包絡分析法に基づく十種競技得点計算の検証

著者	倉田 俊彦
出版者	法政大学経営学会
雑誌名	経営志林
巻	46
号	4
ページ	11-27
発行年	2010-01-30
URL	http://doi.org/10.15002/00008423

[論　文]

包絡分析法に基づく十種競技得点計算の検証

倉　田　俊　彦[1]

1.　導入

十種競技得点計算の起源は1912年であり，当時は，一般人の記録とオリンピック最高記録との間を1000等分して，記録の向上に応じて線形に得点が増加する採点表が採用されていた。その後,「それぞれの種目における記録と得点の妥当性」や「異なる種目間の得点の公平性」などを確保する必要性から，1934年，1952年，1962年，1985年に採点表の改訂が行われて現在に至っている。

得点計算に不具合が生じる要因としては，一つに，種目間で記録向上の速度に差があるため，時代の流れと共にそれらの相対的な重要度に不均衡がもたらされることが挙げられる。実際に，競技技術や道具の発達などよって特定の種目の記録が著しく改善されることがあり，結果として技術革新の起こった種目の重要度が大きくなる現象があった。

このように，競技の特性に依存した不可避な要因とは対照的に，もう一つの要因として,（公平性に関する考察が進むことによって）得点計算式自体の不備が明らかになることが挙げられる。例えば，記録の分布からの類推によって，各種目に対して「記録の伸びに応じて累進的に得点を増加させることが自然である」と考えられるようになった。（例えば，[3]においても記録の向上に応じて順位は指数関数的に上昇する現象が観察されていて，こうした事実からもその妥当性が窺える。）これに対して，1962年の採点表は,（トラック種目に対しては累進的に増加する得点規則を持っていたが）フィールド種目に対して逆累進的に増加する得点規則を与えていたために批判を受ける結果となり，1985年の改訂で累進的に増加する得点規則に変更された経緯がある。

実際に，現在も使用されている1985年採点表においては，トラック競技は秒を単位として，フィールド競技はメートルを単位として記録 r を測定し，以下の指数関数に代入することによってそれぞれの種目の得点が計算されている。

100m	$25.4347 \cdot (18 - r)^{1.81}$
走幅跳	$0.14354 \cdot (100 \cdot r - 220)^{1.4}$
砲丸投	$51.39 \cdot (r - 1.5)^{1.05}$
走高跳	$0.8465 \cdot (r - 75)^{1.42}$
400m	$1.53775 \cdot (82 - r)^{1.81}$
110mH	$5.74352 \cdot (28.5 - r)^{1.92}$
円盤投	$12.91 \cdot (r - 4)^{1.1}$
棒高跳	$0.2797 \cdot (100 \cdot r - 100)^{1.35}$
槍投	$10.14 \cdot (r - 7)^{1.08}$
1500m	$0.03768 \cdot (480 - r)^{1.85}$

これらの式の中で使用されている定数（例えば，100mにおける25.4347や1.81といった数値）は，上に述べた改訂の過程の中で慎重な考察を重ねて得られた数値となっている。その意味で，現在の採点表でもある程度の信頼度は確保されていると考えられるが，原理的に，これらの関数のみで絶対的な概念として種目間の公平性を確保することは不可能である。（実際に，定数のさじ加減に依存して選手間の優劣が決定されるわけであるが，種目間の重要性のバランスは時と共に変化していて，そのような変化にも対応する絶対的に正しい定数というものは存在するはずがない。）

このような背景の下で，今回の考察では，十種競技の得点計算に対して,全く別の視点から（種目間のバランスも考慮に入れて）競技者の優劣に関する絶対的な評価を与えるアプローチを与える。具体的には，事業体の経営効率を分析する方法の一つとして広く知られている包絡分析法と呼ばれる手法[4]を利用して競技者の効率値（成績）の測定を試みる。

包絡分析法の特徴は，競技者の分布形状の情報のみに依存して効率値が決定される点にあり，このことは，評価結果の算出過程に（従来の得点計算に現

[1]法政大学経営学部　102-8160 東京都千代田区富士見 2-17-1　kurata@i.hosei.ac.jp

れる定数のような）主観的に調整できる要素が全く含まれていないことを意味している。そして，こうした特徴から，包絡分析法に基づく評価結果と従来の採点表に基づく評価結果を比較することによって，従来の得点計算規則に関する検証も可能になると考えた。

なお，包絡分析法には，CCR モデルや BCC モデルなど幾つかの変種が存在し，考察対象の性質に応じてそれらを使い分ける必要がある。これについて，今回の分析対象では，明らかに規模の効率性は仮定する必要はなく BCC モデルを採用した。

2.　包絡分析法の BCC モデル

様々な事業体を抽象的に捉えると，何れも複数の入力（資源）から複数の出力（成果）を生み出す活動とみなすことができる。そこで，ある事業に対して s 種類の入力項目と t 種類の出力項目に注目して，それらの関係を考察する際は，個々の入出力の数量を各々数ベクトル $\boldsymbol{a} = (a_1 \cdots a_s)$ と $\boldsymbol{b} = (b_1 \cdots b_t)$ によって表し，入出力ベクトルの組 $\langle \boldsymbol{a}, \boldsymbol{b} \rangle \in \mathbb{R}^s \times \mathbb{R}^t$ として事業体の活動を表現することとする。また，一般的に，小さな入力の下で大きな出力を生み出す活動の方が効率的であると考えられるので，2 つの活動 $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ と $\langle \boldsymbol{c}, \boldsymbol{d} \rangle$ に対して，$\boldsymbol{a} \geq \boldsymbol{c}$ かつ $\boldsymbol{b} \leq \boldsymbol{d}$ の時に，$\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ よりも $\langle \boldsymbol{c}, \boldsymbol{d} \rangle$ の方が効率のよい活動であると定義する[2]。

こうした基本的な仮定の下で，包絡分析法の BCC モデルでは，既存の活動の凸結合（係数の和が 1 となる非負線形結合）として得られる活動，及びそれらよりも効率の悪い活動のみを「実際に存在し得る活動」と認める立場をとる。つまり，既存の活動の集合を $D = \{\langle \boldsymbol{a}_1, \boldsymbol{b}_1 \rangle, \ldots, \langle \boldsymbol{a}_n, \boldsymbol{b}_n \rangle\}$ とした時に，実在し得る活動の集合は

$$\Big\{ \langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \mathbb{R}^s \times \mathbb{R}^t \;\Big|\; \exists \lambda_1, \ldots, \lambda_n \in \mathbb{R}^+ \cup \{0\} \quad \sum_{i=1}^{n} \lambda_i = 1,\ \boldsymbol{x} \geq \sum_{i=1}^{n} \lambda_i \boldsymbol{a}_i,\ \boldsymbol{y} \leq \sum_{i=1}^{n} \lambda_i \boldsymbol{b}_i \Big\}$$

で与えられることになる。この集合を（D から生成される）生産可能集合と呼び $P(D)$ と表記する。また，生産可能集合 $P(D)$ の中で，特に，自分自身よりも効率のよい活動が存在しない（極大な効率を持つ）活動は「効率の悪い活動に対して最良の模範となり得る」という点で重要な意味を持っている。そこで，そのような極大な効率を持つ活動の集合

$$\{ \langle \boldsymbol{x}, \boldsymbol{y} \rangle \in P(D) \mid \forall \langle \boldsymbol{a}, \boldsymbol{b} \rangle \in P(D)\ \ (\boldsymbol{x} \ngeq \boldsymbol{a} \text{ または } \boldsymbol{y} \nleq \boldsymbol{b}) \}$$

を効率的フロンティアと呼び $F(D)$ と表記する。定義から明らかであるが，効率的フロンティアに属する活動は，必ず既存の活動の凸結合として表現することができる。

こうして定義される生産可能集合の範囲内で，各活動に対して，「入力ベクトルの各成分を一律に圧縮する」という限定された形式で効率化を図ることを想定する。入力ベクトルを圧縮すればするほど効率的な活動が得られることになり，活動 $\langle \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ の効率性は，生産可能集合 $P(D)$ に留まる範囲内で最も大きな効率化を実現する際の圧縮率の数値に反映されることとなる[3]。つまり，活動 $\langle \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ の効率性は

$$\min\{ \theta \in \mathbb{R} \mid \langle \theta \boldsymbol{a}_p, \boldsymbol{b}_p \rangle \in P(D) \}$$

として与えられ，この値を活動 $\langle \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ の D-効率値と呼び，θ_p^* と表記することにする。こうして定義された D-効率値 θ_p^* は，明らかに 0 以上 1 以下の実数となり，この値の小さな活動ほど効率が悪い（大きな効率改善が可能）と見なすことが出来る。効率的フロンティアに属する活動の D-効率値は必然的に 1 となるが，その逆は必ずしも成り立たない。実際に，D-効率値が 1 ということは，単に「入力を一律に圧縮するという限定された形式の下で効率化を行った場合は，それ以上の改善は見込めない」ことを意味していて，そのことが一般的な効率化の可能性を全て否定することにはならない。

活動 $\langle \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ を効率化して得られた結果 $\langle \theta_p^* \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ が効率的フロンティアに属している時は，これを $\langle \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ の模範活動と呼ぶ。各活動に対して，

[2] $\boldsymbol{a} \geq \boldsymbol{c}$ は「$\boldsymbol{a}$ と $\boldsymbol{c}$ を各成分毎に比較した時に，どの成分についても $\boldsymbol{a}$ の方が $\boldsymbol{c}$ よりも大きいもしくは等しい値を持っている」ことを意味する。

[3] 双対の選択肢として，出力ベクトルの拡大率を考えることもできる。今回の考察において，入力ベクトルの縮小率を採用したことに特別な意図はない。

その模範活動は「それ以上の効率化が期待できない最良の改善目標」の一つを示していると考えられる。

一方で，$\langle \theta_p^* \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ が効率的フロンティアに属していない場合は，効率的フロンティアの中に $\langle \theta_p^* \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ よりも効率のよい活動が存在するので，それらの集合の中から $\langle \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ の模範活動となる要素を何らかの指針に従って選択することとなる。その場合，$\langle \theta_p^* \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ と選択された模範活動 $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ との間には，削減すべき入力の差 $\theta_p^* \boldsymbol{a}_p - \boldsymbol{a}$ と増大すべき出力の差 $\boldsymbol{b} - \boldsymbol{b}_p$ が存在していることとなり，これらはそれぞれ（入力の）余剰と（出力の）不足と呼ばれる[4]。つまり，活動 $\langle \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ を模範活動 $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ に改善するには，入力データを D-効率値の示す割合で圧縮してから，更に，入力を余剰分だけ削減し出力を不足分だけ増大する必要があることとなる。

一般的に効率的フロンティアに属する活動は既存の活動の凸結合として表現することが可能であったから，活動 $\langle \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ の模範活動は既存の活動の凸結合として

$$\Big\langle \sum_{i=1}^{n} \lambda_{p,i}^* \boldsymbol{a}_i,\ \sum_{i=1}^{n} \lambda_{p,i}^* \boldsymbol{b}_i \Big\rangle$$

の形で（一意に）表現できる。この凸結合は，既存の活動を組合せることによって $\langle \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ の改善目標となる入出力のバランスを示しているわけで，その中に採用されている活動の集合

$$\{\langle \boldsymbol{a}_i, \boldsymbol{b}_i \rangle \in \mathbb{R}^s \times \mathbb{R}^t \mid i \in \{1, \ldots, n\} \ \text{かつ} \ \lambda_{p,i}^* \neq 0\}$$

を活動 $\langle \boldsymbol{a}_p, \boldsymbol{b}_p \rangle$ の優位集合と呼ぶ。全ての活動に対して優位集合を調べた時，それらの中に最も頻繁に登場する活動は効率的フロンティアの中で最も普遍的な重要性を持っていると考えることができる。

3.　検証結果と課題

十種競技では，トラック競技の記録が小さな選手ほど評価が高くなり，フィールド競技の記録が大きな選手ほど評価が高くなる。これに対して，前節に述べた事業体活動の評価基準は，入力ベクトルの小さな活動ほど評価が高くなり，出力ベクトルの大きな活動ほど評価が高くなるといった類似性を持っている。そこで，トラック競技の記録を入力ベクトル，フィールド競技の記録を出力ベクトルとして，包絡分析法を適用することによって，全選手の分布（特に効率的フロンティア）の形状と各選手の位置に関する情報のみから優位性の分析を行うことが可能となる。また，それによって，完全に客観的な視点から各選手の評価を与えることが期待できる。このような評価方法の視点から，以下では，具体的なデータに対する得点計算ついて幾つかの簡単な検証を試みたい。

実際に検証を行う上で，標準的な記録の分布と考えられるサンプル集合が必要となる。これに対して，今回の考察では，表 1 のように 2009 年 1 月 1 日から 2009 年 10 月 14 日までの期間における世界上位 40 名の記録 [5] を採用した[5]。

表において，$a_{p,1}, a_{p,2}, a_{p,3}, a_{p,4}, b_{p,1}, b_{p,2}, b_{p,3}, b_{p,4}, b_{p,5}, b_{p,6}$ はそれぞれ競技者 p の 100m, 400m, 110m ハードル, 1500m, 走り幅跳び，砲丸投げ，走り高跳び，円盤投げ，棒高飛び，槍投げの記録を表している。

このような競技者の分布の下で，先ずは素朴に，全選手に対して包絡分析法を適用しその計算結果（D-効率値，余剰，不足，優位集合）と現時点で採用されている得点計算の結果を比較することを試みたい。実際に，選手 p に関する効率値を求めるためには，制約条件

$$\theta a_{p,q} - \sum_{i=1}^{40} \lambda_i a_{i,q} \geq 0 \quad (q = 1, \ldots, 4),$$

$$b_{p,r} - \sum_{i=1}^{40} \lambda_i b_{i,r} \leq 0 \quad (r = 1, \ldots, 6),$$

$$\sum_{i=1}^{40} \lambda_i = 1, \quad \lambda_i \geq 0 \ \ (i = 1, \ldots, 40)$$

を満たす最小の実数 θ を算出すればよい。そして，これは単なる線形計画問題であるので，シンプレックス法のプログラムを作成して直ちに目的の最小値 θ_p^* を得ることができる。この計算によって D-効率値と同時に係数 $\lambda_1, \ldots, \lambda_{40}$ の値も算出されるが，一般的に，それらに基づく凸結合

$$\Big\langle \sum_{i=1}^{40} \lambda_i \boldsymbol{a}_i,\ \sum_{i=1}^{40} \lambda_i \boldsymbol{b}_i \Big\rangle$$

[4]余剰や不足は効率的フロンティアからの模範活動の選択方針に依存して定義される相対的な概念である。

[5]本来は 33 番目に Massimo Bertocchi が入るが記録データが不明なため考察から外している。

が効率的フロンティアに属している保証はなく，必然的に，これを p の模範活動に採用することもできない。こうした不具合に対して，(何らかの基準に基づいて）既存の活動の凸結合として模範活動を構成し，$\langle \theta_p^* \boldsymbol{a}, \boldsymbol{b} \rangle$ と模範活動の間に存在する余剰・不足，優位集合に関する情報を改めて計算する必要がある。

表 1

p (氏名)	$a_{p,1}$	$a_{p,2}$	$a_{p,3}$	$a_{p,4}$	$b_{p,1}$	$b_{p,2}$	$b_{p,3}$	$b_{p,4}$	$b_{p,5}$	$b_{p,6}$	得点
1 (Trey Hardee)	10.45	48.13	13.86	288.91	7.83	15.33	1.99	48.08	5.2	68	8790
2 (Leonel Suarez)	11.07	47.65	14.15	267.29	7.42	14.39	2.09	46.07	4.7	77.47	8654
3 (Tom Pappas)	10.84	49.49	14.19	300.77	7.63	16.82	2.07	51.57	5.05	61.7	8569
4 (Aleksandr Pogorelov)	10.95	50.27	14.19	288.7	7.49	16.65	2.08	48.46	5.1	63.95	8528
5 (Michael Schrader)	10.64	49.71	14.21	262.26	8.05	14.33	1.94	43.09	5	64.04	8522
6 (Yordani Garcia)	10.88	48.77	14.07	286.8	7.36	16.5	2.1	43.97	4.8	68.1	8496
7 (Oleksiy Kasyanov)	10.63	47.85	14.44	264.52	7.8	15.72	2.05	46.7	4.8	49	8479
8 (Alexey Sysoev)	10.85	49.32	14.97	274.97	6.87	16.17	2.02	53.03	5.1	64.55	8454
9 (Pascal Behrenbruch)	10.92	48.72	14.24	279.45	7.09	15.77	2.02	48.06	4.8	69.72	8439
10 (Nicklas Wiberg)	10.96	48.73	14.75	257.05	7.25	14.99	2.05	42.28	4.5	75.02	8406
11 (Yunior Diaz)	10.66	46.15	14.56	280.58	7.72	14.54	2.02	43.52	4.6	60.09	8357
12 (Roman Šebrle)	11.14	49.5	14.33	288.88	7.86	15.41	1.94	46.71	4.9	67.54	8348
13 (Andrei Krauchanka)	11.18	48.82	14.28	272.6	7.54	13.69	2.1	43.01	5.1	61.56	8336
14 (Norman Müller)	10.91	48.95	14.54	269.2	7.47	15.27	2.02	43.65	4.8	60.89	8295
15 (Mikk Pahapill)	11.1	50.08	14.14	282.26	7.47	14.05	2.03	48.48	5.02	59.99	8255
16 (Romain Barras)	11.05	49.68	14.64	265.26	7.29	15.48	2.01	42.64	4.8	64.68	8239
17 (Larbi Bouraada)	10.68	48.58	14.57	252.15	7.35	12.3	2.05	37.83	4.7	62.53	8171
18 (Maurice Smith)	10.88	48.35	14.09	286.43	7.25	14.67	1.97	51.45	4.7	53.14	8157
19 (Willem Coertzen)	10.89	48.63	14.26	272.57	7.32	13.16	2.02	42.4	4.6	65.46	8146
20 (Andres Raja)	10.82	49	14.22	280.73	7.38	14.55	1.99	42.75	4.8	57.73	8119
21 (Vasiliy Kharlamov)	11.33	49.65	14.66	280.54	7.58	14.33	1.98	46.46	5	59.44	8113
22 (Eelco Sintnicolaas)	11.08	48.22	14.81	267.64	7.29	13.39	1.91	40.52	5.3	61.51	8112
23 (Ashton Eaton)	10.49	47.12	14.01	276.87	7.45	12.63	2.01	39.8	4.85	49.69	8091
24 (Eugene Martineau)	11.15	50.18	14.55	277.63	7.4	13.87	2	43.8	4.8	65.86	8083
25 (Aleksey Drozdov)	11.2	50.99	15.45	284.45	7.26	15.88	2.07	47.46	4.9	61.39	8081
26 (Hans van Alphen)	11.25	49.56	14.6	257.51	7.4	14.22	1.88	45.52	4.62	62.75	8070
27 (Jake Arnold)	11.06	49.17	14.27	271.13	6.92	14.2	2	45.76	4.9	56.84	8069
28 (Mateo Sossah)	11.31	49.19	14.62	260.5	7.3	13.63	2.08	42.44	4.6	58.88	8044
29 (Dmitriy Karpov)	11.07	50.64	14.63	293.23	6.97	16.02	2	50.79	4.9	55.89	8029
30 (Ingmar Vos)	10.9	49.99	14.51	268.51	7.21	13.78	2.02	42.39	4.4	64.27	8009
31 (Moritz Cleve)	10.75	48.39	14.28	274.44	7.31	14.34	1.95	41.04	4.35	56.11	8004
32 (Jacob Minah)	10.79	48.09	14.4	276.5	7.45	13.71	1.97	40.02	4.8	52.75	7991
33 (Roland Schwarzl)	11.24	50.37	14.53	276.01	7.55	13.85	1.97	44.88	5.1	51.08	7971
34 (Mikalai Shubianok)	11.52	49.79	14.78	269.26	7.2	14.31	2.01	44.06	4.6	64.31	7960
35 (Nadir El Fassi)	11.12	51.35	14.9	256.51	7.26	13.62	1.99	42.25	4.8	57.65	7922
36 (Brent Newdick)	11.11	50.1	14.82	270.57	7.42	14.35	1.99	43.6	4.8	51.52	7915
37 (Lars Albert)	11.29	51.47	14.87	285.38	7.17	15.76	1.83	51.04	4.95	58.57	7903
38 (Joe Detmer)	11.01	47.91	14.9	252.67	7.25	12	1.98	37.92	4.85	53.8	7892
39 (Alexis Chivas)	10.85	52.9	14.67	285	7.25	15.52	1.96	48.89	4.3	60.7	7879
40 (Stefan Drews)	11.09	50.05	14.33	270.29	7.44	12.78	1.96	39.09	5.1	50.91	7858

そのための手段の一つとして，上で得られた効率値 θ_p^* を用いて，更に

$$s_q = \theta_p^* a_{p,q} - \sum_{i=1}^{40} \lambda_i a_{i,q} \quad (q = 1, \ldots, 4),$$

$$l_r = b_{p,r} - \sum_{i=1}^{40} \lambda_i b_{i,r} \quad (r = 1, \ldots, 6),$$

$$\sum_{i=1}^{40} \lambda_i = 1, \quad \lambda_i \geq 0 \ \ (i = 1, \ldots, 40),$$

$$s_q \geq 0 \ \ (q = 1, \ldots, 6), \quad l_r \geq 0 \ \ (r = 1, \ldots, 4)$$

の条件下で

$$(1) \qquad \sum_{q=1}^{4} s_q + \sum_{r=1}^{6} l_r$$

を最大にする λ_i, s_q, l_r の値を（各 i, q, r に対して）全て求めることにする。前半の問題と同様に，この問題も線形計画問題の形式を持っているので，シンプレックス法のプログラムを用いて最適解を得ることが可能である。こうして得られた値をそれぞれ $\lambda_{p,i}^*$, $s_{p,q}^*$, $l_{p,r}^*$ と表記することにする。

上の手順によって得られた数値は最大スラック解と呼ばれるものであり，後半の問題で得られた係数 $\lambda^*_{p,1},\ldots,\lambda^*_{p,40}$ に基づいた凸結合の結果は常に効率的フロンティアに属していることが保証されている。そこで，これを p の模範活動と考えることにする。それに伴って，$s^*_{p,q}$ と $l^*_{p,r}$ はそれぞれ余剰と不足の数値を表すこととなる。これらの数値は，効率的フロンティアへの最短距離を表すわけではないので扱いには注意が必要である。しかし，一方で，これら全てが 0 となる時は，(1) の最大値も 0 となり余剰・不足が存在しないことが確実に保証される。

実際に，40 名の記録に対して上記の計算を適用すると各選手の D-効率値，余剰・不足として以下の結果が得られる。

表２

p (氏名)	θ^*_p	$s^*_{p,1}$	$s^*_{p,2}$	$s^*_{p,3}$	$s^*_{p,4}$	$l^*_{p,1}$	$l^*_{p,2}$	$l^*_{p,3}$	$l^*_{p,4}$	$l^*_{p,5}$	$l^*_{p,6}$
1 (Trey Hardee)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2 (Leonel Suarez)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3 (Tom Pappas)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4 (Aleksandr Pogorelov)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5 (Michael Schrader)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
6 (Yordani Garcia)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
7 (Oleksiy Kasyanov)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8 (Alexey Sysoev)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9 (Pascal Behrenbruch)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10 (Nicklas Wiberg)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
11 (Yunior Diaz)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
12 (Roman Sebrle)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
13 (Andrei Krauchanka)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
14 (Norman Muller)	0.98817	0.000	0.000	0.000	0.000	0.198	0.000	0.025	2.098	0.000	0.000
15 (Mikk Pahapill)	0.99849	0.360	1.739	0.000	0.000	0.092	1.235	0.000	0.000	0.000	9.567
16 (Romain Barras)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
17 (Larbi Bouraada)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
18 (Maurice Smith)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
19 (Willem Coertzen)	0.98961	0.000	0.182	0.000	0.000	0.257	0.677	0.009	1.065	0.225	0.000
20 (Andres Raja)	0.98589	0.055	0.000	0.000	0.000	0.382	0.000	0.008	2.606	0.215	8.476
21 (Vasiliy Kharlamov)	0.97242	0.396	0.000	0.000	0.000	0.187	0.898	0.026	0.000	0.000	0.000
22 (Eelco Sintnicolaas)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
23 (Ashton Eaton)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
24 (Eugene Martineau)	0.97024	0.077	0.391	0.000	0.000	0.281	0.186	0.012	0.000	0.079	0.000
25 (Aleksey Drozdov)	0.96951	0.000	0.542	0.689	0.000	0.307	0.000	0.000	0.000	0.016	0.000
26 (Hans van Alphen)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
27 (Jake Arnold)	0.99321	0.167	0.321	0.000	0.000	0.744	0.413	0.020	0.000	0.000	13.912
28 (Mateo Sossah)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
29 (Dmitriy Karpov)	0.97870	0.024	0.678	0.000	0.000	0.504	0.000	0.036	0.000	0.027	1.729
30 (Ingmar Vos)	0.98317	0.000	0.687	0.000	0.000	0.422	0.000	0.000	0.157	0.437	0.000
31 (Moritz Cleve)	0.99046	0.000	0.000	0.000	0.000	0.363	0.000	0.071	3.279	0.535	3.511
32 (Jacob Minah)	0.98449	0.000	0.000	0.000	0.000	0.090	0.000	0.061	2.485	0.005	1.191
33 (Roland Schwarzl)	0.98260	0.329	0.441	0.000	0.000	0.203	0.719	0.000	0.000	0.000	13.409
34 (Mikalai Shubianok)	0.97218	0.359	0.000	0.000	0.000	0.387	0.000	0.018	0.000	0.156	0.000
35 (Nadir El Fassi)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
36 (Brent Newdick)	0.96785	0.021	0.000	0.000	0.000	0.282	0.000	0.037	0.111	0.016	10.262
37 (Lars Albert)	0.97716	0.239	1.526	0.000	0.000	0.093	0.000	0.188	0.000	0.000	0.233
38 (Joe Detmer)	1.00000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
39 (Alexis Chivas)	0.97695	0.000	3.341	0.032	0.000	0.337	0.128	0.055	0.000	0.760	0.949
40 (Stefan Drews)	0.99535	0.297	0.749	0.000	0.000	0.381	1.492	0.000	4.414	0.000	13.120

全体的な傾向としては，順位が高い選手から低い選手に向かって D-効率値が減少しているが，厳密な意味で減少を示しているわけではなく，順位が低い選手の中にも D-効率値が 1 となるものが幾つも存在する。こうした特徴は，包絡分析法の評価の仕組みを考えるとある程度自然なことであり，1 つでも優れた特徴を持っている活動は（その他の数値がどんなに悪くても）その 1 つの長所が最大限に評価され効率フロンティアに到達するので必然的に高い評価が与えられる結果となる。ここで得られた結果に

は，ある程度タイプの異なる選手の存在が反映されていると考えられる。

各選手の優劣はそれらの模範活動の構造により反映されていると考えられる。そこで，各選手に対して，その模範活動を表す凸結合に採用されている要素と係数の数値，更に，他の選手の模範活動の中に採用された回数（優位集合への登場回数）を下表にまとめた。

表 3

p (氏名)	模範活動を表す線形結合に採用される要素（とその係数）	採用回数
1 (Trey Hardee)	1(1.000)	14
2 (Leonel Suarez)	2(1.000)	14
3 (Tom Pappas)	3(1.000)	4
4 (Aleksandr Pogorelov)	4(1.000)	3
5 (Michael Schrader)	5(1.000)	13
6 (Yordani Garcia)	6(1.000)	2
7 (Oleksiy Kasyanov)	7(1.000)	12
8 (Alexey Sysoev)	8(1.000)	10
9 (Pascal Behrenbruch)	9(1.000)	1
10 (Nicklas Wiberg)	10(1.000)	4
11 (Yunior Diaz)	11(1.000)	2
12 (Roman Sebrle)	12(1.000)	1
13 (Andrei Krauchanka)	13(1.000)	3
14 (Norman Muller)	1(0.004) 2 (0.192) 4(0.083) 5(0.134) 7(0.467) 8(0.003) 10(0.117)	0
15 (Mikk Pahapill)	1(0.476) 2(0.293) 3(0.090) 8(0.127) 18(0.015)	0
16 (Romain Barras)	16(1.000)	1
17 (Larbi Bouraada)	17(1.000)	4
18 (Maurice Smith)	18(1.000)	4
19 (Willem Coertzen)	1(0.023) 2(0.440) 5(0.219) 23(0.319)	0
20 (Andres Raja)	1(0.438) 2(0.176) 5(0.252) 23(0.134)	0
21 (Vasiliy Kharlamov)	1(0.325) 2(0.025) 5(0.142) 7(0.404) 8(0.038) 22(0.065)	0
22 (Eelco Sintnicolaas)	22(1.000)	4
23 (Ashton Eaton)	23(1.000)	7
24 (Eugene Martineau)	1(0.080) 2(0.348) 5(0.352) 23(0.221)	0
25 (Aleksey Drozdov)	2(0.190) 4(0.428) 6(0.007) 7(0.348) 8(0.025) 10(0.003)	0
26 (Hans van Alphen)	26(1.000)	2
27 (Jake Arnold)	1(0.151) 2(0.453) 5(0.339) 8(0.057)	0
28 (Mateo Sossah)	28(1.000)	1
29 (Dmitriy Karpov)	3(0.441) 7(0.194) 8(0.132) 18(0.233)	0
30 (Ingmar Vos)	1(0.061) 2(0.219) 5(0.281) 7(0.068) 17(0.266) 23(0.106)	0
31 (Moritz Cleve)	1(0.150) 2(0.187) 5(0.150) 7(0.233) 23(0.280)	0
32 (Jacob Minah)	2(0.139) 7(0.246) 11(0.045) 23(0.551) 38(0.019)	0
33 (Roland Schwarzl)	1(0.232) 5(0.466) 8(0.095) 13(0.090) 22(0.117)	0
34 (Mikalai Shubianok)	2(0.294) 5(0.153) 7(0.239) 10(0.030) 17(0.187) 26(0.098)	0
35 (Nadir El Fassi)	35(1.000)	1
36 (Brent Newdick)	2(0.201) 5(0.284) 7(0.308) 17(0.207)	0
37 (Lars Albert)	1(0.051) 3(0.100) 7(0.194) 8(0.425) 18(0.230)	0
38 (Joe Detmer)	38(1.000)	2
39 (Alexis Chivas)	1(0.466) 7(0.290) 8(0.244)	0
40 (Stefan Drews)	1(0.178) 5(0.539) 13(0.103) 22(0.180)	0

例えば，39 (Alexis Chivas) の模範活動は

$$\langle 0.466\boldsymbol{a}_1 + 0.290\boldsymbol{a}_7 + 0.244\boldsymbol{a}_8,\\ 0.466\boldsymbol{b}_1 + 0.290\boldsymbol{b}_7 + 0.244\boldsymbol{b}_8\rangle$$

のように 1 番，7 番，8 番の選手をそれぞれ 0.466, 0.290, 0.244 の割合で組合わせて構成される（仮想の）選手となる。

十種競技の主旨を考えると，全ての競技にバランスよく優れた記録を持つ万能タイプの選手がより高い評価を得るべきであると考えられる。（実際に，得点計算の方法が設計される過程においても，そうした点が配慮されてきた。）そこで，効率的フロンティアを構成していている選手（つまり採用回数が 0 でない選手）の中でも，採用回数が多い選手ほど，万能タイプの選手層（最も厚い選手層とも考えられる）

の中で秀でた成績を残していると見なすことが出来るので，より高い評価を得るのが自然であると考えられる。(逆に，採用回数が少ない選手は，特定種目の優位性によって効率的フロンティアに押し上げられていると考えられるので，高い評価を得るべきではないと考えられる。) こうした視点から表をみると，全体的な傾向としては，採用回数が多い選手に高い得点が与えられていることが分かり，現在の得点計算の下では「万能タイプの選手に高い評価を与える」という方針が比較的上手に評価に反映されていることが分かる。

但し，厳密に見ると，不公平な評価が与えられている場合も存在している。例えば，23 (Ashton Eaton) の採用回数は7回であるため，多くの選手の模範となる優れたバランスを持つ選手と考えることができる。特に，19 (Willem Coertzen) と 20 (Andres Raja) の模範活動の構成に採用されていていることから，本来はこれらの選手よりも高い評価を得るのが自然である。しかし，実際の得点にはそうした事実が全く反映されてなく，この選手が不当に評価されていることが分かる。類似の現象は他にも数箇所で確認でき，これらは現状の得点計算に関する一種の不健全性を示しているといってよい。

次に，第 2 の考察として，40 人の競技者の中に人工的に作成した競技者を意図的に配置して，それらの効率値を通して種目間の得点計算の公平性を検証したい。そのために先ず，各種目に対して,「40 人の記録の平均値」と「それらを標準偏差の分だけ改善して得られる改善値」を計算してみると下表の通りとなる。

表 4

	100m	400m	110mH	1500m
平均値	10.97775	49.28775	14.48225	274.4005
改善値	10.742003	48.026626	14.162318	262.89493

	走幅跳	砲丸投	走高跳	円盤投	棒高跳	槍投
平均値	7.388	14.5495	2.003	44.787	4.82475	60.727
改善値	7.632532	15.739529	2.060655	48.637568	5.052267	67.22481

以下では，全選手の分布の中で標準的な位置にいる選手を「10 種目の記録が全て平均値である選手」と仮定する。その上で，10 種類の各種目について，注目した 1 種目の記録だけが改善値をとる（残りの 9 種目は平均値をとる）ように標準的選手を（10 通りの方向に）改善することによって新たに 10 人の人工的な選手を作成する。そして，この 10 人の記録を従来の 40 人の記録に加えて包絡分析法を適用し，効率値の比較を行う。

結果として得られる D-効率値を，直ちに「検証対象の活動と効率的フロンティアの距離を表す指標」と考えるのは実態を十分に反映していない危険性がある。実際に，D-効率値が高くても余剰・不足が大きな選手は,(効率的フロンティアから離れていて) 特殊なバランスを持っていると考えられるので，十種競技の主旨から考えて優れた選手とは判断し難い。これに対して，余剰・不足の情報もある程度反映された指標があると望ましく，選手 p の効率を表す数値として，p とその模範活動との間で入出力ベクトルの比率をまとめた数値

$$\frac{\left(\sum_{q=1}^{4}\sum_{i=1}^{40}\lambda^{*}_{p,i}a_{i,q}\right)\left(\sum_{r=1}^{6}b_{p,r}\right)}{\left(\sum_{q=1}^{4}a_{p,q}\right)\left(\sum_{r=1}^{6}\sum_{i=1}^{40}\lambda^{*}_{p,i}b_{i,r}\right)}$$

を新たに導入することを試みる。この数値は p の T-効率値と呼ばれている。たとえ D-効率値が大きくても，余剰・不足が小さく抑えられていなければ T-効率値は小さくなり，余剰・不足が全くなく D-効率値が 1 となる時かつその時に限り，T-効率値が 1 となることが簡単に確認できる。こうした特徴からも，D-効率値と比較して，T-効率値に基づく考察の方が（相対的に）より実態を反映した結果を与えていると考えることができる。但し，ここで扱っている余剰・不足は最大スラック解に基づく数値あるこ

とから，T-効率値を用いても各活動から効率的フロンティアまでの最短距離を完全に捉えているわけではなく，扱いには注意が必要である。

新たな 10 名に対して,「それらを 40 人の選手に加えて包絡分析法を実行した際に観測される D-効率値と T-効率値」と「従来の計算式に従って得られる得点」をまとめると下表の通りとなる。それぞれの表においては，改善値の置かれた競技を明示することで 10 人の選手を表していて，それらを効率値や得点が大きい順に並べている。

表 5

種目	D-効率値
1500m	0.99773
110mH	0.99378
円盤投	0.99211
砲丸投	0.98950
400m	0.98903
棒高跳	0.98708
100m	0.98475
走高飛	0.98321
槍投	0.97959
走幅跳	0.97892

種目	T-効率値
1500m	0.99427
400m	0.98658
砲丸投	0.98400
棒高跳	0.98303
100m	0.97978
槍投	0.97460
円盤投	0.96744
走幅跳	0.94900
走高飛	0.94742
110mH	0.93225

種目	得点
槍投	8284
円盤投	8266
1500m	8262
砲丸投	8259
棒高跳	8256
走幅跳	8247
400m	8246
100m	8239
走高飛	8239
110mH	8227

D-効率値に対しては，その定義から，入力ベクトルの値が改善された方が有利に働くことが予想される。これに対して，実験結果では，実際に，トラック競技で改善値を持つ選手が高い値をとる傾向が観察される。また，D-効率値と T-効率値を対比してみると，円盤投げ，110m ハードルなどの種目で相対的に T-効率値の結果が低くなっていることが分かる。こうした結果から,（大雑把な観察ではあるが）円盤投げや 110m ハードルの種目に特化した選手は特殊なタイプであることが考えられる。

最後に，T-効率値と得点の関係を比較してみると，100m や 400m のような短距離種目でよい成績を出すと T-効率値は（相対的に）高い値をとるが，実際の得点は（相対的に）低いことに気付く。このことは，短距離種目の記録の改善は効率的フロンティアへの接近に有効であり，より高く評価されるべきであると考えられるが，実際の得点計算ではそのことが十分に評価されていないことを意味している。つまり，現在の得点計算の下では短距離種目が不利な扱いを受けている傾向が読み取れる。これと対照的に，槍投げや円盤投げなどの投擲種目は，T-効率値は低い値をとる一方で実際の得点は高くなっていて，これらの種目は有利な扱いを受けている傾向が読み取れる。こうした結果は [3] の主張とも一致している。

今回の考察では，十種競技得点計算の正当性を正確に検証することよりも，経営効率の測定方法を利用した検証方法を提案することに力点が置かれている。そのため，得られた結果は直ちに十種競技得点計算に関する真理を断定しているわけではなく，取り敢えず，そうした方針で素朴なアプローチを試みて得られた結果の報告を行っている。より信頼度の高い検証を行うためには，少なくとも以下の 2 点を改善することが不可欠であり，それによって初めて説得力のある結論が得られると予想される。

最初の課題は分析に使用したデータの改善にある。包絡分析法の計算結果は分析に使用した全競技者の大域的な分布構造に大きく依存する。その点で，今回の考察で使用したデータ数は極めて少なく，ある程度の信頼度を確保するにはより多くの記録データを参照することが必須である。（それに伴って，分析結果に余剰・不足が生じにくくなるといった副産物も期待できる。）更に，使用するデータの分布は 10 種競技の記録の分布として標準的なものでなければならないが，今回の考察で使用した 40 人を標準の分布と見なす根拠は何もない。場合によっては，そ

のような標準的な分布自体が存在せず，様々な選手層で記録の分布の形（特に効率的フロンティアの形）が異なる可能性も考えられる。こうしたことは，各選手層の間で今回と同じ検証を行えばある程度は明らかになると考えられ，今回の考察とは独立した興味深い問題と思われる。

もう一つの課題は分析に使用したアルゴリズムの改善にある。今回の考察では，包絡分析法の最も標準的なアルゴリズムをそのまま適用しただけで，考察対象の特殊性に配慮した調整は何も行っていない。例えば，競技結果の記録についても，100m と 1500m ではスケールに差が生じるので，T-効率値を求める際には，余剰・不足にウェイトを導入してより精密な議論を行った方が公平な評価ができると考えられる。（これに対して，現状の計算はウェイトを全て 1 に統一して全ての記録を一律に扱っている。）また，余剰・不足の数値自体も最大スラック解に基づいて算出されているが，より適切な模範活動の選択方法があると考えられる。特に，余剰・不足をできるだけ小さくとるように模範活動を選択することができれば，T-効率値はより妥当な指標を与えることになると思われる。

謝辞

本考察は，法政大学経営学部のセミナーを通して議論された内容を著者が代表して実験しまとめたものである。経営効率の評価法を十種競技の選手の評価に適用するというアプローチは小澤清貴さんの着想によるところが大きく，小澤さんとは関連研究の調査からデータのまとめに至るまで様々な面で協力して考察を行った。また，他の参加者の方々にもセミナーの中で実験結果に関する有益な感想を頂いた。少なくとも，こうした機会がなければ，著者が単独でこの分野の考察に触れることはなかったと考えられ，セミナーに参加された学生の方々に感謝したい。

参考文献

[1] 茨木俊秀, 福島雅夫，最適化の手法, 共立出版, 1993.
[2] 伊理正夫, 線形計画法, 共立出版, 1986.
[3] 菅原勲，石井隆士，水野増彦，日隈広至，正木健雄，陸上競技・混成競技における採点の等価性の検証，日本体育大学紀要 28 巻 1 号，pp. 27–30，1998.
[4] 刀根薫, 経営効率性の測定と改善, 日科技連, 1993.
[5] http://www.iaaf.org/statistics/toplists/

A.　付録

上記考察に対して，数値算出の根拠となるプログラムのソースコードを載せておく。プログラムは [1, 2] などで紹介されているアルゴリズムの基本的な部分を実装したもので，実際に計算を行った際に生じた不具合に対してのみ，それらを回避する仕組みを導入している。

主な問題の一つとして，double 型のようなデータ構造を安易に使用すると丸め誤差の累積が影響を与え，基底変数と非基底変数の交換を行う際に正しい選択ができないケースが生じることが挙げられる。これに対して，途中の計算を誤差が全く生じないように進める必要があり，全ての数値を 2 つの多倍長整数の組として有理数の形式で扱うこととした。それに伴い，有理数を表現するクラス Q を準備することとした。Q のフィールドは 2 つの BigInteger オブジェクト n と d からなり，それぞれが分子と分母の情報を管理する。Q のメソッドは主に有理数に関する四則演算と大小関係の判定から成る。例えば，

```
 1  static Q div(Q x, Q y) throws ArithmeticException{
 2    Q z;
 3
 4    z = new Q(0.0);
 5
 6    if(! y.n.equals(BigInteger.ZERO)){
 7      z.n = x.n.multiply(y.d);  z.d = x.d.multiply(y.n);
 8    } else {
 9      throw new ArithmeticException();
10    }
11
12    return optimize(z);
13  }
```

は Q のオブジェクト x, y に関する割算の評価を行い，分子として「x の分子と y の分母の積」を，分母として「x の分母と y の分子の積」をそれぞれ計算して（y の分子が 0 の時は，0 による割算となるので例外を投げている）最終的にそれらをメソッド optimize によって最適化して計算結果としている。最後に適用されている optimize は以下のように定義されていて，分母と分子を両者の最大公約数で約分する処理と，符号の情報を分子に与える処理を施している。

```
 1  private static Q optimize(Q x){
 2    BigInteger gcd;
 3
 4    gcd = x.n.gcd(x.d);
 5    x.n = x.n.divide(gcd);  x.d = x.d.divide(gcd);
 6
 7    if(x.d.compareTo(BigInteger.ZERO) < 0 ){
 8      x.n = x.n.negate();  x.d = x.d.negate();
 9    }
10
11    return x;
12  }
```

具体的なコードは省略するが，Q のオブジェクトに関する和差積の計算を行うメソッド add, sub, mult についても上で説明した div と同様に定義することができる。また，大小関係の判定は（符号の情報が分子にあるので）通分を行ってから分子の大小を比較すれば十分である。例えば，

```
 1  static boolean lt(Q x, Q y){
 2    BigInteger a, b;
 3
 4    a = x.n.multiply(y.d);  b = y.n.multiply(x.d);
 5
 6    if(a.compareTo(b) < 0){
 7      return true;
 8    } else {
 9      return false;
10    }
11  }
```

のように定義されるメソッド lt はオブジェクト x と y の大小を比較して x が y よりも小さい時には true を返し，そうでない時には false を返す。また，等号付の大小関係の判定を行うメソッド leq と等号の判定を行うメソッド eq も同様に定義することとする。Q のオブジェクトを出力する際には，以下のメソッド toDouble によって，実際に割算を評価して double 型のデータに変換することとする。

```
 1  static double toDouble(Q x){
 2    BigDecimal numerator, denominator, result;
 3
 4    numerator = new BigDecimal(x.n);  denominator = new BigDecimal(x.d);
 5    result = numerator.divide(denominator, 50, BigDecimal.ROUND_HALF_UP);
 6
 7    return result.doubleValue();
 8  }
```

対照的に，Q のオブジェクトを生成する際は，以下のコンストラクタを用いて double 型のデータを引数として受け取り，それを分子に設定して，小数点以下の部分がなくなるまで分子と分母に 10 を掛けることを繰り返して有理数を生成する。例えば，Q(-12.345) によって生成されるオブジェクトの n フィールドは -12345 となり，d フィールドは 1000 となる。なお，変換の途中段階で long 型のデータを経由しているため生成できる数値には限度がある。より大きなデータを扱う際には配慮が必要と思われるが，今回の実験で扱われた数値について不具合が生じることはない。

```
 1  Q(double r){
 2    long numerator, denominator;
 3
 4    denominator = 1;  numerator = (long) r;
 5
 6    while(numerator != denominator * r){
 7      denominator = 10 * denominator;  numerator = (long) (denominator * r);
 8    }
 9
10    this.d = new BigInteger(Long.toString(denominator));
11    this.n = new BigInteger(Long.toString(numerator));
12  }
```

この基本的なデータ構造に基づいて，更に，各選手に必要な情報を管理するためのクラス DMU を以下のように定義する。DMU はフィールドを 7 つ持っていて，name は選手の名前を，IN と OUT はそれぞれ入力ベクトルと出力ベクトルの数値を（表 1 の順番に従って），Score は計算結果として得られる D-効率値を，Superiority は模範活動となる凸結合の係数を，INwaste と OUTlack はそれぞれ余剰と不足の数値を管理する。DMU のコンストラクタは name，IN，OUT フィールドを引数として要求していて，それらに選手のデータを渡すことによって各選手に対応するオブジェクトを生成する。

```
 1  class DMU{
 2    String name;
 3    double[] IN, OUT, Superiority, INwaste, OUTlack;
 4    Q Score;
 5
 6    DMU(String n, double[] i, double[] o){
 7      this.IN = new double[i.length];    this.INwaste = new double[i.length];
 8      this.OUT = new double[o.length];   this.OUTlack = new double[o.length];
 9
10      this.name = n;  this.IN = i;  this.OUT = o;
11    }
12  }
```

以上の準備の下で，実際の計算を行うクラス BCC を記述する。BCC はクラスフィールドとして DMU の配列 dmu を持ち，今回の例では，以下のように，dmu[0] から dmu[39] に「40 人分の氏名・入力ベクトル・出力ベクトルの情報」を（コンストラクタの指示通りに）与えて初期化している。

```
static DMU[] dmu = {
  new DMU("1 (Trey Hardee)",
          new double[] {10.45, 48.13, 13.86, 288.91},
          new double[] {7.83, 15.33, 1.99, 48.08, 5.2, 68}),
                        ⋮
  new DMU("40 (Stefan Drews)",
          new double[] {11.09, 50.05, 14.33, 270.29},
          new double[] {7.44, 12.78, 1.96, 39.09, 5.1, 50.91})
};
```

BCC のクラスフィールドとしては，その他にも，制約条件の係数を保存するための 2 次元配列 A，制約条件の定数項を保存するための配列 B，目的関数の係数を保存するための配列 C，シンプレックス法の途中で算出される相対コスト係数を保存するための配列 RCC，目的関数の値を保存するための f があり，これらの要素は全て Q のオブジェクトとなっている。また，非基底変数と基底変数に対してそれぞれに通し番号を導入することとして，「非基底変数に関する通し番号」と「基底変数に関する通し番号」を表すための int 型の配列 h と k を準備する。計算の全体的な構造は以下の通りである。

```
 1  public static void main(String[] args){
 2    try{
 3      for(int dmucounter = 0; dmucounter < dmu.length; dmucounter++){
 4        dmu[dmucounter].Superiority = new double[dmu.length];
 5
 6        Generate1stPhaseTableauForLPProblem1(dmucounter);
 7        Simplex(A[0].length);
 8        for(int i = 0; i < k.length; i++)
 9          if(A[0].length - dmu[0].OUT.length <= k[i])
10            ExcludeArtificalVariable(i, A[0].length - dmu[0].OUT.length - 1);
11        Generate2ndPhaseTableauForLPProblem1();
12        Simplex(A[0].length - dmu[0].OUT.length - 1);
13        dmu[dmucounter].Score = Q.mult(f, new Q(-1.0));
14
15        Generate1stPhaseTableauForLPProblem2(dmucounter);
16        Simplex(A[0].length);
17        for(int i = 0; i < k.length; i++)
18          if(A[0].length - (dmu[0].IN.length + dmu[0].OUT.length) <= k[i])
19            ExcludeArtificalVariable(i,
20                        A[0].length - dmu[0].IN.length - dmu[0].OUT.length - 1);
21        Generate2ndPhaseTableauForLPProblem2();
22        Simplex(A[0].length - dmu[0].IN.length - dmu[0].OUT.length - 1);
23
24        SaveResult(dmucounter);
25        System.out.printf("%3d:%s に関する計算が終わりました．%n",
26                        dmucounter, dmu[dmucounter].name);
27      }
28      OutputResult();
29    } catch (Exception e){
30      e.printStackTrace();
31    }
32  }
```

この手続きでは，dmucounter の数値を 0 から 39 の範囲で動かしながら，対応する選手の情報を以下の手順で計算している。先ず，4 行目で計算対象となる選手のオブジェクトに対して Superiority フィールドの領域を確保している。（配列 dmu を初期化する段階では選手の総数を取得できず，Superiority フィールドの領域は確保することができなかった。）その後，6–13 行目において前半の線形計画問題の最適解を算出している。最適解の計算は 2 段階シンプレックス法に基づいていて，具体的には，Generate1stPhaseTableauForLPProblem1 によって（人為変数を含んだ）第 1 段階のタブローを作成し，Simplex によってシンプレックス法を適用している。この時点で，基底変数の中にまだ人為変数が混ざっている時は，8–10 行目で検出され，ExcludeArtificalVariable によって明示的に人為変数を排除する処理が施される。ここまでの計算が問題なく遂行されると，与えられた線形計画問題の実行可能基底解の 1 つを発見することができる。そこで，Generate2ndPhaseTableauForLPProblem1 によって，第 1 段階の手続きで得られた情報から人為変数の部分を排除することで第 2 段階のタブローを作成し，再度 Simplex を適用することによって最適解を求め D-効率値の情報を得ることができる。15–22 行目では，上と同様の手順によって後半の線形計画問題の最適解を算出していて，これによって模範活動や余剰・不足の情報を得ることができる。これらの処理で得られた数値は，SaveResult によって，各選手に対応するオブジェクトの中に記録され，最終的に全ての活動の数値が確定した段階で OutputResult によって出力している。（SaveResult と OutputResult はどちらも計算の本質的な部分ではないので，コードや説明は省略する。）なお，Simplex の中で例外を投げている特殊な状況が 2 箇所存在するので，main の中でそれらを受け取るようにしているが，実際の計算でそのような例外が発生するケースは 1 回もなかった。

以下では，main の中で利用されているメソッドの内容に触れたい。先ず，前半の線形計画問題に対する第 1 段階のタブローを作成するためのメソッド Generate1stPhaseTableauForLPProblem1 は以下の通りとなっている。

```
 1  private static void Generate1stPhaseTableauForLPProblem1(int dmucounter){
 2    A = new Q[dmu[0].IN.length + dmu[0].OUT.length + 1]
 3             [2 + dmu.length + dmu[0].IN.length + 2*dmu[0].OUT.length];
 4    B = new Q[A.length];
 5    C = new Q[A[0].length];
 6    h = new int[A[0].length - A.length];
 7    k = new int[A.length];
 8
 9    for(int i = 0; i < dmu[0].IN.length; i++){
10      A[i][0] = new Q(-dmu[dmucounter].IN[i]);
11      for(int j = 0; j < dmu.length; j++)
12        A[i][1 + j] = new Q(dmu[j].IN[i]);
13      for(int j = 0; j < dmu[0].OUT.length; j++)
14        A[i][1 + dmu.length + j] = new Q(0.0);
15      for(int j = 0; j < dmu[0].IN.length; j++)
16        if(i == j){
17          A[i][1 + dmu.length + dmu[0].OUT.length + j] = new Q(1.0);
18        } else {
19          A[i][1 + dmu.length + dmu[0].OUT.length + j] = new Q(0.0);
20        }
21      for(int j = 0; j < dmu[0].OUT.length; j++)
22        A[i][1 + dmu.length + dmu[0].OUT.length + dmu[0].IN.length + j] = new Q(0.0);
23      A[i][1 + dmu.length + dmu[0].IN.length + 2*dmu[0].OUT.length] = new Q(0.0);
24    }
25
26    for(int i = 0; i < dmu[0].OUT.length; i++){
27      A[dmu[0].IN.length + i][0] = new Q(0.0);
28      for(int j = 0; j < dmu.length; j++)
29        A[dmu[0].IN.length + i][1 + j] = new Q(dmu[j].OUT[i]);
30      for(int j = 0; j < dmu[0].OUT.length; j++)
31        if(i == j){
32          A[dmu[0].IN.length + i][1 + dmu.length + j] = new Q(-1.0);
33        } else {
34          A[dmu[0].IN.length + i][1 + dmu.length + j] = new Q(0.0);
35        }
36      for(int j = 0; j < dmu[0].IN.length; j++)
37        A[dmu[0].IN.length + i][1 + dmu.length + dmu[0].OUT.length + j] = new Q(0.0);
38      for(int j = 0; j < dmu[0].OUT.length; j++)
39        if(i == j){
40          A[dmu[0].IN.length + i]
41           [1 + dmu.length + dmu[0].OUT.length + dmu[0].IN.length + j] = new Q(1.0);
42        } else {
43          A[dmu[0].IN.length + i]
44           [1 + dmu.length + dmu[0].OUT.length + dmu[0].IN.length + j] = new Q(0.0);
45        }
46      A[dmu[0].IN.length +i]
47       [1 + dmu.length + dmu[0].IN.length + 2*dmu[0].OUT.length] = new Q(0.0);
48    }
49
50    A[dmu[0].IN.length + dmu[0].OUT.length][0] = new Q(0.0);
51    for(int i = 0; i < dmu.length; i++)
52      A[dmu[0].IN.length + dmu[0].OUT.length][1 + i] = new Q(1.0);
53    for(int i = 0; i < dmu[0].IN.length + 2*dmu[0].OUT.length; i++)
54      A[dmu[0].IN.length + dmu[0].OUT.length][1 + dmu.length + i] = new Q(0.0);
55    A[dmu[0].IN.length + dmu[0].OUT.length]
56     [1 + dmu.length + dmu[0].IN.length + 2*dmu[0].OUT.length] = new Q(1.0);
57
58    for(int i = 0; i < dmu[0].IN.length; i++)
59      B[i] = new Q(0.0);
60    for(int i = 0; i < dmu[0].OUT.length; i++)
61      B[dmu[0].IN.length + i] = new Q(dmu[dmucounter].OUT[i]);
62    B[dmu[0].IN.length + dmu[0].OUT.length] = new Q(1.0);
63
64    for(int i = 0; i < 1 + dmu.length + dmu[0].IN.length + dmu[0].OUT.length; i++)
65      C[i] = new Q(0.0);
66    for(int i = 0; i < dmu[0].OUT.length + 1; i++)
```

```
67      C[1 + dmu.length + dmu[0].IN.length + dmu[0].OUT.length + i] = new Q(-1.0);
68
69    for(int i = 0; i < A[0].length - A.length; i++)
70      h[i] = i;
71    for(int i = 0; i < A.length; i++)
72      k[i] = A[0].length - A.length + i;
73  }
```

9–56 行目で配列 A の値を設定している。例えば，dmucounter が 10 の時[6]は，制約条件に出現する変数，サープラス変数，スラック変数，人為変数に対して対応する係数の値が下表の順番で A の中に代入されることとなる。また，配列の添字として，表の左端から順番に全ての変数に対して 0 からの通し番号が割り当てられるので，これを各変数の大域的な変数番号と考えて以後の処理の中で利用することにする。（この例の場合は，0 番から 57 番までの変数が存在することとなる。）

表 6

初期非基底変数											初期基底変数							
θ	λ_1	$\cdots$	λ_{40}	サープラス							スラック				人為							
-dmu[10].IN[0]	dmu[0].IN[0]	$\cdots$	dmu[39].IN[0]	0	0	0	0	0	0		1	0	0	0	0	0	0	0	0	0	0	
-dmu[10].IN[1]	dmu[0].IN[1]	$\cdots$	dmu[39].IN[1]	0	0	0	0	0	0		0	1	0	0	0	0	0	0	0	0	0	
-dmu[10].IN[2]	dmu[0].IN[2]	$\cdots$	dmu[39].IN[2]	0	0	0	0	0	0		0	0	1	0	0	0	0	0	0	0	0	
-dmu[10].IN[3]	dmu[0].IN[3]	$\cdots$	dmu[39].IN[3]	0	0	0	0	0	0		0	0	0	1	0	0	0	0	0	0	0	
0	dmu[0].OUT[0]	$\cdots$	dmu[39].OUT[0]	-1	0	0	0	0	0		0	0	0	0	1	0	0	0	0	0	0	
0	dmu[0].OUT[1]	$\cdots$	dmu[39].OUT[1]	0	-1	0	0	0	0		0	0	0	0	0	1	0	0	0	0	0	
0	dmu[0].OUT[2]	$\cdots$	dmu[39].OUT[2]	0	0	-1	0	0	0		0	0	0	0	0	0	1	0	0	0	0	
0	dmu[0].OUT[3]	$\cdots$	dmu[39].OUT[3]	0	0	0	-1	0	0		0	0	0	0	0	0	0	1	0	0	0	
0	dmu[0].OUT[4]	$\cdots$	dmu[39].OUT[4]	0	0	0	0	-1	0		0	0	0	0	0	0	0	0	1	0	0	
0	dmu[0].OUT[5]	$\cdots$	dmu[39].OUT[5]	0	0	0	0	0	-1		0	0	0	0	0	0	0	0	0	1	0	
0	1	$\cdots$	1	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	1	

次に，58–62 行目では，配列 B に制約条件の定数項の情報を

```
0 0 0 0 dmu[10].OUT[0] ··· dmu[10].OUT[5] 1
```

の順番で保存している。64–67 行目では，配列 C に目的関数の係数の情報を与えている。第 1 段階の手続きでは，人為変数の係数を全て −1 にして，それ以外の係数を全て 0 にする必要がある[7]。最後に，69–72 行目では，（初期の）非基底変数・基底変数となるべき大域的変数番号を指定している。非基底変数の番号は配列 h で管理されていて，h[i] には i 番目の非基底変数に相当する変数の大域的変数番号が記憶される。シンプレックス法の開始時における非基底変数は「制約条件に出現する変数とサープラス変数」であるから，0 から 46 までの番号が順番に h に保存されることとなる。同様に，基底変数の大域的変数番号は配列 k で管理されていて，k[i] には i 番目の基底変数に相当する変数の大域的変数番号が記憶される。従って，残りの 47 から 57 までの番号が k に保存されることとなる。

第 2 段階のタブローを作成するためのメソッド Generate2ndPhaseTableauForLPProblem1 は以下の通りであり，目的関数の係数を本来の数値に戻すことのみを行っている。（配列 A, B, h, k の値は，既に，第 1 段階の手続きによって完成している。）具体的には，（θ に相当する）第 0 番目の変数に対して係数を −1 にして，それ以外の係数を 0 にしている。

```
1  private static void Generate2ndPhaseTableauForLPProblem1(){
2    C[0] = new Q(-1.0);
3    for(int i = 0; i < dmu.length + dmu[0].IN.length + dmu[0].OUT.length; i++)
```

[6]配列の添字は 1 ずれているので第 11 番目の活動に対する計算となる。

[7]メソッド Simplex は最大化問題の形式でシンプレックス法の適用を行っているので，人為変数の係数を負にしておく必要がある。

```
4       C[1 + i] = new Q(0.0);
5   }
```

後半の線形計画問題についても同様に，2段階シンプレックス法の手続きに基づいて解を算出していて，Generate1stPhaseTableauForLPProblem2 によって，実行可能基底解を発見するために必要な第1段階の初期タブローを作成している。例えば，dmucounter の値が10の時には，配列 A の値は下表の通りとなる。（条件が全て等式の形をしているので，スラック変数やサープラス変数は存在しない。）その後，Generate2ndPhaseTableauForLPProblem2 によって第2段階のタブローを作成している。

表7

初期非基底変数												初期基底変数										
λ_1	$\cdots$	λ_{40}	s_1	s_2	s_3	s_4	l_1	l_2	l_3	l_4	l_5	l_6	人為									
dmu[0].IN[0]	$\cdots$	dmu[39].IN[0]	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
dmu[0].IN[1]	$\cdots$	dmu[39].IN[1]	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
dmu[0].IN[2]	$\cdots$	dmu[39].IN[2]	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
dmu[0].IN[3]	$\cdots$	dmu[39].IN[3]	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
dmu[0].OUT[0]	$\cdots$	dmu[39].OUT[0]	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
dmu[0].OUT[1]	$\cdots$	dmu[39].OUT[1]	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
dmu[0].OUT[2]	$\cdots$	dmu[39].OUT[2]	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0
dmu[0].OUT[3]	$\cdots$	dmu[39].OUT[3]	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0	0
dmu[0].OUT[4]	$\cdots$	dmu[39].OUT[4]	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1	0
dmu[0].OUT[5]	$\cdots$	dmu[39].OUT[5]	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	1
1	$\cdots$	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

また配列 B の値は，前半の線形計画問題から得られた D-効率値 θ_{11}^* を参照して，

```
θ*11 * dmu[10].IN[0]  ···  θ*11 * dmu[10].IN[3]
     dmu[10].OUT[0]   ···   dmu[10].OUT[5]   1
```

とする。（他の配列の初期値は前半の問題と同様に定義する。）また，第2段階のタブローにおいて，配列 C の値は $s_1, \ldots, s_4, l_1, \ldots, l_6$ に対応する係数のみを1とし，それ以外の係数は全て0とする。

以上の手続きによって生成されるタブローに対して，毎回適用されるシンプレックス法のアルゴリズムは広く知られたものであり，以下のように定義される。

```
 1  private static void Simplex(int boundary) throws Exception {
 2    int in, out, temp;
 3
 4    do{
 5      in = GetIncludeIndex(boundary);
 6      if (in != -1){
 7        out = GetExcludeIndex(h[in]);
 8        temp = h[in];  h[in] = k[out];  k[out] = temp;
 9        Pivoting(out, k[out], boundary);
10      }
11    }while(in != -1);
12  }
```

先ず，5行目の GetIncludeIndex によって「新たに基底変数に取り入れるべき非基底変数の番号」が算出される。特に，GetIncludeIndex の評価結果が -1 となる時は，取り入れるべき非基底変数が存在せず最適解が得られたことを意味しているので計算を終了する。対照的に，取り入れるべき非基底変数が存在する場合は，更に，7行目の GetExcludeIndex によって「非基底変数へ排除すべき基底変数の番号」が算出され，8行目で「in 番目の非基底変数として認識されている大域的変数番号」と「out 番目の基底変数として認識されている大域的変数番号」の情報を入れ替える。そして，最後に Pivoting によって out 行 k[out] 列を軸にした行列の掃き出し計算を行う。

以下では，これらの3つのメッソドの処理を確認したい。先ず，GetIncludeIndex の定義は

```
 1  private static int GetIncludeIndex(int boundary){
 2    boolean initialize;
 3    int in;
 4
 5    RCC = new Q[boundary];
 6
 7    for(int i = 0; i < boundary; i++){
 8      RCC[i] = new Q(0.0);
 9      for(int j = 0; j < A.length; j++)
10        RCC[i] = Q.add(RCC[i], Q.mult(C[k[j]], A[j][i]));
11      RCC[i] = Q.sub(C[i], RCC[i]);
12    }
13
14    f = new Q(0.0);
15    for(int i = 0; i < A.length; i++)
16      f = Q.add(f, Q.mult(C[k[i]], B[i]));
17
18    initialize = false;  in = -1;
19
20    for(int i = 0; i < h.length; i++)
21      if(initialize == false){
22        if(h[i] < boundary && Q.lt(new Q(0.0), RCC[h[i]])){
23          initialize = true;  in = i;
24        }
25      } else {
26        if(h[i] < boundary && Q.lt(new Q(0.0), RCC[h[i]]) && h[i] < h[in])
27          in = i;
28      }
29
30    return in;
31  }
```

となっている。7–12 行目において，各変数に対して相対コスト係数の値を計算して RCC に保存している。また，14–16 行目では，同様の計算を定数項に適用して目的関数の値を求め f に保存している。その後，20–28 行目で全ての非基底変数に対して相対コスト係数の値を比較して，0 よりも大きなものが存在すれば，その中で「大域的変数番号が最も小さな非基底変数の番号」を求める。（なお，全てが 0 以下の場合は計算を終了するため −1 を返すことにする。）

```
 1  private static int GetExcludeIndex(int max) throws Exception {
 2    boolean  solvability;
 3    int out;
 4    Q temp;
 5
 6    solvability = false;  out = 0;  temp = new Q(0.0);
 7
 8    for(int i = 0; i < A.length; i++){
 9      if(Q.lt(new Q(0.0), A[i][max]) && Q.leq(new Q(0.0), Q.div(B[i], A[i][max])))
10        if(solvability == false){
11          solvability = true;  out = i;  temp = Q.div(B[i], A[i][max]);
12        } else {
13          if((Q.eq(Q.div(B[i], A[i][max]), temp) && k[i] < k[out]) ||
14          Q.lt(Q.div(B[i], A[i][max]), temp)){
15            out = i;  temp = Q.div(B[i], A[i][max]);
16          }
17        }
18    }
19
20    if(solvability == false)  throw new Exception();
21    return out;
22  }
```

GetExcludeIndex では，8–18 行目において，制約条件の各式に対して，「GetIncludeIndex で得られた非基底変数」の係数で定数項を割った結果を評価している。そして，その除算の結果が正で最小となる式を求め，その式に対応する基底変数の番号を返す。特に，同じ評価結果となるものが複数存在する場合は，（巡回に陥らないように）最小添字規則に従い「大域的変数番号が最も小さな基底変数」を選択することとする。また，正になるものが一つも存在しない場合は，必然的に実行可能領域が非有界となる。その場合は 20 行目で検出され，例外を投げて処理を終了させている。

Pivoting の定義は以下の通りとなっていて，配列 A と B に対して，l 行 c 列を軸とした掃き出しを行う。これは行列の計算に関して広く知られているものと全く同じであり，特に注意する点はない。

```
 1  private static void Pivoting(int l, int c, int boundary){
 2    Q k;
 3
 4    k = A[l][c];
 5    for(int i = 0; i < boundary; i++)
 6      A[l][i] = Q.div(A[l][i], k);
 7    B[l] = Q.div(B[l], k);
 8
 9    for(int i = 0; i < A.length; i++){
10      if(i != l){
11        k = A[i][c];
12        for(int j = 0; j < boundary; j++)
13          A[i][j] = Q.sub(A[i][j], Q.mult(k, A[l][j]));
14        B[i] = Q.sub(B[i], Q.mult(k, B[l]));
15      }
16    }
17  }
```

以上のシンプレックス法の処理と独立して「基底変数に残っている人為変数」を排除するためのメソッド ExcludeArtificalVariable があり，その定義は以下の通りとなっている。

```
 1  private static void ExcludeArtificalVariable(int l, int boundary) throws Exception {
 2    int temp;
 3    boolean exclusion;
 4
 5    exclusion = false;
 6
 7    for(int i = 0; i < h.length; i++){
 8      if(h[i] < boundary && Q.toDouble(A[l][h[i]]) != 0.0){
 9        Pivoting(l, h[i], A[0].length);
10        temp = h[i];  h[i] = k[l];  k[l] = temp;
11        exclusion = true;
12        break;
13      }
14    }
15
16    if(exclusion == false)  throw new Exception();
17  }
```

ExcludeArtificalVariable は l 番目の基底変数を排除する処理を行う。具体的には，l 番目の制約条件式の中で「0 でない係数を持つ非基底変数で人為変数ではないもの」を探し，そのような非基底変数が存在すれば，これを新たに基底変数として採用することにして，l 番目の基底変数と入れ替える作業を行う。なお，入れ替えの対象となる非基底変数を見つけることができなかった場合は，例外を投げて処理を終了している。（この場合は，「タブローから l 行と k[l] 列を消去する処理」を行えば計算を継続することができるが，今回の実験ではこうした状態に陥るケースはなく対処する必要がなかった。）