Post Keynesian Review
Vol. 7, No. 1, Pages 1–13 (the 1st November, 2019)
ISSN 2187-4875

Linear Programming with Moore-Penrose Inverses and the Gravitation Method

LI Bangxi, LU Rui, ZHAO Yihan and Yoriaki Fujimori

Abstract. This paper deals with an application of MP-inverse to linear programming by introducing the gravitation method. The idea starts from obtaining the solution of a usual system of equations with a rectangular coefficient matrix. The first solution vector represented in terms of MP-inverses may contain non-positive components. In the light of the gravitation method, we improve solution vectors step by step to the direction with a better magnitude of the objective function. We apply MP-inverses in our algorithm of finding the optimum solutions. The gravitational method displays a concrete incremental way to optimize allocation rule.

***Keywords*:** LP, Moore-Penrose inverse, Gravitation method, Optimization

1. **Introduction**

In linear economics, linear programming plays an important role in describing the framewok of analysis and proving theorems. The major theme of this article is to present an algorithm of solving linear programming problems with the application of Moore-Penrose inverses.

In Section 2, fundamentals of MP-inverses are described. In Section 3, it will be shown that solutions of the system of linear equations are represented precisely with MP-inverses of the coefficient matrix.

In Section 4, the procedure of the algorithm will be explained in details, and we will prove the completeness and the correctness for every step of the algorithm. In Section 5, we will simulate step-by-step the algorithm with a concrete numerical example in Section 6. It will be demonstrated that our algorithm converges to the optimal point. In Appendix, a python script is presented to demonstrate the numerical example.

Refer to, *e.g.*, Rao and Mitra(1971), Campbell and Meyer(1991) and Ben-Israel and Greville(2003), for the details of mathematical preliminaries. Also, refer to Pyle(1967,1972,1973), from which an important inspiration was bestowed to the authors.

As for the mathematical notations, the following symbols are employed. **0** indicates the column zero-vector of relevant dimensions. **O** denotes a zero-matrix of relevant dimensions.

The first draft received: the 3rd of September, 2018. The final accepted: the 1st of October, 2019.

2. **Moore-Penrose Inverses**

Our discussion begins with the singular value decompositon, SVD, of matrices.

The singular value decomposition of matrix A is often represented by the form

$$A = U\Sigma V^T, \tag{2.1}$$

where U and V are unitary matrices of dimension $m \times m$ and $n \times n$, respectively, Σ is a diagonal matrix with the same shape as A. It should be pointed out that if there are k non-zero entries of matrix Σ on diagonal, then the first k columns of U and V form the orthonormal basis of the column space and the row space of A, respectively.

Corresponding to Σ, we can define an $n \times m$ diagonal matrix, with diagonal elements $\frac{1}{\sigma_i}$. This matrix is represented by Σ^+. namely, for $\Sigma_{m\times n} = \begin{pmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \ddots \end{pmatrix}$, also define $\Sigma^+_{n\times m} = \begin{pmatrix} \frac{1}{\sigma_1} & & \\ & \frac{1}{\sigma_2} & \\ & & \ddots \end{pmatrix}$. Further, the A^+ is derived from

$$A^+ = V\Sigma^+U^T. \tag{2.2}$$

Mathematical manipulation shows that the following four equations are fulfilled for A and A^+:

$$AA^+A = A,\ A^+AA^+ = A^+,\ (AA^+)^* = AA^+,\ (A^+A)^* = A^+A. \tag{2.3}$$

A^+ as defined above is called the *Moore-Penrose inverse*, in short MP-inverse, of A.

Consider what matrix A does to a vector: first by taking product of x with the inverse of matrix V, x will be decomposed into coordinates in $\mathbb{R}^n$ with respect to basis in columns of V. Only those coordinates which correspond to the basis lying in the row space would remain non-zero by multiplication of Σ. Finally, this product is the coordinate in the column space; multiply by U, and we get the final vector $Ax = U\Sigma V^T x$. Hence, we see that once there exists an x such that $Ax = b$, which means $b \in \mathcal{R}(A)$, then A^+b would definitely be a satisfying vector, because $A(A^+b) = (AA^+)b = b$. This can also be verified by $AA^+ = (U\Sigma V^T)(V\Sigma^+U^T) = UI^{(r)}U^T, r = \text{rank}(A)$. The deduction entails that AA^+ (*resp.* A^+A) *is the projection matrix onto the column* (*resp. the row*) *space of* A.

3. **Derive the Whole Solution Set of** $Ax = b$

The solution to a system of linear equations can be written in the form of $x = x_p + x_n$, where x_p is called the *particular solution* and x_n called the *null solution*. Let x_p be a particular solution to equations where $Ax_p = b$, and x_n be all vectors in the null space which satisfies $Ax_n = \mathbf{0}$. Normally finding either x_p or x_n is not easy, however, with MP-inverse, we can simplify both x_p and x_n as follows:

$$x_p = A^+b, \tag{3.1}$$

$$x_n = (I - A^+A)y. \tag{3.2}$$

Here, we let y run over all possible vectors among $\mathbb{R}^n$. We need to verify two results:

$$Ax_p = AA^+b = b, \tag{3.3}$$

$$A(I - A^+A)y = \mathbf{0},\ \forall y \in \mathbb{R}^n. \tag{3.4}$$

The first equation is obvious. As for the second, in view of $AA^+A = A$, we obtain

$$A(I - A^+A)y = (A - AA^+A)y = \mathbf{O}y = \mathbf{0}.$$

What we need to emphasize is that, set $\{x\colon x = (I - A^+A)y, y \in \mathbb{R}^n\} = \{x\colon Ax = \mathbf{0}\}$. By deduction above, we demonstrated that $\{x\colon x = (I - A^+A)y, y \in \mathbb{R}^n\} \subset \{x\colon Ax = \mathbf{0}\}$, but actually there is no x_n left. This is because, simply let $y = x_n$, and we have $x = (I - A^+A)x_n = x_n - A^+(Ax_n) = x_n$. Hence, $\{x\colon Ax = \mathbf{0}\} \subset \{x\colon x = (I - A^+A)y, y \in \mathbb{R}^n\}$, and two sets are equal. This originates from matrix $I - A^+A$ is the projection to the nullspace, the complement of the row space.

4. **LP and Moore-Penrose-Inverse**

Consider a canonical form of maximising linear programming:

By adding slack variables, one obtains the equality form of linear programming as:

$$\text{Maximize } (x, c) \quad \text{s.t. } Ax = b, x \geq \mathbf{0}, \tag{4.1}$$

where $A \in \mathbb{R}^{m\times n}$ and $x \in \mathbb{R}^n$.

All the solutions (not necessarily unique) to equation $Ax = b$, $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$ can be represented as

$$x = A^+b + (I - A^+A)y, y \in \mathbb{R}^n. \tag{4.2}$$

Next, we will show how this form could help to get the insight of linear programming.

Since all the solutions to equations $Ax = b$ are in the form (4.2), by adding another constraint that $x \geq \mathbf{0}$, we obtain the whole solution set.

We can further expand the second term $(I - A^+A)y$. Essentially it express all the vectors in the column space of $I - A^+A$. Notice that $\dim(\mathcal{R}(I - A^+A)) = n - r, r = \text{rank}(A)$. Hence, we can rewrite $(I - A^+A)y, y \in \mathbb{R}^n$ as

$$x_n = \sum_{i=1}^{n-r} \lambda_i e_i.$$

Here, $e_i, i = 1, 2, ..., n - r$ are basis of $\mathcal{R}(I - A^+A)$.

Now, consider our maximizing goal, and apply the transformation as below:

$$\begin{aligned}(x,c) &= (x,(I-A^+A)c + A^+Ac)\\ &= (x,(I-A^+A)c) + (x, A^+Ac)\\ &= (x,(I-A^+A)c) + (A^+Ax, c)\\ &= (x,(I-A^+A)c) + (A^+b, c). \end{aligned} \tag{4.3}$$

Since (A^+b, c) in (4.3) is a constant, we have only to maximize the first term:

$$\begin{aligned}(x,(I-A^+A)c) &= (x_n,(I-A^+A)c)\\ &= \sum_{i=1}^{n-r} \lambda_i (e_i, (I-A^+A)c). \end{aligned} \tag{4.4}$$

Denote $\eta = (I - A^+A)c$, and it is clear that $\eta \in \mathcal{R}(I - A^+A)$. Hence, we can choose $e_1 = \frac{\eta}{|\eta|}$, and set e_i to be the orthonormal basis for the complement space. Then, we have

$$(e_1, (I-A^+A)c) = |(I-A^+A)c|, \tag{4.5}$$

$$(e_i, (I-A^+A)c) = 0, i = 2, 3, ..., n-r. \tag{4.6}$$

Hence, we simplify the maximizing goal function into maximizing

$$\max_{\lambda_i}(x,c) = \lambda_1 |(I-A^+A)c| + (A^+b, c) \quad \text{s.t.} \quad x = A^+b + \sum_{i=1}^{n-r} \lambda_i e_i \geq \mathbf{0}. \tag{4.7}$$

We can find the optimal point by achieving the maximum λ_1.

5. **Iterated Method of Optimization**

5.1. **Iteration steps for optimization.**

Here we introduce an iterated method of solving linear programming utilizing MP-inverse. Given the instance $A \in \mathbb{R}^{m\times n}, b, c \in \mathbb{R}^n$ for LP

$$\max_x (x,c) = c^T x, \tag{5.1}$$

$$s.t.\ Ax = b, x \geq \mathbf{0}. \tag{5.2}$$

The method will be explained stepwise and we will prove the correctness of this method.

The iterated method follows five steps as below:

1. Start from an initial feasible point $x = x_0$.
2. Find all the zero entries for x. Each entry means touching the constraint with corresponding hyperplane normal vector $n_i, i = 1, 2, ..., k$.
3. Do the projection from maximum direction η to the intersection of spaces $S_i = \{x: x^T n_i \geq 0, Ax = \mathbf{0}\}, i = 1, 2, ..., k$.
4. Denote the projection of η as η^*, find the largest $\lambda \geq 0$ s.t. $x_{t+1} = x_t + \lambda\eta^* \geq 0$.
5. If the last step stucks, which means $x_{t+1} = x_t$, then stop. Otherwise, go back to step 2 and repeat.

First, we prove that such a method optimizes the position vector x_t step by step. According to our goal function, we have

$$(x_{t+1}, \eta) = (x_t + \lambda\eta^*, \eta) = (x_t, \eta) + \lambda(\eta^*, \eta) \geq (x_t, \eta).$$

From the last section we know that optimizing the original LP is equivalent to maximizing the coordinate along the direction e_1, or η. The last inequality is derived from that η^* is a projection of η, which means $\eta^* \perp (\eta - \eta^*)$. Hence, $(\eta^*, \eta) = (\eta^*, \eta^*) + (\eta^*, \eta - \eta^*) = \|\eta^*\|^2 + 0 \geq 0$. During the procedure of this method, the goal function value will not decrease. The equality holds if and only if $\eta^* = \mathbf{0}$, which means that the point is optimal.

It remains to demonstrate that the method would optimize the solution point if there exists any possible direction η^*. Moreover, if such property holds, we have to prove that the method would terminate in finite steps until it reaches the optimal solution position.

5.2. **Existence of optimization direction.** Denote all the feasible point set as $\Lambda = \{x\colon Ax = b\}$. Suppose that current position is given by x_c. A necessary and sufficient condition for x_c being the optimal solution for LP is that

$$\Lambda \cap \{x\colon (x, \eta) > (x_c, \eta)\} = \varnothing.$$

By the property that Λ is a compact and convex set, it follows that function (x, c) is continuous. If denoting the neighbourhood as $U(x_0, \varepsilon) = \{x\colon \|x - x_0\| < \varepsilon\}, \varepsilon > 0$, this condition can also be stated equally as

$$\forall \varepsilon > 0, \quad U(x_c, \varepsilon) \cap \Lambda \cap \{x\colon (x, \eta) > (x_c, \eta)\} = \varnothing.$$

The point illustrated here is that if x_c is not optimal, then just in the neighbourhood $U(x_c, \varepsilon)$ we can find a better solution. Denote that point as $x_d \in U(x_c, \varepsilon)$, and we find at least one direction $\hat{\eta} = x_d - x_c$, s.t. $\exists \lambda > 0, (x_c + \lambda\hat{\eta}, \eta) = (x_c, \eta) + \lambda(\hat{\eta}, \eta) > (x_c, \eta)$, just within the range of $\lambda\|\hat{\eta}\| \leq \varepsilon$.

5.3. **Finding optimizing direction.** The main task is to find a concrete $\hat{\eta}$. Rather than searching it in whole continuous space Λ, we propose a novel and much efficient way to search $\hat{\eta}$. We focus on its relation with constraint hyperplane's normal vectors. Denote the index for zero entries of x_c as $\{i : x_c[i] = 0\} = \{i_1, i_2, \ldots, i_s\}$. The boundary of feasible sets for $Ax = b$ is now simply hyperplanes $\{x[i] = 0 : i = 1, 2, \ldots, n\}$. Hence, the normal vectors for boundaries are simply natural coordinate basis $n_i = [0, \ldots, 1, \ldots, 0]^T, i = 1, 2, \ldots, n$ of which the only non-zero entry for n_i is $n_i[i] = 1$.

According to $x_c + \hat{\eta} \in \Lambda$, we know $\hat{\eta}$ must satisfy $\hat{\eta}^T n_{i_k} \geq 0$ for all normal vectors $n_{i_k}, k = 1, 2, \ldots, s$. Moreover, this $\hat{\eta}$ should naturally satisfies $A\hat{\eta} = \mathbf{0}$, since $A(x + \hat{\eta}) = Ax + A\hat{\eta} = A\hat{\eta} = \mathbf{0}$ is the condition that the new iteration point should still be in the feasible set. Thus, we formulate finding $\hat{\eta}$ into an optimization problem

$$\max_x \frac{x^T\eta}{\|x\|}, \quad s.t. \quad Ax = \mathbf{0}, \tag{5.3}$$

$$x^T n_{i_k} \geq 0, \quad k = 1, 2, \ldots, s. \tag{5.4}$$

It should be pointed out that this search mimic what gravitation does for a certain object. Like a ball positioned at the intersection of some hard surfaces. Once there is a way for the ball to roll down, gravitation could always manage to find a steepest direction which ball could decline. That is also the intuition for our method here. Since the ball would always finally go at the direction parallel to some neighbourhood's surface, the real search space is actually discretized rather than continuous, and this makes efficient searching possible.

Rigorously speaking, it suffices to just compute some finite number of discrete candidates and pick the best. The reason in mathematics is by the fact that apart from the trivial case where solution is $x = \eta$, we have $d(x, \eta) = \|x-\eta\|^2 = \eta^T\eta + x^T x - 2x^T\eta > 0$. Hence, maximizing the projection is equal to minimizing the distance between η and x since the norm of x does not matter and η^2 is constant. The point for transforming the problem into minimizing distance is to show that the optimal point must lie at the border of the feasible space. Suppose that the optimal x^* lies on the boarder of a subset of halfspace $\{x^T n_{i_{k'}} \geq 0, k' = 1, 2, \ldots, d\}$, namely $n_{i_{k'}}^T x^* = 0, k' = 1, 2, \ldots, d$. By Lagrangian multipliers we have

$$F(x, \mu_{k'}) = d(x, \eta) + \sum_{k'=1}^{d} \mu_{k'} n_{i_{k'}}^T x, \tag{5.5}$$

$$\frac{\partial F}{\partial x} = 2(x - \eta) + \sum_{k'=1}^{d} \mu_{k'} n_{i_{k'}} = O, \tag{5.6}$$

$$\frac{\partial F}{\partial \mu_{k'}} = n_{i_{k'}}^T x = 0. \tag{5.7}$$

Hence, we have $x^{*T}(x^* - \eta) = -\frac{1}{2}\sum_{k'=1}^{d} \mu_{k'} n_{i_{k'}}^T x^* = 0$. Equally saying, x^* is the projection onto the intersection of the space $Ax = \mathbf{0}$ and $n_{i_{k'}}^T x = 0$. The projection property makes it possible to use MP-inverse to find it quickly once given the candidate set of $n_{i_{k'}}, k' = 1, 2, \ldots, d$.

Thus, a practical way is to enumerate simply all the subsets $N' \subset N = \{n_{i_k}\}$. Denote $N' = \{v_1, v_2, ..., v_d\}$ as the selected normal vectors. These are the constraint surface that the vector is try to keep on; to compute the projection we construct the matrices:

$$U = \left(n_{i_1}, n_{i_2}, ..., n_{i_k}\right)^T, \tag{5.8}$$

$$T = \begin{pmatrix} A \\ v_1^T \\ v_2^T \\ \vdots \\ v_d^T \end{pmatrix}. \tag{5.9}$$

Given that $I - T^+T$ is a projection matrix which projects a vector onto the nullspace of matrix T. It is ensured that $\tilde{\eta} = (I - T^+T)\eta$ will be perpendicular to all the normal vectors in N', because $T\tilde{\eta} = (T - TT^+T)\eta = \mathbf{O}\eta = \mathbf{0}$. Next issue is to check whether it has non-negative inner product with all the normal vectors in N, namely, check whether

$$U\tilde{\eta} \geq \mathbf{0}. \tag{5.10}$$

If such a condition is satisfied, we successfully find a vector $\eta^* = \tilde{\eta}$ which optimize the goal function along its direction. By enumerating all the possible subsets of normal vectors in N, η^* with the largest norm is the optimal direction for x_c.

From the discussion above, we know that this method must find such an η^* if ever it exists. Specially, we know that we achieve an optimal extreme point, if and only if $I - T^+T = \mathbf{O}$ for every feasible T, hence there is no way to optimize it by finding a T such that $U\eta^* = (U - UT^+T)\eta > \mathbf{0}$.

For large scale problems, find the "steepest direction" can be formalized into a nearest point problem as below:

$$\min_y (c^T - y^T)(c - y) \quad \text{s.t.} \quad Uy \geq \mathbf{0}, \|y\|_2 = 1. \tag{5.11}$$

The intuition is that to find the optimal direction is equal to find the closest point to c on unit sphere $\|y\| = 1$ under the constraint of $Uy \geq 0$. Actually, $Uy \geq 0$ defines a relatively much simple sphere polyhedron compared with original LP. Such a problem can be solved by any classical nearest neighbour algorithm like ε-approximate nearest neighbour search *etc.* It should be emphasized that in economics practice this problem is relatively simple, because the angle between hyperplanes are obtuse, which means the normal vectors are concentrated comparatively.

5.4. **Starting point and termination.** In most practical situations, especially economic settings, b means resources or limitations. Hence, usually $b \geq \mathbf{0}$. What is more, during the transformation from inequality to equality. The right part of matrix A is an identity matrix, since there are m slack variables. Hence we can always partition A into

$$A = (M, I_m),$$

where M is an $m \times (n - m)$ matrix. Hence, a simplest initial point would be

$$x_0 = \begin{pmatrix} \mathbf{0} \\ b \end{pmatrix}.$$

To start up the process, we can even increase $x_0 + \lambda\eta, \lambda > 0$ to the utmost until it reaches the boundaries of polytope. Sometimes the initial state may be given *ex ante*, for instance current situation or investment strategy. We try to find the possibility for a better result.

Next, we will prove that this method will terminate for sure. For the sake of simplicity, denote $V^{(J)} \subset \Lambda, J = \{j_1, j_2, ..., j_d\} \subset \{1, 2, .., n\}$ as the set contains all the points x whose entries $x[j_k] = 0, k = 1, 2, ..., d$. Also, define a *potential function* $P(\mathbf{x}) = -c^T\mathbf{x}$ for all the points $x \in \Lambda$, where Λ is the feasible point set. Term *potential* is an analogy to physics concept, indicates that there will be a potential field following the gradient of $-\nabla P(\mathbf{x}) = c$. This terminology also shows the intimate connection between our method and real physical process.

A point x is called *extreme point* in non-degenerate case if there exists a set J such that $\{x\} = V^{(J)}$. This means these constraints has narrowed the feasible set down to a single point.

The LP problem is *non-degenerate* if any corresponding set J always has cardinality $|J| = \text{rank}(A)$.

Two extreme points $V^{(J)}$ and $V^{(K)}$ are called *adjacent extreme points*, if $|J \setminus K| = 1, |K \setminus J| = 1$. If two points $V^{(J)}$ and $V^{(K)}$ are adjacent extreme points, then $\{y\colon y = tV^{(J)} + (1-t)V^{(K)}, 0 < t < 1\} = V^{(J \cap K)}$ is called an *edge*.

One important lemma is that, during the iteration, there is a unique direction $\eta^*(x)$ corresponding to the current position x which records the gravitational force on that point. What our algorithm in step 3 does is exactly to find this $\eta^* = \eta^*(x)$. The maximum ratio will be

$$\lambda(x) = \max_{\lambda\eta^*(x)+x \geq \mathbf{0}} \lambda. \tag{5.12}$$

Hence, there is a function $\sigma\colon \Lambda \to \Lambda$ such that

$$\sigma(x) = x + \lambda(x)\eta^*(x). \tag{5.13}$$

Function σ characterizes the next destination of the current position. Apart from optimal point x_0, all x satisfy $\sigma(x) \neq x$ and $P(\sigma(x)) < P(x)$.

Furthermore, there must be a switch between zero entries and non-zero entries, which corresponds to transmission between extreme points and edges. Since edges and extreme points are finite and point $P(x_k)$ decrease monotonically, this iterated algorithm will not be able to repeat infinitely. Once leaving one edge or extreme point, the procedure would leave it ever since and never back. Hence, *the iteration must terminate after finite steps.*

6. **Numerical Example**

In order to show the basic procedure of solving LP, consider an empirical example below:

$$\text{Maximize } (c^T x) \qquad \text{s.t. } Ax = b.x \geq \mathbf{0}$$

Here, we assume

$$A = \begin{pmatrix} 2, & 1, & 1, & 0, & 0 \\ 1, & 2, & 0, & 1, & 0 \\ -1, & 1, & 0, & 0, & 1 \end{pmatrix}, b = \begin{pmatrix} 6 \\ 4 \\ 1 \end{pmatrix}, c = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Some simple calculations show that

$$A^+ = \frac{1}{40} \begin{pmatrix} 11, & 1, & -10 \\ 1, & 11, & 10 \\ 17, & -13, & 10 \\ -13, & 17, & -10 \\ 10, & -10, & 20 \end{pmatrix},$$

$$I - A^+A = \frac{1}{40} \begin{pmatrix} 7, & -3, & -11, & -1, & 10 \\ -3, & 7, & -1, & -11, & -10 \\ -11, & -1, & 23, & 13, & -10 \\ -1, & -11, & 13, & 23, & 10 \\ 10, & -10, & -10, & 10, & 20 \end{pmatrix}, \eta = (I - A^+A)c = \begin{pmatrix} 0.1 \\ 0.1 \\ -0.3 \\ -0.3 \\ 0 \end{pmatrix}.$$

The goal is to increase η as much as possible, since its direction correspond to the maximum optimal extreme point. Here we use *active constraints* to refer to the entries that the requirements of non-negative for whom would be violated if naively let the point go along the optimized direction. This is to distinguish between fake constraints, where the point touches the plane (as that entry is 0), but not actually hindering it from optimizing. Just like a ceiling floor does not feel the force by the ball even if it is touched. We are just like computing the joint force by surfaces and gravity now.

Our iteration proceeds as follows:

- **stage 1**
  $x_0 = (0, 0, 6, 4, 1)^T$, as the starting point
  active constraints: None.
- **stage 2**
  At this stage, since there is no active constraints, the optimal direction is just falling directly as η does.

$$\eta^* = \eta.$$

We then update our position by going as far as we can until we touch some new constraint such that a new update is needed.

$$x_1 = x_0 + \frac{40}{3}\eta^* = \left(\frac{4}{3}, \frac{4}{3}, 2, 0, 1\right)^T.$$

Now, we see if we add any positive scale of η to x_1, the fourth entry $x_1[4]$ would become negative, so that we have new active constraint as:
active constraint: $n_1 = (0,0,0,1,0)^T$.

- **stage 3**

  Now, we are hoping to find a direction which can optimize the current position with respect to the constraint requirement. Since there is only one constraint, the subset $N' = \{n_{i_1}\}$, we can construct

$$T_1 = \begin{pmatrix} A \\ n_{i_1}^T \end{pmatrix} = \begin{pmatrix} 2, & 1, & 1, & 0, & 0 \\ 1, & 2, & 0, & 1, & 0 \\ -1, & 1, & 0, & 0, & 1 \\ 0, & 0, & 0 & 1, & 0 \end{pmatrix} .$$

The reason that we put the active normal vectors to the bottom of the matrix is that, by getting projection matrix,

$$I - T_1^+ T_1 = \frac{1}{23} \begin{pmatrix} 4, & -2, & -6, & 0, & 6 \\ -2, & 1, & 3, & 0, & -3 \\ -6, & 3, & 9, & 0, & -9 \\ 0, & 0, & 0, & 0, & 0 \\ 6, & -3, & -9, & 0, & 9 \end{pmatrix} ,$$

we actually get a projector that, by multiplying any x as $(I - T_1^+ T_1)x$, we could eliminate all the component of x which belongs to T_1 row space. Namely the output $(I - T_1^+ T_1)x$ would automatically be orthogonal to all the row vectors in original T_1, including all the normal vector constraints which we want our optimizing direction to be perpendicular to. Hence, we are sure to expect that the new η^* has the zero entry at the fourth place.

$$\eta^* = (I - T_1^+ T_1)\eta = \frac{1}{23}(2,-1,-3,0,3)^T .$$

After update,

$$x_2 = x_1 + \frac{46}{3}\eta^* = \left(\frac{8}{3}, \frac{2}{3}, 0, 0, 3\right)^T .$$

Because η has negative components for both zero entries of x_2, we have active constraints:

$$n_1 = (0,0,1,0,0)^T , n_2 = (0,0,0,1,0)^T .$$

- **stage 4**
  By enumerating we know that both n_1 and n_2 have to be considered, hence

$$T_2 = \begin{pmatrix} A \\ n_1^T \\ n_2^T \end{pmatrix} = \begin{pmatrix} 2, & 1, & 1, & 0, & 0 \\ 1, & 2, & 0, & 1, & 0 \\ -1, & 1, & 0, & 0, & 1 \\ 0, & 0, & 1, & 0, & 0 \\ 0, & 0, & 0, & 1, & 0 \end{pmatrix}.$$

  Thus, we can repeat the operation as the last stage in the same manner

$$I - T_2^+ T_2 = \mathbf{O},$$

$$\eta^* = (I - T_2^+ T_2)\eta = \mathbf{O}\eta = \mathbf{0}.$$

  Algorithm stucks at same point and terminates.

Hence, we know $\left(\frac{8}{3}, \frac{2}{3}, 0, 0, 3\right)^T$ is the optimal solution, with the optimal value $\frac{8}{3} + \frac{2}{3} = \frac{10}{3}$. This proves to be the correct answer as expected.

7. **Concluding Remarks**

Although solving LP is a quite classical problem, such method still possesses significance of the two folds.

First is the coincidence of economic optimization with gravitational descent process, also the relationship between optimality of LP and economic equilibrium. Unlike other methods, the gravitational method displays a concrete incremental way to optimize investment strategy or the allocation rule. Faced with some limitations, one would seek for better possibilities to optimize its utility just regarding its local property, which is current "tight" constraints. Such greedy and short-sighted strategy for optimization, actually, leads to the optimal solution. Such a process appears an interesting resemblance from physical gravitational descent process, in which gravity is decomposed and its component along tight constraints are cancelled. We prove that this method converges to global optimal in finite steps. Moreover, the equilibrium of demand and supply, or the equilibrium of labour and capital, is exactly identical to the balance of support forces from different constraints and the gravity force from optimization direction. When such an equilibrium is reached, the pseudo inverse become *real* inverse and all the constraint hyperplanes are active, hence causes $I - T^+T = \mathbf{O}$, which is equivalent to saying that the current position is the optimal solution.

Second importance is that, unlike general mathematical LP problems, there are several special conditions for LP in economics. The counterexample by Morin(2001) that the gravitational method degenerates to exponential times of iteration seems to be unrealistic in economics, since all the variables involved are very unlikely to be of geometry distribution but rather on the similar magnitude. Also, the core difficulty of finding next descend direction on each iteration is easy in economic scenario, because there is few "sharp angles" in polytope. Hence, the normal vectors of a corner is confined into a small conical area, making finding η^* much easier than general cases. Sometimes we may already be at a reasonable point, so that we need not consider all the constraints and start from none to optimize the solution vector. That is also the reason why the gravitational method matters here.

In economic theory, Fujimori(1982) showed interests in applying generalised inverses to multi-sector economic models. Li-Fujimori(2013) and Li(2017), however, developed the spectral analysis to investigate equilibria of joint-production systems. Their contributions will be combined with the theory of this article elsewhere in near future.

Appendix A. Python Script for Numerical Example

The script used in this paper by Python 3.7 on Windows10 and/or OS X10.13 is as follows:

```
# PM-LP.py ver. 1.0
# Copywrite Lu Rui, 2019.10.02
# Run this on your shell prompt:  python PM-LP.py

from numpy import *

def incre(v, delta):
    bottle = 1e4
    for i in range(len(v)):
      if delta[i]< -1e-6 and -v[i]/delta[i]<bottle:
          bottle = -v[i] / delta[i]
    return v + bottle * delta

def getp(K, x):
    global A
    d = len(x); m = len(K)
    maxim = -1e-6
    ans = zeros(d)
    for cnt in range(1, 2**m):
     tmp = cnt; M = []
     for i in range(m):
         if tmp % 2:
                M.append(K[i])
         tmp = tmp // 2
     M = concatenate((M,A), axis=0)
     tri = dot(eye(d)-dot(linalg.pinv(M), M), x)
     if min(dot(K,tri)) > -1e-6 and dot(tri,x)>maxim:
           maxim = dot(tri, x)
           ans = tri
    return ans

set_printoptions(precision = 4, suppress = True)
m = 3; n = 5
A = [[2, 1, 1, 0, 0], [1, 2, 0, 1, 0],
     [-1, 1, 0, 0, 1] ]
b = [6, 4, 1]
c = [1, 1, 0, 0, 0]

#Computation section:
A_ = linalg.pinv(A)
E = eye(n) - dot(A_, A)
x_p = zeros(n); x_p[n-m:] = b
maxdir = dot(E,c)
print('maximum direction:',maxdir)
x_p = incre(x_p, maxdir)
print('starting point:',x_p)

cnt = 0; bnd = ones(n)

while max(fabs(bnd)) > 1e-6:
    cnt += 1
    print(str(cnt)+' iteration')
    K = []
    for i in range(n):
        if fabs(x_p[i]) < 1e-6:
            tmp = zeros(n); tmp[i] = 1
            K.append(tmp)
    print('constraint matrix:')
    print(K)

    bnd = getp(K, maxdir)
    print('direction:')
    print(bnd)
    x_p = incre(x_p, bnd)
    print("new point:",x_p)
    print()

print('optimal point:',x_p)
print('optimal value',dot(c,x_p))
# End of the script.
```

Bibliography

[1] Ben-Israel, and Greville, T. (2003), *Generalized Inverse*: *Theory and Applications* 2nd ed., Springer.

[2] Campbell, and Meyer, C.D. (2009), *Generalized Inverse of Linear Transformations*, SIAM.

[3] Chang, S. Y. and Murty, K. G. (1988), “The steepest descent gravitational method for linear programming”, *Discrete Applied Mathematics* 25(3), pp.211-39.

[4] Fujimori, Y. (1982), *Modern Analysis of Value Theory*, Springer.

[5] Greenberg, H. J. (1997), “Klee-minty polytope shows exponential time complexity of simplex method”, Mimeo.

[6] LI, Bangxi(2017), *Linear Theory of Fixed Capital and China’s Economy*: *Marx, Sraffa and Okishio*, Springer-Nature.

[7] LI, Bangxi and Yoriaki Fujimori(2013), “Fixed Capital, Renewal Dynamics and Marx-Sraffa Equilibrium”, in Kasamatsu, M. (ed), *Macro- and Microfoundations of Economics*, pp. 51-72, Waseda University Press.

[8] Meyer, C. D. (2000), *Matrix Analysis and Applied Linear Algebra*, SIAM.

[9] Morin, T. L., Prabhu, N., and Zhang, Z. (2001), “Complexity of the gravitational method for linear programming”, *Journal of Optimization Theory and Applications* 108(3), pp.633-58.

[10] Pyle, L. D. (1967), “A generalized inverse ε-algorithm for constructing intersection projection matrices with applications”, *Numerische Mathematik* 10(4), ss. 86-102.

[11] Pyle, L. D. (1972), “The generalized inverse in linear programming basic structure”, *SIAM Journal on Applied Mathematics* 22(3), pp.335-55.

[12] Pyle, L. D. and Cline, R. E. (1973), “The generalized inverse in linear programming–interior gradient projection methods”, *SIAM Journal on Applied Mathematics* 24(4), pp. 511-34.

[13] Rao, C. R. and Mitra, S. K. (1971), *Generalized Inverse of Matrics and Its Applications*, John Wiley & Sons.

[14] van Rossum, Guido (2016), *Python Tutorial* 3rd ed., Python Software Foundation.

Author(s)

LI Bangxi, Associate Professor Dr. of Economic Theory, Institute of Economics, School of Social Sciences, Tsinghua University, Beijing, China,
LU Rui, Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China (Corresponding author), Email: `lu233rui@gmail.com`
ZHAO Yihan, Graduate School, Institute of Economics, School of Social Sciences, Tsinghua University, Beijing, China (co-corresponding author). Email: `zyh18@mails.tsinghua.edu.cn`
Yoriaki Fujimori, Professor Emeritus, Waseda University, Tokyo, Japan.

Acknowledgements: This paper is supported by The National Social Science Foundation of China (No.17BJL020). The authors should express their thanks to helpful comments and suggestions to the editors of the journal. Usual caveats apply.