合戦略 $\vec{y}^*$ を所与として, $\vec{x}^*$ が $\vec{x}' A \vec{y}^*$ の最大値を与えることを, (4)はプレイヤー I の混合戦略 $\vec{x}^*$ を所与として, $\vec{y}^*$ が $\vec{x}^{*'} B \vec{y}$ の最大値を与えることを意味する。すなわち, (3)において, $\vec{x}^*$ は次の線形計画問題⁴⁾の解である。この線形計画問題を問題 P_I とする。

$$\begin{aligned} (P_I) \quad & \text{Maximize } \vec{x}' A \vec{y}^* = (A \vec{y}^*)' \vec{x} \\ & \text{Subject to} \\ & \vec{p}_m \cdot \vec{x} = 1 \\ & \vec{x} \in R_{\geq 0}^m \end{aligned} \quad (5)$$

同様に, (4)において, $\vec{y}^*$ は次の線形計画問題の解である。そして, この線形計画問題を問題 P_{II} とする。

$$\begin{aligned} (P_{II}) \quad & \text{Maximize } \vec{x}^{*'} B \vec{y} = (B' \vec{x}^*)' \vec{y} \\ & \text{Subject to} \\ & \vec{p}_n \cdot \vec{y} = 1 \\ & \vec{y} \in R_{\geq 0}^n \end{aligned} \quad (6)$$

ここで, (5), (6)の $\vec{p}_k$ は, すべての要素が1の k 次ベクトル $\vec{p}_k = (1, \dots, 1)' \in R_k$ を意味する。

3 最適反応と Nash 均衡

この節では, 各プレイヤーの持つ純粋戦略が2 個の場合について, プレイヤーの最適反応を描く。Nash 均衡は, プレイヤーの最適反応の共通集合である。

\Mathematica <http://www.jip.co.jp/si/mathematica/price.html>

Maple <http://www.cybernet.co.jp/maple/price/index.shtml>

これでは, 追実験を行うには学問以外の問題点による敷居が高すぎる。

3) 上付記号の'は転置を意味する。

4) この定式化は von Stengel [4] による。

問題 (P_I) の解を $\vec{x}^*$ とする。 $\vec{x}^*$ はプレイヤー II の戦略 $\vec{y}^*$ を所与とした場合のプレイヤー I の最適戦略である。所与とする $\vec{y}^*$ に対して、 $\vec{x}^*$ は一義的に決まるとは限らない。そこでこれを集合

$$\phi(\vec{y}^*) = \{\vec{x}^* \mid (A\vec{y}^*)'\vec{x}^* = \max_{\vec{x} \in X} (A\vec{y}^*)'\vec{x}\} \quad (7)$$

で表す⁵⁾。 $\vec{y}^*$ と $\phi(\vec{y}^*)$ に属する $\vec{x}^*$ からつくられる組 $(\vec{x}^*, \vec{y}^*)$ の集合はプレイヤー I の最適反応と呼ばれる。これを D_I とする。

$$D_I = \{(\vec{x}^*, \vec{y}^*) \mid \vec{x}^* \in \phi(\vec{y}^*)\} \quad (8)$$

同様に、問題 (P_{II}) の解を $\vec{y}^*$ とする。 $\vec{y}^*$ はプレイヤー I の戦略 $\vec{x}^*$ を所与とした場合のプレイヤー II の最適戦略である。所与とする $\vec{x}^*$ に対して、 $\vec{y}^*$ は一義的に決まるとは限らない。そこで、これを集合

$$\phi(\vec{x}^*) = \{\vec{y}^* \mid (B'\vec{x}^*)'\vec{y}^* = \max_{\vec{y} \in Y} (B'\vec{x}^*)'\vec{y}\} \quad (9)$$

とする。 $\vec{x}^*$ と $\phi(\vec{x}^*)$ に属する $\vec{y}^*$ からつくられる組 $(\vec{x}^*, \vec{y}^*)$ の集合はプレイヤー II の最適反応と呼ばれる。これを D_{II} とする。このとき、Nash 均衡は D_I と D_{II} の共通集合として与えられる。

$$D_I \cap D_{II} = \{(\vec{x}^*, \vec{y}^*) \mid \vec{x}^* \in \phi(\vec{y}^*), \vec{y}^* \in \phi(\vec{x}^*)\} \quad (10)$$

$m = 2, n = 2$ のとき、いくつかの利得行列を仮定して最適反応⁶⁾をシミュレーションによって図示した。シミュレーションのためのプログラム、およびその実行の様子は付録に掲載している。

例えばプレイヤー I の利得行列 A を $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ で与え、プレイヤー II の利得行列 B を $B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ で与える。このとき Nash ゲームの最適反応を図示すると次のようになる。

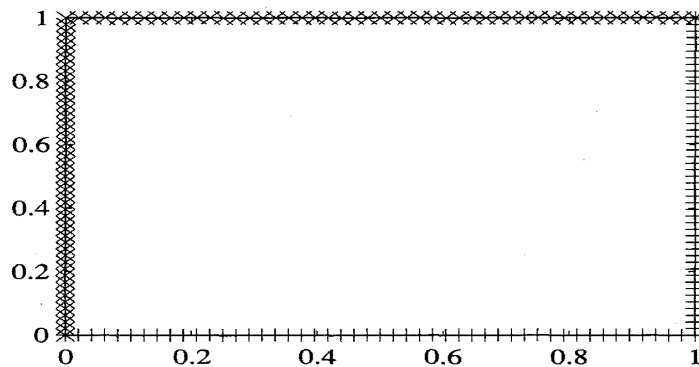


図 1：最適反応の例 1

このグラフから、Nash 均衡は 2 点存在することが予想される。

5) (7) の $\max_{\vec{x} \in X}$ は $\vec{x}$ に関して最大化を行うことを意味する。

6) 利得行列の要素と最適反応の関係は、白井 [6], 265 ページに示されている。

次に例えばプレイヤー I の利得行列 A を $A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$ で与え、プレイヤー II の利得行列 B を $B = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ で与える⁷⁾。このとき Nash ゲームの最適反応を図示すると次のようになる。

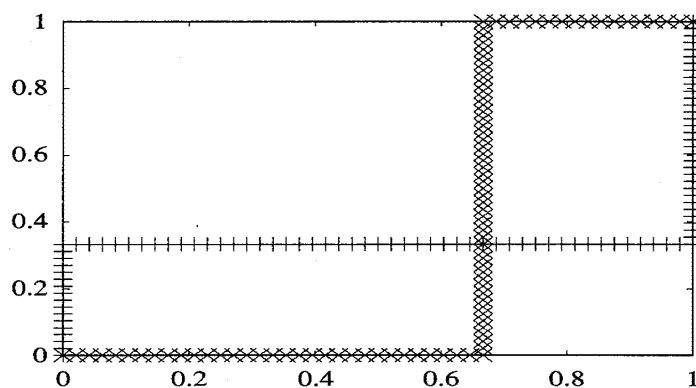


図 2：最適反応の例 2

このグラフから、Nash 均衡は 3 点存在することが予想される。

最後に例えばプレイヤー I の利得行列 A を $A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ で与え、プレイヤー II の利得行列 B を $B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ で与える⁸⁾。このとき Nash ゲームの最適反応を図示すると次のようになる。

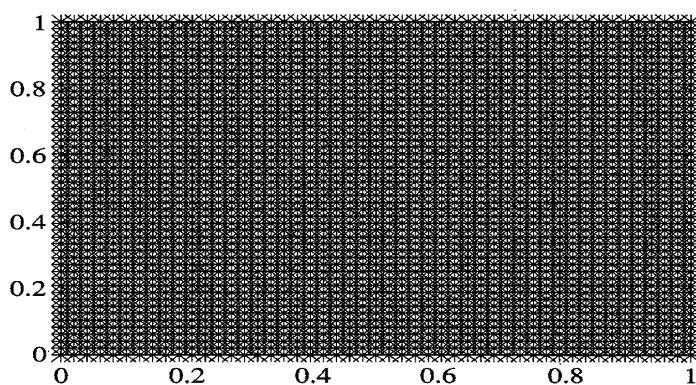


図 3：最適反応の例 3

このグラフから、Nash 均衡は $[0, 1] \times [0, 1]$ 全体に存在することが予想される。

4 線形相補方程式

前節では、プレイヤーの持つ純粋戦略が 2 個の場合について、シミュレーションによって最適反応をみた。この節では、一般の $m \times n$ 型の利得行列が与えられた場合の Nash 均衡を求めることを試みる。

7) これは「男女の闘い」と呼ばれる有名なゲームの例である。

8) 利得行列から明らかなように、戦略に関係なく利得が一定である。現実問題であれば、戦略を選択する気が無くなるゲームではある。

いま, 問題 P_I と問題 P_{II} の双対問題を考える。問題 P_I の双対問題を問題 Q_I とし, 双対変数を $u \in R$ とする。そして, 問題 P_{II} の双対問題を問題 Q_{II} とし, 双対変数を $v \in R$ とする。問題 Q_I と問題 Q_{II} はそれぞれ,

$$\begin{aligned} (Q_I) \quad & \text{Minimize } u \\ & \text{Subject to} \\ & u\vec{p}_m \geq A\vec{y}^* \\ & u \in R \end{aligned}$$

$$\begin{aligned} (Q_{II}) \quad & \text{Minimize } v \\ & \text{Subject to} \\ & v\vec{p}_n \geq B'\vec{x} \\ & v \in R \end{aligned}$$

である。

問題 P_I の最適解を $\vec{x}^*$ とし, その双対問題 Q_I の最適解を u^* とする。そのとき, 双対定理⁹⁾より, $u^* = (\vec{x}^*) A\vec{y}^*$ を得る。この等式は, (5) より, $u^*\vec{p}_m \cdot \vec{x}^* = u^*$ となるので, $\vec{x}^* \cdot (u^*\vec{p}_m - A\vec{y}^*) = 0$ と表すことができる。これは相補性条件と呼ばれる。同様にして, 問題 P_{II} の最適解を $\vec{y}^*$ とし, その双対問題 Q_{II} の最適解を v^* とする。そのとき, 双対定理より, $\vec{y}^* \cdot (v^*\vec{p}_n - B'\vec{x}^*) = 0$ を得る。

以上をまとめると, 行列ゲーム (A, B) は適当な u, v に対して, 次の式が成立することが, Nash 均衡 $\{(\vec{x}, \vec{y})\}$ を持つための必要十分条件である。

$$u\vec{p}_m - A\vec{y} \geq 0 \quad (11)$$

$$v\vec{p}_n - B'\vec{x} \geq 0 \quad (12)$$

$$\vec{p}_m \cdot \vec{x} = 1 \quad (13)$$

$$\vec{p}_n \cdot \vec{y} = 1 \quad (14)$$

$$\vec{x} \cdot (u\vec{p}_m - A\vec{y}) = 0 \quad (15)$$

$$\vec{y} \cdot (v\vec{p}_n - B'\vec{x}) = 0 \quad (16)$$

$$\vec{x} \in R_{\geq 0}^m, \vec{y} \in R_{\geq 0}^n \quad (17)$$

ここで, 各プレイヤーの利得行列の要素は非負であるものと仮定する。そして, その行列において列ベクトルまたは行ベクトルが $\vec{0}$ であるようなものは存在しないと仮定すると, (11)~(17) は次のようになる。

$$\frac{1}{v} \vec{x} \cdot (u\vec{p}_m - A\vec{y}) = 0 \quad (18)$$

$$\frac{1}{u} \vec{y} \cdot (v\vec{p}_n - B'\vec{x}) = 0 \quad (19)$$

$$\frac{1}{u} (A\vec{y} + (u\vec{p}_m - A\vec{y})) = \vec{p}_m \quad (20)$$

9) ここで言う双対定理は, 問題 P_I と問題 Q_I の一方が最適解をもてば, 他方も最適解をもち, 問題 P_I の目的関数の値と, 問題 Q_I の目的関数の値は一致することを意味する。

$$\frac{1}{v} \left(B' \vec{x} + (v \vec{p}_n - B' \vec{x}) \right) = \vec{p}_n \quad (21)$$

$$u \vec{p}_m - A \vec{y} \in R_{\geq 0}^m, \vec{x} \in R_{\geq 0}^m, v \vec{p}_n - B' \vec{x} \in R_{\geq 0}^n, \vec{y} \in R_{\geq 0}^n \quad (22) \quad (23)$$

$$v = \frac{v}{\vec{p}_m \cdot \vec{x}}, u = \frac{u}{\vec{p}_n \cdot \vec{y}} \quad (24)$$

さらに, (18), (19), (20), (21) は, 次のように表すことができる。

$$M \vec{z} + \vec{w} = \vec{p}_{m+n} \quad (25)$$

$$\vec{z} \cdot \vec{w} = 0 \quad (26)$$

$$\vec{z}, \vec{w} \in R_{\geq 0}^{m+n}$$

ここに,

$$M = \begin{pmatrix} 0_m & A \\ B' & 0_n \end{pmatrix}, \vec{z} = \begin{pmatrix} \vec{x}/v \\ \vec{y}/u \end{pmatrix}, \vec{w} = \begin{pmatrix} \vec{p}_m - A(\vec{y}/u) \\ \vec{p}_n - B'(\vec{x}/v) \end{pmatrix}$$

と置いている。この問題は線形相補計画問題と呼ばれている。このとき, $\vec{z}, \vec{w}$ の成分について (z_i, w_i) , $(i = 1, \dots, m+n)$ は相補対と呼ばれる。

5 Maxima による線型相補方程式の解

線形相補方程式 (25), (26) において, 方程式の数は $2(m+n)$ 式, 変数は $2(m+n)$ 個であり, 変数と方程式の数は一致する。したがって, もしこの方程式を直接解いて非負条件を満たす解を得ることができれば, それらの解に対応する Nash 均衡を求めることができる。線形相補方程式を解いて Nash 均衡を求める手順は次のとおりである。

- 利得行列の要素を正数にした後, 上述の行列 M を作る。
- 線形相補方程式のすべての解を求める¹⁰⁾。
- 解のなかで不定解を除いて, 非負条件をみたすものを抽出して, そこから Nash 均衡を計算する。

線型相補方程式を利用して Nash 均衡解を求めるための Maxima プログラムは, とい nash-equilibria という関数名¹¹⁾でパッケージ化した。この関数は 2 プレイヤー I, II の利得行列 A, B を引数として与えると, その Nash 均衡解の中で “一意的” なものを返すプログラムである。ここで言う “一意的な解” とは, 非負条件をみたす不定解でない均衡解を意味する。イメージで言えば, 線分などではなく離散的な点集合によって示される解をいう。

この nash-equilibria の構造は, 次の 3 つのパートから成る。

- 2 つの (m, n) 利得行列から線型相補方程式を立てる。
- 2^{m+n} 種類の $m+n$ 元連立 1 次方程式を解く。
- 解集合の中で, 一意的な解を選択する。

以下では, それぞれのパートについて, 計算手順を詳しく述べる。

10) 線形相補方程式の非負解を求める古典的な方法として Lemke-Howson 法 [2] が有名である。

11) function の漢語訳として, Matteo Ricci (利瑪竇) らの尽力によって “函數”, 現代字の “函数” が与えられていた。しかし, 日本では戦後直後に起こった漢字全廃運動の悪影響で “函” が「難解な文字である」という理由で当用漢字から外された。その後, 1950 年代に代用として “関数” を用いることを文部省が許可したので, その当て字が学校教育によって定着し, 漢字文化圏で function を関数と書く国は日本だけである。本稿では, 数学の function を “関数” で, コンピュータプログラムの function を “函数” として書き分けることにする。

5.1 線型相補方程式を立てる

線型相補方程式を立てるときに、

- (1) 利得行列のすべての成分に同じ正数を加えて非負化する。
- (2) 相補性に注目して、方程式を線型化する。

という2つのアイデアを利用した。

(1)は、利得行列である行ベクトルの成分がすべて0であれば、方程式として係数行列のrankが下がるために、そこから得られる方程式情報が失われる。それを避けるためのアイデアであるが、これは次の命題に基づいている。

命題 5.1 利得行列のすべての成分に同じ数を加えても、そのNash均衡解は変化しない。

証明 5.1

プレイヤー I の期待利得は、 $\vec{x}'A\vec{y}^*$ であり、Nash 均衡解の最大化条件から、

$$(\vec{x}^*)'A\vec{y}^* - \vec{x}'A\vec{y}^* \geq 0 \quad (27)$$

をみtas。ここで $m \times n$ 行列

$$P_{mn} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

と置くと、 k を任意のスカラーとして、

$$\begin{aligned} & (\vec{x}^*)'(A + kP_{mn})\vec{y}^* - \vec{x}'(A + kP_{mn})\vec{y}^* \\ &= (\vec{x}^*)'A\vec{y}^* - \vec{x}'A\vec{y}^* + k(\vec{x}^*)'P_{mn}\vec{y}^* - k\vec{x}'P_{mn}\vec{y}^* \quad (\text{線型性より}) \\ &= (\vec{x}^*)'A\vec{y}^* - \vec{x}'A\vec{y}^* + k(\vec{x}^*)'\vec{p}_m - k\vec{x}'\vec{p}_m \quad (\sum_{i=1}^n y_i = 1 \text{ より}) \\ &= (\vec{x}^*)'A\vec{y}^* - \vec{x}'A\vec{y}^* + k - k \quad (\sum_{i=1}^m x_i = 1 \text{ より}) \\ &= (\vec{x}^*)'A\vec{y}^* - \vec{x}'A\vec{y}^* \end{aligned}$$

となり、式(27)と同値の式となる。

同様にして、プレイヤー II の期待利得は、 $\vec{x}^*\vec{y}'B$ であり、Nash 均衡解の最大化条件から、

$$(\vec{x}^*)'\vec{y}'B - \vec{x}^*\vec{y}'B \geq 0 \quad (28)$$

をみtas。すると k を任意のスカラーとして、

$$\begin{aligned} & (\vec{x}^*)'(B + kP_{mn})\vec{y}^* - \vec{x}^*(B + kP_{mn})\vec{y}^* \\ &= (\vec{x}^*)'\vec{y}'B - \vec{x}^*\vec{y}'B + k(\vec{x}^*)'P_{mn}\vec{y}^* - k\vec{x}^*P_{mn}\vec{y}^* \quad (\text{線型性より}) \\ &= (\vec{x}^*)'\vec{y}'B - \vec{x}^*\vec{y}'B + k(\vec{x}^*)'\vec{p}_m - k\vec{x}^*\vec{p}_m \quad (\sum_{i=1}^n y_i = 1 \text{ より}) \end{aligned}$$

$$\begin{aligned}
&= (\vec{x}^*)' \vec{B} \vec{y}^* - \vec{x}^{*'} \vec{B} \vec{y} + k - k \quad \left(\sum_{i=1}^m x_i = 1 \text{ より} \right) \\
&= (\vec{x}^*)' \vec{B} \vec{y}^* - \vec{x}^{*'} \vec{B} \vec{y}
\end{aligned}$$

となり、式(28)と同値の式となる。よって、同じ Nash 均衡解を得る。

(2)は、計算の高速化に関わるアイデアである。2 プレイヤーによる (m, n) 戦略 Nash ゲームの線型相補方程式全体では非線型方程式であるが、線型方程式の部分 $M\vec{z} + \vec{w} = \vec{p}_{m+n}$ と相補性の部分 $z_i w_i = 0, (1 \leq i \leq m+n)$ に分けることができる。後半の相補性とは、同じ i について z_i または w_i が 0 であるということを意味する。どちらかが 0 になる組み合わせは全部で 2^{m+n} 組存在するが、その 1 つを採用してそれを線型方程式の部分に代入すれば、 $m+n$ 元連立 1 次方程式を解くことになる。そこで、どの変数を 0 にするか組み合わせを、0 から $2^{m+n} - 1$ までの $m+n$ 桁 2 進数¹²⁾ のリストから指定することで、 2^{m+n} 種類の $m+n$ 元連立 1 次方程式を立てた。

5.2 連立 1 次方程式を解く

コンピュータによる代数計算において、非線形方程式を含め、代数的に解けるものの多くを解く関数¹³⁾は、かなり時間が掛かる。一方、線形方程式に限るのであれば、大変早く解く方法¹⁴⁾がよく研究されている。この研究では、この線形方程式専用の解法を十分に駆使した。

計算環境は、CPU が Celeron 1.06 GHz, メモリが 512MB, OS が Microsoft Windows 2000 で Maxima 5.9.3 を用いた。そこで 2 つの (n, n) 利得行列として、各成分を 0 から 9 までの乱数で与え、その一意的な Nash 均衡解を計算した。そして、その計算時間 $t(n)$ は、 $n \leq 6$ において、次の実験結果を得た。

この表から、 $t(n) = 10^{0.828n-2.79}$ と線形回帰式を得る。先行研究である Dickhaut と Kaplan の論文 [1] では、2 つの (n, n) 利得行列を与えたときの計算時間 $t(n)$ を、 $t(n) = 10^{0.97n-2.0}$ としている。この論文は 1993 年に出版されたものであり、論文にコンピュータの性能が明記されていないので単純には比較できない。だが、コンピュータの性能は、主に指数部の定数項に影響を与えると考えてよい。これに対して計算アルゴリズムは、主に指数部の n の係数に影響を与える。これを 84% ほどに小さくできたことは、意義があると考えている。ちなみに、線形相補方程式をそのまま一般型の解法に引き渡した場合の線形回帰式は、 $t(n) = 10^{0.957n-2.67}$ となり、Dickhaut と Kaplan の研究結果のコンピュータ性能の影響項が良くなっただけのものに近くなる。

なお、今回のコンピュータ環境では $n \geq 7$ となると、メモリ不足のために仮想メモリとしてのハードディスクへのアクセスが頻繁に行われたため、計算速度が著しく低下した。例えば、線形回帰式では $n=7$ ならば 1014 秒ほどであると推測されるが、実際には 2230 秒前後となった。このハードウェアでは、これ以上メモリの増設ができない機種なので、 $n \leq 6$ までとした。

5.3 一意的な解を選択する

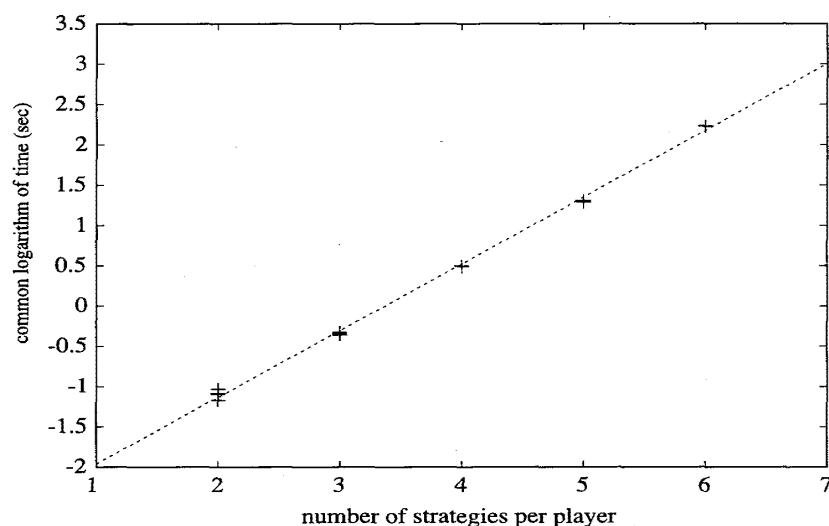
求める解は、各戦略を選択する確率の比であるので、 $\vec{z}, \vec{w} \in R_{\geq 0}^{m+n}$ を満たす。よって、まずは線型方程式の解の中から、非負解のみを選択する。非負条件は、絶対値をとっても変化しないこと

12) 桁数が不足するときは、上位桁にダミーとして 0 を補うことで $m+n$ 桁にする。例えば、45 を 8 桁の 2 進数にするなら、00101101 (2) とする。

13) Maxima では solve (方程式のリスト, 変数のリスト); で解くことができる。

14) Maxima では linsolve (方程式のリスト, 変数のリスト); で解くことができる。

n	1 回目	2 回目	3 回目	4 回目	5 回目	平均(秒)
2	0.07	0.08	0.09	0.09	0.08	0.082
3	0.47	0.44	0.45	0.46	0.44	0.452
4	3.09	3.17	3.16	3.16	3.11	3.138
5	19.99	19.45	19.78	20.26	19.40	19.776
6	171.42	171.35	169.23	170.75	170.14	170.578



と同値であるから、それによって判断した。Maxima に線形方程式を解かせると、不能解の場合は空集合を、不定解の場合は適当な媒介変数による表記を返す。理論的には、不定解の中にも媒介変数の領域次第で Nash 均衡解になるものが存在するが、本稿では、一意的な解のみを求めることにしたので、不定解も同時に省くことにした。これは、媒介変数が未定である以上、絶対値をとって変化しないとは限らないので、負解を省くときと同じ函数で省くことができる。

6 おわりに

本稿では、行列ゲームの Nash 均衡を図示するためのプログラムと理論的に求めるためのプログラムを構築した。ただし、より多くの Nash 均衡を求めるためには、線形相補方程式の不定解の中から非負解を取り出す統一的なアルゴリズムを構築することが必要になる。また、線形でも連立方程式の個数が指数的に増えるので、ネットワークを利用してグリッドコンピューティングを実装する必要がある。これらの点は今後の課題である。

Maxima は GNU General Public Licenses の下で利用可能な無課金のソフトウェアである点が特徴的である。誰もが自由にインターネットを通じてダウンロード¹⁵⁾することができるのである。この点にこのソフトウェアを利用する価値がある。

本稿は、社会科学分野の研究や教育への、Maxima の応用を意図した研究計画の端緒である。今後、この分野の研究や教育において、この数式処理ソフトを積極的に活用したいと考えている。

A Maxima のインストール

これから Maxima の標準的なインストール方法を説明する。以下は、2006年6月現在の情報で、最新の Maxima のバージョンは 5.9.3 である。2006年9月にバージョンが 5.10.0 にバージョンアップするので、最新情報は <http://www.biz.ryukoku.ac.jp/~yamasita/> から得られるようにする。

A.1 Linux の場合

```
http://sourceforge.net/project/showfiles.php?group-id=4933
```

に rpm 版バイナリがあるので、それをダウンロードする。なお一緒に、CLISP 5.9.3 と XMaxima 5.9.3 もダウンロードすることをお勧めする。さらに、

```
ftp://ftp.pbone.net/mirror/www.arklinux.org/1.0-0.alpha12.2/4586/
```

から、gnuplot-4.0.0-1ark.i586.rpm をダウンロードする。後は、通常のパッケージと同じように、root になってインストールする。つまり

```
# rpm-ivh maxima-5.9.3-1.fedora1.i386.rpm
# rpm-ivh maxima-exec-clisp-5.9.3-1.fedora1.i386.rpm
# rpm-ivh maxima-xmaxima-5.9.3-1.fedora1.i386.rpm
# rpm-ivh gnuplot-4.0.0-1ark.i586.rpm
```

で、インストールできる。これで Maxima を利用できるが、`~/.bashrc` の末尾にでも、

```
alias xmaxima="xmaxima>/dev/null"
```

を加えておくと便利であろう。

A.2 MacOS X の場合

以下の URL の記事によると、apt-get でインストールできるようである。

```
http://macwiki.sourceforge.jp/cgi-bin/wiki.cgi?Imaxima
```

なお、筆者の手に MaxOS X が動くコンピュータが無いので、インストールの追実験は行っていない。

A.3 Microsoft Windows の場合

Microsoft Windows 用の Maxima は、英語版 gnuplot まで含めたインストーラがある。

```
http://sourceforge.net/project/showfiles.php?group id=4933
```

から Windows 用インストーラ maxima-5.9.3.exe をダウンロードする。

コンピュータ本体にインストールするならば、ライセンス確認のときに I accept the agreement のラジオボタンを選び直すこと以外は、単純に Next をクリックし続けるだけでよい。

B Maxima による Nash 均衡解の計算プログラム

本研究では、以下のプログラムを使用した¹⁶⁾。

15) この点は付録 A 「Maxima のインストール」を参照のこと。

16) なお、このプログラムは2006年6月現在、

```
http://www.biz.ryukoku.ac.jp/~yamasita/
```

からダウンロードできるようにしてある。これを適当なディレクトリに保存すれば、load 函数にそのフルノ

```

list__transpose(lst):=makelist(
    makelist(lst[j][i], j, 1, length(lst)),
    i, 1, length(first(lst))
);
reaction__function(x, y, mtx):=[x, 1-x].mtx.[y, 1-y];
reaction__mesh(mtx, stp):=makelist(
    makelist([reaction__function((i-1)/stp, (j-1)/stp, mtx), i, j], i, 1, stp+1),
    j, 1, stp+1
);
maximal__elements(bmx):=block([i, j, rvm, mxl, lst:[]],
    for i:1 thru length(bmx) do(
        rvm:last(sort(first(list__transpose(bmx[i])))),
        mxl:apply(
            "+",
            map(
                entier,
                first(list__transpose(bmx[i]))/rvm
            )
        ),
        lst:append(lst, reverse(rest(reverse(sort(bmx[i])), mxl-length(bmx[i]))))
    ),
    return(lst)
);
plot__points__list(mxe, stp):=float(
    (sort(list__transpose(rest(list__transpose(mxe))))-1)/stp
);
nash__simulation(a, b, stp):=block([i, m],
    m:apply(min, flatten([a[1], a[2], b[1], b[2]])),
    plot2d(
        [
            [discrete, plot__points__list(
                maximal__elements(
                    reaction__mesh(a-m+1, stp)
                ),
                stp)
            ],
            [discrete, map(
                lambda([i], reverse(i)),
                plot__points__list(

```

パスを与えると利用することができる。後述の計算例では、Cドライブにtempというディレクトリを作成して、そこにnash.macを置いて利用した様子を掲載した。Microsoft Windows 追実験するのであれば、フルパス名として日本語文字を含まないようにしなければMaximaで入力できない。つまり、マイドキュメントに置くと利用できないということになる。

```

        maximal__elements(
            reaction__mesh(transpose(b-m+1), stp)
        ),
        stp))
    ]
],
[gnuplot__curve__styles, ["with points"]],
[gnuplot__preamble, ["unset key"]]
)
);
block__matrix(mtx):=apply(
    append,
    makelist(
        transpose(
            apply(
                append,
                matrixmap(transpose, mtx)[i]
            )
        ),
        i, 1, length(mtx)
    )
);
positive__select(slv):=block([i, lst],
    lst:listify(setify(
        map(
            lambda([i], if i=abs(i) then i else []),
            slv
        )
    )),
    if first(lst) = [] then return(rest(lst)) else return(lst)
);
digit2binarylist(x, n):=block([i:0, lst:[], y:x, z],
    while i<n do(
        z:entier(y/2),
        lst:cons(y-2*z, lst),
        y:z, i:i+1
    ),
    return(lst));
nash__equilibria(a, b):=block(
    [me, i, j, m:length(a), n:length(a[1]), mx, w, z, ww, zz, z0, fg, eq, sl, ps, ms, ne:[],
    solve__inconsistent__error:false,
    linsolvewarn:false,
    me:apply(min, flatten([makelist(a[i], i, 1, m), makelist(b[i], i, 1, m)])),
    mx:block__matrix(

```

```

matrix([zeromatrix(m, m), a-me+1], [transpose(b-me+1), zeromatrix(n, n)])
),
ww:makelist(w[i], i, 1, m+n),
zz:makelist(z[i], i, 1, m+n),
z0:makelist(z[i]=0, i, 1, m+n),
fg:makelist(digit2binarylist(i-1, m+n), i, 1, 2^(m+n)),
eq:map(
  lambda([i], if first(i)=first([0=1]) then rest(i) else i),
  makelist(
    listify(setify(makelist(
      (mx.(zz*fg[j])+ident(m+n).(ww*fg[2^(m+n)-j+1]))[i][1]=1,
      i, 1, m+n
    ))),
    j, 1, 2^(m+n)
  )
),
sl:map(
  lambda([i], if length(i)=m+n then append(z0, i) else i),
  makelist(
    append(
      sort(linsolve(eq[i], listofvars(eq[i]))),
      outermap("=", j, fg[i])
    ),
    i, 1, 2^(m+n)
  )
),
ps:map(
  lambda([i], [rest(i, -n), rest(i, m)]),
  listify(setify(map(
    lambda([i], rest(i, -m-n)*rest(i, m+n)),
    positive__select(matrixmap(rhs, sl))
  )))
), ms:matrixmap(
  lambda([i], apply("+", i)),
  apply(matrix, ps)
),
for i:1 thru length(ps) do(
  if apply("*", ms[i])>0 then ne:append(ne, [ps[i]*(1/ms[i])]),
  solve__inconsistent__error:true,
  linsolvewarn:true,
  return(ne)
);

```

C Nash 均衡解の計算例

実際に Maxima で行った数値実験の計算例を示す¹⁷⁾。

```
(%i1) load("C:/temp/nash.mac");
(%o1) C:/temp/nash.mac
(%i2) a:matrix([1, 1], [0, 1]);
      [ 1  1 ]
(%o2) [      ]
      [ 0  1 ]

(%i3) b:matrix([1, 0], [1, 1]);
      [ 1  0 ]
(%o3) [      ]
      [ 1  1 ]

(%i4) nash__simulation(a, b, 50);
(%o4)
(%i5) a:matrix([2, 0], [0, 1]);
      [ 2  0 ]
(%o5) [      ]
      [ 0  1 ]

(%i6) b:matrix([1, 0], [0, 2]);
      [ 1  0 ]
(%o6) [      ]
      [ 0  2 ]

(%i7) nash__simulation(a, b, 60);
(%o7)
(%i8) a:matrix([1, 1], [1, 1]);
      [ 1  1 ]
(%o8) [      ]
      [ 1  1 ]

(%i9) b:matrix([1, 1], [1, 1]);
      [ 1  1 ]
(%o9) [      ]
      [ 1  1 ]

(%i10) nash__simulation(a, b, 50);
(%o10)
(%i11) showtime:ture;
Evaluation took 0.00 seconds (0.00 elapsed)
(%o11) ture
(%i12) a:matrix([-1, -1, 1, -1], [-1, -1, 0, 0], [0, 0, 1, -1], [0, 0, 0, 0]);
Evaluation took 0.00 seconds (0.00 elapsed)
      [ -1 -1  1 -1 ]
```

17) ただし, nash__simulation 関数は, グラフを表示するだけのものであり, そのグラフは既出しているので再掲は控える。

```

[
[ -1 -1 0 0 ]
(%o12) [
[ 0 0 1 -1 ]
[
[ 0 0 0 0 ]
(%i13) b:matrix([-1, -1, 1, -1], [-1, -1, 2, 2], [2, 2, 1, -1], [2, 2, 2, 2]);
Evaluation took 0.00 seconds(0.00 elapsed)
[ -1 -1 1 -1 ]
[
[ -1 -1 2 2 ]
(%o13) [
[ 2 2 1 -1 ]
[
[ 2 2 2 2 ]
(%i14) nash__equilibria(a, b);
Evaluation took 3.12 seconds (3.12 elapsed)
(%o14) [[[0, 0, 0, 1], [0, 0, 0, 1]], [[0, 0, 0, 1], [0, 0,  $\frac{1}{2}$ ,  $\frac{1}{2}$ ]],
[[0, 0, 0, 1], [0, 1, 0, 0]], [[0, 0, 0, 1], [1, 0, 0, 0]],
[[0, 0, 1, 0], [0, 1, 0, 0]], [[0, 0, 1, 0], [1, 0, 0, 0]],
[[0, 1, 0, 0], [0, 0, 0, 1]], [[0, 1, 0, 0], [0, 0,  $\frac{1}{2}$ ,  $\frac{1}{2}$ ]],
[[ $\frac{1}{3}$ , 0,  $\frac{2}{3}$ , 0], [0, 0, 1, 0]], [[1, 0, 0, 0], [0, 0, 1, 0]]]
(%i15) mtx(n):=apply(matrix, makelist(makelist(random(10), i, 1, n), j, 1, n));
Evaluation took 0.00 seconds (0.00 elapsed)
(%o15) mtx(n):=apply(matrix, makelist(makelist(random(10), i, 1, n), j, 1, n))
(%i16) a:mtx(6);
Evaluation took 0.00 seconds (0.00 elapsed)
[ 6 6 3 2 4 7 ]
[
[ 5 5 5 3 4 5 ]
[
[ 4 5 1 2 4 7 ]
(%o16) [
[ 5 6 0 4 2 7 ]
[
[ 4 3 8 2 7 8 ]
[
[ 8 2 9 4 0 1 ]
(%i17) b:mtx(6);
Evaluation took 0.00 seconds(0.00 elapsed)
[ 4 2 9 1 1 3 ]
[

```

```
(%o17)
[ 8 3 8 7 5 8 ]
[
[ 8 1 7 8 5 8 ]
[
[ 8 6 1 7 7 3 ]
[
[ 9 8 3 9 1 3 ]
[
[ 3 7 9 5 4 4 ]

(% i 18) nash__equilibria(a,b);
Evaluation took 165.68 seconds (165.68 elapsed)
(%o18) [[[0, 0,  $\frac{1}{2}$ ,  $\frac{1}{2}$ , 0, 0], [0, 0, 0, 0, 0, 1]],
[[0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 1]], [[0,  $\frac{1}{2}$ , 0, 0,  $\frac{1}{2}$ , 0], [ $\frac{3}{7}$ , 0, 0, 0,  $\frac{4}{7}$ , 0]],
[[0,  $\frac{7}{8}$ , 0, 0, 0,  $\frac{1}{8}$ ], [0, 0, 0, 0, 1, 0]], [[0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0]],
[[ $\frac{1}{4}$ , 0,  $\frac{3}{4}$ , 0, 0, 0], [0, 0, 0, 0, 0, 1]]]
```

参考文献

- [1] Dickhaut, J. and T. Kaplan, T., "A program for finding Nash equilibria", *Economic and Financial Modeling with Mathematica*, Springer-Verlag, (1993).
- [2] Lemke, C. E. and Howson, J. T., Jr., "Equilibrium points in bimatrix games", *Journal of the Society for Industrial and Applied Mathematics* 12, (1964).
- [3] McKelvey, R. D. and A. McLennan, "Computation of Equilibria in Finite Games", *Handbook of Computational Economics*, Vol.1, Elsevier, (1996).
- [4] von Stengel, B. "Computing Equilibria for Two-Person Games",
<http://www.maths.lse.ac.uk/Personal/stengel/TEXTE/nashsurvey.pdf>
- [5] Schelter, W. et al., Maxima,
<http://maxima.sourceforge.net>, software.
- [6] 臼井功, 『経営数学概論』, 有斐閣, 1985年

(受理 2006年 6 月30日)