Proc. Natl. Acad. Sci. USA
Vol. 91, pp. 7169–7173, July 1994
Economic Sciences

A decentralized process for finding equilibria given by linear equations

STANLEY REITER

Center For Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, IL 60208

Communicated by Leonid Hurwicz, April 11, 1994

ABSTRACT I present a decentralized process for finding the equilibria of an economy characterized by a finite number of linear equilibrium conditions. The process finds all equilibria or, if there are none, reports that, in a finite number of steps at most equal to the number of equations. The communication and computational complexity compare favorably with other decentralized processes. The process may also be interpreted as an algorithm for solving a distributed system of linear equations. Comparisons with the Linpack program for LU (lower and upper triangular decomposition of the matrix of the equation system, a version of Gaussian elimination) are presented.

This paper presents a decentralized process for finding the equilibria of an economy characterized by a finite number of linear equilibrium conditions. The process differs from others that have been studied in connection with stability (global and local) of equilibrium in that it is given by an algorithm requiring a finite number of steps, rather than an infinite iterative process, as, for instance, a process given by difference or differential equations.

Furthermore, it is not assumed that equilibria exist. If there are no equilibria, the process discovers this and stops; if there are equilibria, the process finds them all in a fixed number of steps depending on the number of equilibrium conditions.

The process is decentralized in the following sense. In a certain subset of economies with N agents, the equilibrium condition of an individual agent is a linear equation in R^N (possibly several linear equations)[†] with $N + 1$ coefficients, not all zero.[‡] That equation (i.e., its coefficients) need be known only to the agent whose equilibrium condition it is. The process requires each agent to transmit a message consisting of a certain finite number of points of R^N to one other agent. When $N > 2$, the messages transmitted are insufficient for any agent to learn the complete system of equilibrium equations.[§] At each step of the process one agent decides on a message by using only knowledge of his/her own equilibrium condition and the message received from one other agent.

Literature. The problem of finding equilibrium is related to and stems from previous work on the stability of equilibrium (1–13). [Work on computation of equilibria should also be mentioned, although that deals with nonlinear equilibrium conditions and does not concern itself with decentralized methods (14).] For the present purpose the literature on stability of equilibrium may be summarized as follows. Papers (1–3) studied price adjustment processes in which prices change according to excess demand. The objective was to show that every economic environment that satisfies conditions ensuring the existence of competitive equilibria has (globally or locally) stable equilibria. This turned out not to be the case (4, 5). To guarantee stability, additional conditions, such as "gross substitutes," are required. Smale (6) studied an adjustment process that involved much bigger messages than just the sign, or value, of excess demand. He showed that price equilibria could be stabilized, provided the process starts sufficiently near the boundary of the price simplex, by using the derivatives of the demand functions. Saari and Simon (7) showed that the number of variables used in the global Newton process cannot be significantly reduced without losing stability. Previously, I (8) analyzed an example consisting of a pure exchange economy with two goods and two agents and quasilinear utilities (hence, linear individual equilibrium conditions) and showed that there is no decentralized message exchange process given by difference equations with a nonvanishing linear part, whose message space is the same size as that of the competitive mechanism, that can stabilize the (unique) static equilibrium for all specifications of the agents' parameters. Jordan (9) introduced a new and broader class of decentralized dynamic processes, given by differential equations, and showed that even in that class the informational requirements of local stability are larger than what is needed for decentralized verification of equilibrium. Papers (10, 11) differ from Jordan's in particular specifications but come to the same general conclusion—namely, that requiring local stability of equilibrium by a decentralized message exchange process entails larger informational requirements than does the decentralized verification of equilibrium. The literature shows that to guarantee local stability of equilibrium for all environments for which equilibria exist entails higher informational costs. The difficulties exposed in that literature are already present even in the case of linear individual equilibrium conditions.

Motivation and Summary. From the standpoint of mechanism design, both the amount of communication required and the complexity of calculations performed by economic agents are important attributes of a mechanism for finding equilibrium. In ref. 13 a class of processes is presented in which agents are asked to respond to messages only by indicating whether the proposal encoded in the message is "too big" or "too small." Those processes augment with one bit the messages required for decentralized verification of equilibrium and use an algorithm based on the idea of bisection. Those processes converge in the limit when individual equilibrium conditions are linear. However, while the message space used by these processes is small, perhaps minimal, the complexity is exponential in the number of agents. The motivation for the process presented in this paper is to

The publication costs of this article were defrayed in part by page charge payment. This article must therefore be hereby marked "*advertisement*" in accordance with 18 U.S.C. §1734 solely to indicate this fact.

[†]If some agents have more than one equation and there are N agents, then the space will be R^M, $M > N$. For simplicity we assume that each agent's equilibrium condition is given by one equation.

[‡]The *Appendix* presents an example of a two-person, two-commodity exchange economy with a decentralized message exchange process whose equilibria realize the Walrasian allocation. The equilibria of the message exchange process are given by linear individual equilibrium equations.

[§]Agent 2 receives enough information to deduce the coefficients of the first equation up to a factor of proportionality. However, agent 2 learns nothing of subsequent equations, if any. Hence, as is seen below, $N = 2$ is the sole exception.

preserve as much as possible the low communication requirements of the binary bisection processes in ref. 13, while reducing complexity from exponential to polynomial.

The process presented below achieves polynomial (in the number of equations) complexity, but at the cost of increasing the messages used by the process from one binary bit to one real number. The complexity of the calculations required of each agent is linear in the number, N, of equations, the total complexity (as if one agent performed the entire computation) is of order N^3, and convergence occurs in, at most, N steps.

In addition, the process allows at low cost the exploration of the effects of policy interventions that change coefficients of some of the equations of the system.

The remainder of this paper is organized as follows. The process is defined formally and it is shown that (*a*) it applies to any system of N linear equations; (*b*) if the equations have solutions, it finds all of them in N steps; and (*c*) if there are no solutions, the process discovers this in, at most, N steps.

The results of trial computations are discussed in which the process is viewed as an algorithm for solving a general system of linear equations. These are compared with the performance of Linpack's[¶] implementation of LU (lower and upper triangular decomposition of the matrix of the equation system, a version of Guassian elimination). Linpack's program, a highly refined Fortran program, seems preferable for many but not all problems. However, the present algorithm is decentralized whereas Linpack's is not.[||] Linpack's LU is slightly faster, slightly more accurate, and with pivoting quite stable. The present program seems to have an advantage in situations in which a class of related problems is to be solved in which there is a block of equations common to all problems of the class.

Notation. Suppose there are N agents, each with a linear equilibrium condition. Let the equilibrium condition of agent i be

$$a^i x = c^i \qquad a^i, x \in R^N, c^i \in R, \qquad i = 1, \ldots, N, \tag{1}$$

where $a^i x$ is the inner product of a^i and x. The solutions of the equation system **1** are called equilibria. For $i = 1, \ldots, N$ define

$$g^i(x) = a^i x - c^i. \tag{2}$$

Then the equation system **1** can also be written vectorially as $g(x) = 0$, or in matrix notation as $Ax = c$, where A is the coefficient matrix $((a_{i,j}))$ and c is the vector of constant terms.

We assume that agent i knows the function g^i, and hence his/her own equilibrium condition, but does not know g^j for $j \neq i$. Thus, the system of equations **1** is distributed among the agents.

We do not assume that an equilibrium exists, or if one does exist, that it is unique.

The equation system **1** can also be regarded as a distributed system in which the equations are distributed among processors; if fully distributed then each processor has in its memory the $N + 1$ coefficients of one and only one equation.

In this interpretation the process is seen as a distributed algorithm for solving the equation system **1**.

The Algorithm (Informal). The basic idea of the process or algorithm can be described in intuitive geometric terms. This is easiest in the case where A is nonsingular, and there are no other nongeneric special cases, so this assumption is maintained throughout this subsection. A more formal and complete specification of the algorithm follows.

Each equation i of system **1** defines a hyperplane, H_i in R^N, provided $a^i \neq 0$. The agents (processors) are ordered 1, 2, . . . , N. The process goes in steps from agent 1 to agent N. First, agent 1 finds N points that satisfy equation 1; that is, agent 1 finds N points in H_1. Agent 1 transmits those points to agent 2, who uses them to find $N - 1$ points that satisfy both equations 1 and 2—i.e., that lie in $H_1 \cap H_2$. These $N - 1$ points are communicated to agent 3, who uses them to find $N - 2$ points in $H_1 \cap H_2 \cap H_3$, and so on until agent N uses the two points transmitted from agent $N - 1$ to find one point, the solution, in $H_1 \cap \ldots \cap H_N$.

The process of finding these points is as follows. Agent 1 uses the standard coordinate axes in R^N, each of which can be characterized as the line generated by two points, one of which is the origin and the other a point all of whose coordinates but one are 0 and the remaining coordinate is 1. For each of these N lines, agent 1 finds the point at which it intersects H_1. (Generally there will be such a point. Special cases in which such a point does not exist are treated in detail below.) These N points are affinely independent, because the lines that generated them are linearly independent. They are the points that agent 1 communicates to agent 2.

Agent 2 uses the N points received from agent 1 to generate $N - 1$ lines, by selecting one of these points to be common to all the lines and using each of the remaining $N - 1$ points communicated by agent 1 to generate each of the $N - 1$ lines.

Because the $N - 1$ points sent by agent 1 lie in H_1, so do the lines connecting pairs of them. Having calculated these lines, agent 2 then finds the $N - 1$ points at which those lines intersect H_2. The points of intersection exist generically. These $N - 1$ points lie in $H_1 \cap H_2$, because the lines used to generate them lie in H_1, and the points chosen on those lines also lie in H_2. This process continues until agent N finally finds the point in $H_1 \cap \ldots \cap H_N$.

In economic models the equilibrium condition of an agent often consists of several equations. In that case each agent will use the algorithm, starting from the points received from the preceding agent and find the requisite number of points satisfying his/her equations in sequence, one equation at a time.

The various nongeneric special cases that can arise, and the modifications needed to make the process deal with them are described in the formal presentation.

The Algorithm (Formal). The process or algorithm proceeds equation by equation in steps from equation 1 to equation N of system **1**.

Step 1: Agent 1 begins with $N + 1$ points of R^N denoted $V^0 = \{v_0^0, v_1^0, \ldots, v_N^0\}$, where v_j^0 is the jth standard basis vector, and v_0^0 is the null vector. These $N + 1$ points are affinely independent,[**] or equivalently, the N vectors, $\mathbf{\bar{V}}^0 = \{\mathbf{v}_j^0 - \mathbf{v}_0^0, j = 1, \ldots, N\}$, are linearly independent.

Agent 1 uses these $N + 1$ points to find N points that satisfy equation 1 of system **1**. We make use of certain auxiliary results (*Propositions 1–4*).

PROPOSITION 1. *There is at least one point* v_i^0 *in* V^0 *such that* $\mathrm{g}^1(\mathrm{v}_i^0) \neq 0$.

[¶]See footnote[††].

[||]The forward step of Gaussian elimination involves operations on every equation below the one under consideration. Pivoting, essential for numerical stability of Gaussian elimination, requires at each step the calculation of the maximum of certain coefficients over the set of equations below the current candidate for the pivot. While it is possible to perform these operations in a distributed fashion—i.e., where each equation is stored in a separate processor—the amount of communication required to do so would be so large as to negate many of the attractive features of the algorithm.

[**]A collection of r points in R^N, $r \geq N$, is an affinely independent set, if the smallest linear manifold that contains them has dimension r. The smallest linear manifold that contains the points is the intersection of all linear manifolds that contain them.

Proof: Otherwise V^0 would contain $N + 1$ affinely independent points, each an element of the $(N - 1)$-dimensional hyperplane H_1 defined by the equation $g^1(x) = 0$, which is impossible. q.e.d.

Suppose, without loss of generality (the points can be relabeled), that $g^1(v_0^0) \neq 0$. Agent 1 forms the lines

$$L(v_0^0, v_j^0) = \{x \in R^N | x = v_0^0 + \lambda_j^0(v_j^0 - v_0^0),\ \lambda_j^0 \in R\},$$

$$\text{for } j \in \{1, \ldots, N\}.$$

For each $j \in \{1, \ldots, N\}$ the line $L(v_0^0, v_j^0)$ either intersects the hyperplane H_1 given by $g^1(x) = 0$ or is parallel to it. In the first case, $g^1(v_j^0) - g^1(v_0^0) \neq 0$, and in the second case, $g^1(v_j^0) - g^1(v_0^0) = 0$. For $j \in \{1, \ldots, N\}$, define

$$\hat{\lambda}_j^0 = \frac{-g^1(v_0^0)}{g^1(v_j^0) - g^1(v_0^0)} \qquad \text{if } g^1(v_j^0) - g^1(v_0^0) \neq 0$$

and otherwise let $\hat{\lambda}_j^0$ be undefined. It is easy to see the following.

PROPOSITION 2. *There is at least one point* v_j^0 *in* V^0, *other than* v_0^0, *such that* (∗) $g^1(v_j^0) - g^1(v_0^0) \neq 0$.

Proof: If not, then there exists a scalar k such that for all $j \in \{1, \ldots, N\}$, $g^1(v_j^0) = g^1(v_0^0) = k$, and hence for all $j \in \{0, 1, \ldots, N\}$, $g^1(v_j^0) = k$. Thus, $N + 1$ affinely independent points lie in an $(N - 1)$-dimensional hyperplane, which is impossible. q.e.d.

Let v_1^0 be one of the points for which (∗) holds. Then we may define $v_1^1 = v_0^0 + \hat{\lambda}_1^1(v_1^0 - v_0^0)$, and for $j \in \{2, \ldots, N\}$ define

$$v_j^1 = \begin{cases} v_0^0 + \hat{\lambda}_j^1(v_j^0 - v_0^0) & \text{if } \hat{\lambda}_j^1 \text{ is defined,} \\ v_1^1 + (v_j^0 - v_0^0) & \text{otherwise.} \end{cases} \qquad [3]$$

PROPOSITION 3. *The points in* $V^1 = \{v_1^1, \ldots, v_N^1\}$ *satisfy equation 1 of system* **1**.

Proof: (*i*) Suppose j is a value for which $g^1(v_j^0) - g^1(v_0^0) \neq 0$. Equation 1 of system **1**, $a^1v_j^1 - c^1 = 0$, is equivalent to

$$0 = a^1[v_0^0 + \hat{\lambda}_j^1(v_j^0 - v_0^0)] - c^1 = a^1v_0^0 + \hat{\lambda}_j^1a^1(v_j^0 - v_0^0) - c^1,$$

that is, to

$$\hat{\lambda}_j^1 = \frac{c^1 - a^1v_0^0}{a^1(v_j^0 - v_0^0)} = \frac{-g^1(v_0^0)}{g^1(v_j^0) - g^1(v_0^0)},$$

which holds by construction. (*ii*) On the other hand, if j is a value for which $g^1(v_j^0) - g^1(v_0^0) = 0$, then

$$\begin{aligned} a^1v_j^1 - c^1 &= a^1(v_1^1 + v_j^0 - v_0^0) - c^1 \\ &= a^1v_1^1 - c^1 + a^1v_j^0 - c^1 - a^1v_0^0 + c^1 \\ &= g^1(v_1^1) + [g^1(v_j^0) - g^1(v_0^0)] \\ &= 0, \end{aligned}$$

because $g^1(v_1^1) = 0$ by part *i*, and $g^1(v_j^0) - g^1(v_0^0) = 0$ by hypothesis. Hence, $a^1v_j^1 - c^1 = g^1(v_j^1) = 0$. q.e.d.

It is also straightforward to show that the points in $V^1 = \{v_1^1, \ldots, v_N^1\}$ are affinely independent. However, in order to use the same argument in subsequent steps of the process we state the more general proposition as follows.

PROPOSITION 4. *If the points in* $V' = \{v_0', \ldots, v_q'\}$ *are affinely independent, and the points in* $V'' = \{v_1'', \ldots, v_q''\}$ *are generated by Eqs. 3, i.e.,*

$$v_j'' = \begin{cases} v_0' + \hat{\lambda}_j^k(v_j' - v_0') & \text{if } g^k(v_j') - g^k(v_0') \neq 0, \\ v_1'' + (v_j' - v_0') & \text{otherwise,} \end{cases}$$

then for some equation g^k, $0 \leq k \leq N$, *the points of* V'' *are affinely independent.*

Proof: Suppose not. Then there exist scalars $\theta_j, j = 2, \ldots, N$, not all zero, such that $\theta_2(v_2'' - v_1'') + \ldots + \theta_q(v_q'' - v_1'') = 0$. But

$$\begin{aligned} 0 &= \theta_2(v_2'' - v_1'') + \ldots + \theta_q(v_q'' - v_1'') \\ &= \theta_2\lambda_2^*(v_2' - v_0') + \ldots + \theta_q\lambda_q^*(v_q' - v_0') \\ &\quad - \sum_{j=2}^{q} \theta_j\hat{\lambda}_1^k(v_1' - v_0'), \end{aligned}$$

where we define

$$\lambda_j^* = \begin{cases} \hat{\lambda}_j^k & \text{if } g^k(v_j') - g^k(v_0') \neq 0, \\ 1 & \text{otherwise.} \end{cases}$$

Note that for all $j = 2, \ldots, q$, $\lambda_j^* \neq 0$, and by construction, $\hat{\lambda}_1^k \neq 0$. Since the scalars $\theta_j, j = 2, \ldots, q$, are not all zero, it follows that the vectors are linearly dependent, contrary to the hypothesis that the points in V' are affinely independent. q.e.d.

Proposition 4, together with the fact that the points in V^0 are affinely independent, establishes that the points in V^1 are affinely independent.

Thus, at the end of step 1, agent 1 has found N affinely independent points that satisfy equation 1 of system **1**. The set V^1 consisting of these points is the starting state for step 2. The starting state for step s (if Step s is reached) is a set V^{s-1} consisting of p_{s-1} points satisfying equations 1, 2, . . . , $s - 1$, where p_{s-1} is a positive integer at most equal to N.

Step s: Note first that *Proposition 1* need not hold for $s \geq 2$. Step s begins with the set V^{s-1} consisting of $N - p_{s-1} + 1$ affinely independent points that satisfy the first $s - 1$ equations of system **1**. If the points in V^{s-1} also satisfy equation s—i.e., if $g^s(v_j^{s-1}) = 0$, for $j = p_{s-1}, \ldots, N$—then $V^s = V^{s-1}$ and the process goes to step $s + 1$.

If, on the other hand, in step s there is some element v_q^{s-1} in V^{s-1}, say, without loss of generality, $v_{p_{s-1}}^{s-1}$, such that $g^s(v_{p_{s-1}}^{s-1}) \neq 0$, and if $g^s(v_j^{s-1}) - g^s(v_{p_{s-1}}^{s-1}) = 0$ for all $j = p_{s-1}, \ldots, N$, then equation s is inconsistent with the preceding $s - 1$ equations, and the system **1** has no solution.

If, however, this is not the case—i.e., if $g^s(v_{p_{s-1}}^{s-1}) \neq 0$ and, for some j, $g^s(v_j^{s-1}) - g^s(v_{p_{s-1}}^{s-1}) \neq 0$, then, for $j = p_s, \ldots, N$, $\hat{\lambda}_j^s$ is defined by

$$\hat{\lambda}_j^s = \frac{-g^s(v_{p_{s-1}}^{s-1})}{g^s(v_j^{s-1}) - g^s(v_{p_{s-1}}^{s-1})} \qquad \text{if } g^s(v_j^{s-1}) - g^s(v_{p_{s-1}}^{s-1}) \neq 0,$$

and is otherwise not defined. Furthermore,

$$v_j^s = \begin{cases} v_{p_{s-1}}^{s-1} + \hat{\lambda}_j^s(v_j^{s-1} - v_{p_{s-1}}^{s-1}) & \text{if } \hat{\lambda}_j^s \text{ is defined,} \\ v_{p_s}^s + (v_j^{s-1} - v_{p_{s-1}}^{s-1}) & \text{otherwise.} \end{cases}$$

The counterparts of *Propositions 3* and *4* show that $V^s = \{v_{p_s}^s, \ldots, v_N^s\}$ consists of $N - p_s + 1$ affinely independent points that satisfy the first s equations, where

$$p_s = \begin{cases} p_{s-1} & \text{if } V^s = V^{s-1} \\ p_{s-1} + 1 & \text{otherwise.} \end{cases}$$

In applying *Proposition 4*, the set V' is replaced by V^{s-1} and V'' by V^s.

The process ends in one of two states:

- Equation $s \leq N$ is the first equation that is inconsistent with the preceding ones, in which case the process stops in step s; if equation N is the first equation that is inconsistent with the preceding ones, then the process ends in step N (i.e., step N is not completed) and notes the inconsistency—i.e., V^N is empty.
- The process continues through step N, with V^N not empty, in which case V^N spans the $(N - p_N)$-dimensional linear manifold of solutions.

If the matrix A of system 1 is of full rank, then $p_s = s$ for all $s = 1, \ldots, N$. In particular, $p_N = N$, and the solution manifold consists of one point.

Communication and Complexity. The communication and computational complexity properties of this process are as follows. (All calculations are made for the nonsingular case, so that $p_s = s$; when each agent has only one equilibrium equation, as has been assumed, we can write $s = i$, and hence $p_s = p_i = i$.)

(*a*) The messages used by the process consist of finite subsets of R^N, which is the space in which equilibrium resides. The first agent transmits N points, the second $N - 1$, and so on until the last agent transmits one point, the solution. Thus, the total number of points of R^N communicated is at most $\Sigma_{k=1}^{N} k = N(N + 1)/2$, which is of order N^2. In the "normal" case (i.e., without singularities or other nongeneric special cases), the nonzero coordinates in the points transmitted by agent i occur in a known pattern, and their number is i. Hence the number of coordinates communicated by agent i is $i(N - i + 1)$, which is linear in N. As i varies from 1 to N, the number of coordinates transmitted varies from N to $N^2/4$ [or $(N + 1)^2/4$, depending on whether N is even or odd] and then back to N.

Note that while agent 2 receives enough information to allow him/her to identify the equation of the first agent, agent 2 learns nothing about the equilibrium conditions of subsequent agents, and no agent after agent 2 receives enough information to identify any equation other than his/her own.

(*b*) The complexity [the number of floating point operations (flops)] of agent i's computation is, for $i = 2, \ldots, N$, $2(2i + 1)N - 4i(i - 1)$, which is, for each i, linear in N. (For $i = 1$ the complexity is N.) The agent whose computations have maximum complexity is $i = N/2$ [or $(N + 1)/2$, depending on whether N is even or odd], and the complexity of the most complex computation is of order N^2.

The total number of flops performed by all agents is of order $(2/3)N^3$; if only multiplications are counted, it is of order $(1/3)N^3$.

(*c*) Each agent i must have enough memory to store at most $N + 1$ coefficients of his/her equation, and, of course, sufficient computational capacity to carry out the simple arithmetic operations needed to evaluate his/her linear function and to calculate the points in V^i. We should also mention the possibility that a specialized network of low capacity (cheap) processors might be used to solve distributed problems in an on-line fashion.

Comparing the communication and computational complexity of this algorithmic process with the one(s) presented in ref. 13, we see that the increase in message size per agent from bits to real numbers is accompanied by a decrease in both communication and computational complexity per agent from exponential to polynomial, and a reduction in the number of steps corresponding to the replacement of a process that converges only in the limit to one that requires only a finite number of steps.

Trial Computations. We may suppose that all the equations are in the memory of a single processor, and view this process as an algorithm for solving a general system of linear equations. A Fortran implementation of this algorithm[††] was used to run trial computations and to compare its performance with that of the Linpack implementation of LU (referring to lower and upper triangular decomposition of the matrix A), a version of Gaussian elimination, also a Fortran program.[‡‡] The results of trials on a set of randomly generated matrices, ranging in size up to $N = 1200$, are summarized as follows. Linpack is slightly faster; Linpack is about an order of magnitude more accurate—i.e., if Linpack is accurate to 20 decimal places, the present algorithm is accurate to about 19; both algorithms were stable in all problems run.

While a zero denominator in the formula for one of the multipliers λ_j^s does not present any difficulty, a sufficiently small denominator might create instability for the algorithm presented here. This can be dealt with by using a procedure analogous to pivoting to select the common point in step s, denoted v_{s-1}^{s-1} above, so as to ensure that the denominator is not too small. As might be expected, this problem did not appear in the trial computations with random matrices but can be expected to slow the computations in difficult cases.

The Linpack program for LU is highly efficient. It is the result of many years of experience with that algorithm. It remains to be seen whether the program we now have for the algorithm presented here might still have room for improvement.

In any case, there is a class of problems for which the present algorithm appears to have an advantage, aside from being decentralized. In many situations in which related systems of linear equations are to be solved, the systems have a block of equations in common and only a subset of equations varies from problem to problem. In this case, the present algorithm would solve the common equations once, finding a set of points spanning the solution manifold for those equations, and use this set as the starting points for the remaining equations. For all but the first problem, this effectively reduces the size of the problem to be solved to the size of the block of equations that differ among problems.

This property should be useful in comparative static, or sensitivity, analyses in which the equations can be ordered so that parameter changes effect only a final subset of equations.

Appendix. The following is an example of an economy and a decentralized mechanism given by a system of linear equations. The outcome resulting from an equilibrium of the mechanism for a given set of parameters characterizing the economic agents (the environment) is the Walrasian trade vector for that environment.

There are two agents, 1 and 2, and two commodities, X and Y. Let the initial endowments be w_X^i and w_Y^i, $i = 1, 2$; and let the utility functions be

$$u^i(X, Y) = \alpha^i Y^i + 1/2\beta^i(Y^i)^2 + X^i, \qquad i = 1, 2.$$

To express the model in terms of net trades, let $y^i = Y^i - w_Y^i$; $x^i = X^i - w_X^i$, $i = 1, 2$; $y^1 + y^2 = 0$. The budget constraint of agent i is, taking X as numeraire, $x^i + py^i = 0$. Note that if y^i and p are known, then x^i is determined. Therefore it suffices to determine y^i and p. First-order conditions are

$$\frac{\partial u^i}{\partial y^i} = \alpha^i + \beta^i(y^i + w_Y^i) + p = 0, \qquad i = 1, 2.$$

Let $y \equiv y^1$; then from the feasibility constraint on trades, $-y = y^2$, which on substitution into the first-order conditions

[††]The program was written by Frederick D. Dean, who also carried out the trials. The program is available on request.

[‡‡]The Linpack LU program can be obtained by e-mail from ⟨netlib@research.att.com⟩. The "Linpack Users' Guide," by J. J. Dongarra, C. B. Moler, J. R. Bunch, & G. W. Stewart, was published in 1979 by SIAM, Philadelphia.

gives $\alpha^1 + \beta^1(y + w_Y^1) + p = 0$ and $\alpha^2 + \beta^2(-y + w_Y^2) + p = 0$. Let $\gamma^i = \alpha^i + \beta^i w_Y^i$, $i = 1, 2$. Then further substitution in the first-order conditions yields $\gamma^1 + \beta^1 y + p = 0$, and $\gamma^2 - \beta^2 y + p = 0$. Eliminating p yields the equation $\gamma^2 - \beta^2 y = \gamma^1 + \beta^1 y$. For $\theta^i = (\theta_1^i, \theta_2^i)$, define $\theta^1 = (\gamma^1, \beta^1)$ and $\theta^2 = (\gamma^2, -\beta^2)$. Then the first-order condition is $\theta_1^2 + \theta_2^2 y = \theta_1^1 + \theta_2^1 y$. Therefore,

$$y = \frac{\theta_1^1 - \theta_1^2}{\theta_2^2 - \theta_2^1}, \quad \text{provided } \theta_2^2 - \theta_2^1 \neq 0. \qquad \textbf{[A1]}$$

Eq. **A1** expresses the "optimal" or desired value of y (the Walrasian trade) as a function of the parameters.

A decentralized message-exchange process whose equilibrium yields the desired value of y—i.e., the one given by Eq. **A1**—is as follows. Let the messages be (m_1, m_2), and let the individual equilibrium condition of agent i, for $i = 1, 2$, be

$$\theta_1^1 + \theta_2^1 m_2 - m_1 = 0, \quad \text{and} \quad \theta_1^2 + \theta_2^2 m_2 - m_1 = 0. \qquad \textbf{[A2]}$$

Solving for m_1 and m_2, we find that

$$m_1 = \frac{\theta_1^1\theta_2^2 - \theta_2^1\theta_1^2}{\theta_2^2 - \theta_2^1} \quad \text{and} \quad m_2 = \frac{\theta_1^1 - \theta_1^2}{\theta_2^2 - \theta_2^1}.$$

We see that $m_2 = y$, the desired trade in the commodity Y, as a function of the parameters, and m_1 is the price.

The equation system **A2** has the property that the ith equation contains only parameters of agent i, for $i = 1, 2$, and the equilibrium message equations are, of course, linear in m_1 and m_2.

To complete the specification of the mechanism, the outcome function is the projection into the space of m_2; i.e., $h(m_1, m_2) = m_2$.

I am indebted to Leo Hurwicz and Carl Simon for helpful comments. Scott E. Page wrote a Pascal program implementing the process presented here, and Frederick D. Dean, a graduate student in Computer Science at Northwestern University, wrote the Fortran program used in the computational trials reported. Neither of them is responsible for any errors that remain. This research was supported by National Science Foundation Grant IRI-9020270.

1. Samuelson, P. A. (1948) *Foundations of Economic Analysis* (Harvard Univ. Press, Cambridge, MA).
2. Arrow, K. J. & Hurwicz, L. (1958) *Econometrica* **26,** 522–552.
3. Arrow, K. J., Block, H. D. & Hurwicz, L. (1959) *Econometrica* **27,** 82–109.
4. Scarf, H. (1960) *Int. Econ. Rev.* **1,** 157–172.
5. Gale, D. (1963) *Nav. Res. Logist. Q.* **10,** 81–87.
6. Smale, S. (1976) *J. Math. Econ.* **3,** 107–120.
7. Saari, D. G. & Simon, C. P. (1978) *Econometrica* **46,** 1097–1125.
8. Reiter, S. (1979) (Northwestern Univ., Evanston, IL), preprint.
9. Jordan, J. S. (1987) in *Information, Incentives and Economic Mechanisms, Essays in Honor of Leonid Hurwicz,* eds. Groves, T., Radner, R. & Reiter, S. (Univ. of Minnesota Press, Minneapolis), pp. 183–212.
10. Mount, K. R. & Reiter, S. (1987) in *Information, Incentives and Economic Mechanisms, Essays in Honor of Leonid Hurwicz,* eds. Groves, T., Radner, R. & Reiter, S. (Univ. of Minnesota Press, Minneapolis), pp. 213–240.
11. Williams, S. R. (1988) *J. Econ. Theory* **35,** 127–154.
12. Saari, D. G. & Williams, S. R. (1986) *J. Econ. Theory* **40,** 152–167.
13. Reiter, S. & Simon, C. P. (1992) *J. Econ. Theory* **56,** 400–425.
14. Scarf, H. & Hansen, T. (1973) *The Computation of Economic Equilibria* (Yale Univ. Press, New Haven, CT).