MATHEMATICS OF OPERATIONS RESEARCH
Vol. 9, No. 1, February 1984
Printed in U.S.A.

A NEWTON-TYPE ALGORITHM FOR THE SOLUTION OF THE IMPLICIT PROGRAMMING PROBLEM*

C. D. FEINSTEIN† AND S. S. OREN‡

A second-order algorithm is presented for the solution of the implicit programming problem. The implicit programming problem is a mathematical programming problem that has as its solution the optimal steady-states of an optimal control problem defined on an infinite horizon. The algorithm is shown to be a variant of Newton's method for solving a fixed-point problem defined implicitly by the implicit programming problem. Local and global convergence properties of the algorithm are briefly discussed.

1. Introduction. A very popular and useful model of economic dynamics is provided by the optimal control problem with discounting, formulated on an infinite time horizon ([3]–[5], [14]–[16]):

$$\text{minimize} \quad \int_0^\infty e^{-\rho t} l(x,u)\,dt \tag{1a}$$

$$\text{subject to} \quad \dot{x}(t) = f(x(t),u(t)); \tag{1b}$$

$$x(0) = x_0; \tag{1c}$$

$$(x(t),u(t)) \in X \times U \subseteq \mathbb{R}^n \times \mathbb{R}^m \quad \text{for each} \quad t \in [0,\infty). \tag{1d}$$

The variable $x \in \mathbb{R}^n$ is the state variable and the variable $u \in \mathbb{R}^m$ is the control variable. The set $U \subseteq \mathbb{R}^m$ may depend on $x(t)$ and t, explicitly. In that case, we shall write $U = U(x,t)$, where $U(x,t)$ is a set-valued mapping from $X \times [0,\infty)$ into $2^{\mathbb{R}^m}$, the set of all subsets of $\mathbb{R}^m$. The cost function l is real-valued and the system dynamics is described by the function f that takes values in $\mathbb{R}^n$. The scalar ρ is the discount rate.

The analysis of problem (1) indicates the special role that certain state-control pairs, referred to as optimal steady-states, play in the structure of the optimal trajectory of the dynamic model. In particular, it can be shown that, under certain convexity assumptions, the optimal dynamic trajectory converges to a unique point, say (x^*,u^*), which belongs to the null space of the system dynamics function f (thus, the terminology "steady-state") [15]. This result is known in the economics literature as the "turnpike" property of the optimal trajectory ([4], [16]). It can also be shown that this particular optimal steady-state may be thought of as a constant dynamic trajectory that is optimal in the class of trajectories with initial conditions equal to the state-component of the optimal steady-state, i.e., if $x_0 = x^*$ in (1c). In general, the optimal steady-state will not be unique. In such a case, the collection of optimal steady-states may be viewed as attractors for various dynamic trajectories: the optimal steady-state to which the dynamic trajectory converges is determined by the initial conditions (1c).

Given the importance of such steady-states in the analysis of the dynamic problem, it would be quite valuable to be able to find them without having to solve the full

*Received June 26, 1981; revised July 30, 1982.
AMS 1980 Subject Classification. Primary: 49D15, Secondary: 93C15.
OR/MS Index 1978 Subject Classification. Primary: 642 Programming/nonlinear/algorithms.
Key words. Infinite-horizon control, implicit programming problem, Newton's method.
†University of Santa Clara.
‡University of California, Berkeley.

0364-765X/84/0901/0075$01.25

dynamic problem. It is natural to seek a characterization of the optimal steady-states in terms of a static optimization problem. The static problem that characterizes the optimal steady-states has been formulated by Feinstein and Luenberger [6], and is called the implicit programming problem. The implicit programming problem, since it combines both static and dynamic elements, differs significantly in structure compared with the standard mathematical programming problem. Because of this structural difference, a new algorithmic approach is required for the solution of the implicit programming problem. In this paper, we present such an algorithm. The algorithm is a second-order method, essentially Newton's method, applied to a system of equations. However, since the equations are defined implicitly by the first-order necessary conditions of the implicit programming problem, Newton's method takes a form which may be conveniently referred to as a "horizontal" Newton method. This terminology is chosen to contrast with the standard "vertical" Newton method, as we shall depict in Figures 1 and 2 below.

2. The implicit programming problem. The implicit programming problem, which has as its solutions the set of optimal steady-states for the dynamic problem (1) (see [6] for this result), is (where θ is the zero vector)

$$\begin{aligned} &\text{minimize} && l(x,u) && \text{(2a)}\\ &\text{subject to} && f(x,u) - \rho(x - x^*) = \theta && \text{(2b)}\\ &&& (x,u) \in X \times U(x) \subseteq \mathbb{R}^n \times \mathbb{R}^m. && \text{(2c)} \end{aligned}$$

The unique structural feature of the implicit programming problem is contained in the constraint (2b). The term x^* indicates the value of the x-component of the solution to problem (2). Thus, the constraint of the problem is defined implicitly by the solution to the problem itself.

The most natural way to interpret the implicit programming problem is that it actually defines a mapping from $\mathbb{R}^n$ to $\mathbb{R}^n \times \mathbb{R}^m$. To highlight this interpretation, let us replace x^* in the constraint (2b) with a parameter $c \in \mathbb{R}^n$. As c varies over $\mathbb{R}^n$, a family of nonlinear programming problems is created. Moreover, for any c, the problem

$$\begin{aligned} &\text{minimize} && l(x,u) && \text{(3a)}\\ &\text{subject to} && f(x,u) - \rho(x - c) = \theta, && \text{(3b)}\\ &&& x \in X, \quad u \in U(x) && \text{(3c)} \end{aligned}$$

defines a mapping that takes $c \in \mathbb{R}^n$ into the solution of problem (3), $(x^*(c), u^*(c))$.

The implicit programming problem (2) may then be written

$$\begin{aligned} &\underset{c}{\text{minimize}} && l(x^*(c), u^*(c)) && \text{(4a)}\\ &\text{subject to} && x^*(c) = c, && \text{(4b)}\\ &&& c \in X, && \text{(4c)} \end{aligned}$$

where the minimization is taken over the fixed-points of the mapping $c \to x^*(c)$. The fact that this mapping is defined implicitly by the mathematical programming problem (3) suggests the terminology "Implicit Programming Problem." One would expect the minimization defined by (4) to be over a discrete set (although there is a possibility of degeneracy in the data of the problem, in which case the fixed-points of the mapping form a continuum). Indeed, the discrete nature of the "feasible" set for (4) indicates that what is of primary importance in the analysis of the implicit programming problem is the set of local solutions, i.e., all the fixed-points of the implicit mapping defined by (3). This is consistent with the dynamic problem from which the implicit

programming problem is derived: the local solutions are the attractors of the collection of the optimal dynamic trajectories. We will designate such points as feasible points for the implicit programming problem (2). It is clear that every feasible point of the implicit programming problem (i.e., a fixed-point of the implicit mapping) is a steady-state of the dynamic system $\dot{x} = f(x,u)$, since the term $\rho(x - c)$ in the constraint (3b) vanishes identically at the solution $x^*(c)$ whenever $x^*(c) = c$.

A natural candidate for determining the solution of the implicit programming problem is the method of successive approximations, often used for fixed-point calculations. Here, the sequence of approximations $\{x_{k+1}, u_{k+1}\}$ is determined recursively by solving the mathematical programming problems

$$\underset{x_{k+1},u_{k+1}}{\text{minimize}} \quad l(x_{k+1},u_{k+1}) \tag{5a}$$

$$\text{subject to} \quad f(x_{k+1},u_{k+1}) - \rho(x_{k+1} - x_k) = \theta \tag{5b}$$

where x_k is the solution of the immediately prior problem in the sequence. Unfortunately, the sequence of approximations may not converge, as can be demonstrated in the example where: $l = \frac{1}{2}(x^2/4 - 2u^2)$; $f = \frac{1}{2}x + u - 1$; $\rho = 1$.

The structural reason for such failure is that problem (5) need not be a contraction mapping on the (x,u,λ)-space, where λ represents the Lagrange multiplier. However, the structure of problem (3) contains other features that may be exploited to guide the iterations. We proceed to produce an algorithm that incorporates such information.

3. Derivation of the proposed algorithm. The algorithm we develop can be viewed as a way to generate a sequence of parameters $\{c_k\}$ that identify a sequence of problems in the family (3). The sequence of parameters is selected so that the sequence of solutions $\{x_k^*\} = \{x^*(c_k)\}$ converges to an element in the feasible set of the implicit programming problem, $\{c : x^*(c) = c\}$.

The idea underlying the algorithm is straightforward. Suppose we choose a value of the parameter c_k that generates a solution to problem (3) such that $x_k^* \neq x^*(c_k)$. Thus the point (x_k^*, u_k^*) is not feasible for the implicit programming problem. We can change c_k, letting $c_{k+1} = c_k + \Delta c$. This will induce perturbations in the solution $(\Delta x, \Delta u, \Delta\lambda)$, which are obtained by solving (3) for c_{k+1}.

If we knew how $x^*(c)$ varies with c, then we could select Δc such that $x^*(c_k + \Delta c) = c_k + \Delta c$, determining a feasible point. Of course, in general we cannot express $x^*(c)$ analytically, but we can specify Δc to satisfy the fixed-point condition to first order. Assuming $\nabla_c x^*(c)$, the gradient of the solution with respect to the parameter, exists, we may select Δc to satisfy the first-order fixed-point condition

$$x^*(c_k) + \nabla_c x^*(c_k)\Delta c = c_k + \Delta c. \tag{6}$$

The question of the behavior of the mapping $x^*(c)$ is a sensitivity issue and has been studied by various authors. We state a result, given by Fiacco [7], that indicates that the mapping $x^*(c)$ is continuously differentiable in a neighborhood of the parameter c_k when the second-order sufficiency and regularity conditions for (3) are satisfied at $(x_k^*, u_k^*, \lambda_k^*) = (x^*(c_k), u^*(c_k), \lambda^*(c_k))$.

We define the Lagrangian for problem (3) as

$$L_\rho(x,u,\lambda;c_k) = l(x,u) + \lambda^T\big[f(x,u) - \rho(x - c_k)\big]. \tag{7}$$

THEOREM 1. *Suppose the functions l and f are twice continuously differentiable in a neighborhood of the point (x_k^*, u_k^*). Then (x_k^*, u_k^*) is a local minimum of* (6) *with $c = c_k$ if*

there exists a Lagrange multiplier λ_k^* *such that*

$$\nabla_{x,u,\lambda} L_\rho(x_k^*, u_k^*, \lambda_k^*; c_k) = \theta, \quad \textit{or}$$

$$\nabla_x l(x_k^*, u_k^*) + \lambda_k^{*T}\left[\nabla_x f(x_k^*, u_k^*) - \rho I\right] = \theta,$$

$$\nabla_u l(x_k^*, u_k^*) + \lambda_k^{*T}\left[\nabla_u f(x_k^*, u_k^*)\right] = \theta,$$

$$f(x_k^*, u_k^*) - \rho(x_k^* - c_k) = \theta, \quad \textit{and} \tag{8d}$$

$\nabla_{x,u}^2 L_\rho(x_k^*, u_k^*, \lambda_k^*; c_k^*)$ *is positive-definite over the subspace*

$$M = \left\{(\xi, \eta) : \left[\nabla_x f(x_k^*, u_k^*) - \rho I\right]\xi + \left[\nabla_u f(x_k^*, u_k^*)\right]\eta = \theta\right\}. \tag{8e}$$

Further, if (x_k^*, u_k^*) *is a regular point of the constraints* (3b) (*i.e., the* $n \times n + m$ *matrix* $[\nabla_{x,u} f(x_k^*, u_k^*) - \rho(I \mid \Theta)]$ *is rank* n, *where* $(I \mid \Theta)$ *is an* $n \times n$ *identity matrix adjoined with an* $n \times m$ *zero matrix*), *then for* c *in a neighborhood of* c_k, *there exists a unique once continuously differentiable function* $(x^*(c), u^*(c), \lambda^*(c))$ *satisfying the second-order sufficiency conditions for a local minimum of* (3) *for all* c *in this neighborhood, and such that* $(x_k^*, u_k^*, \lambda_k^*) = (x^*(c_k), u^*(c_k), \lambda^*(c_k))$.

PROOF. See [7, Theorem 2.1]. ∎

Thus, if the second-order sufficiency conditions are satisfied at a regular point of the constraints, the solution to (3) along with the Lagrange multiplier are locally, continuously differentiable functions of the parameter c. This result indicates that the approach of selecting the perturbation Δc to satisfy the first-order fixed-point condition (6) is well defined. Solving (6) for the perturbation Δc, assuming the indicated inverse exists, we have the iteration: set $c_{k+1} = c_k + \Delta c$, where

$$\Delta c = -\left[I - \nabla_c x^*(c_k)\right]^{-1}(c_k - x^*(c_k)). \tag{9}$$

In order to implement this iteration, we require an expression for $\nabla_c x^*(c_k)$ in terms of the functions l and f. The theorem above indicates that for c in a neighborhood of c_k, $(x^*(c), u^*(c), \lambda^*(c))$ satisfies the first-order necessary conditions (8(b), (c), (d)). Let us represent these equations concisely as $F(x, u, \lambda, c) = \theta$. Hence in a neighborhood of c_k there holds

$$F(x^*(c), u^*(c), \lambda^*(c)) = \theta. \tag{10}$$

The Jacobian matrix of system (10) is the $(n + m + n) \times (n + m + n)$ symmetric matrix

$$\left[\partial F / \partial(x, u, \lambda)\right] = \left[\begin{array}{c|c|c} \nabla_{x,u}^2 L_\rho & \left[\nabla_{x,u} f - \rho(I \mid \Theta)\right] & \left[\nabla_{x,u} f - \rho(I \mid \Theta)\right]^T \\ \hline \left[\nabla_{x,u} f - \rho(I \mid \Theta)\right] & & \\ \hline & \Theta & \end{array}\right] \tag{11}$$

where $(I \mid \Theta)$ is an $n \times n$ identity matrix adjoined with an $n \times m$ zero matrix and Θ denotes an $n \times n$ zero matrix. If the second-order conditions (8e) hold at a regular point, then the Jacobian matrix is nonsingular [12, p. 231] and the implicit functions $(x^*(c), u^*(c), \lambda^*(c))$ are continuously differentiable. (This follows from the implicit function Theorem [1], and is the essence of the argument used to prove the theorem

quoted above.) Hence taking the gradient of (10):

$$[\partial F/\partial(x,u,\lambda)]\begin{bmatrix}\nabla_c x^*(c)\\ \nabla_c u^*(c)\\ \nabla_c \lambda^*(c)\end{bmatrix}+[\partial F/\partial c]=\Theta \qquad \text{where} \tag{12a}$$

$$[\partial F/\partial c]=\begin{bmatrix}\Theta\\ \Theta\\ \rho I\end{bmatrix}. \tag{12b}$$

Hence,

$$\begin{bmatrix}\nabla_c x^*(c)\\ \nabla_c u^*(c)\\ \nabla_c \lambda^*(c)\end{bmatrix}=-[\partial F/\partial\,(x,u,\lambda\,)]^{-1}\begin{bmatrix}\Theta\\ \Theta\\ \rho I\end{bmatrix} \tag{12c}$$

If we partition the inverse Jacobian matrix

$$[\partial F/\partial(x,u,\lambda)]^{-1}=\left[\begin{array}{c|c}F_{11} & F_{12}\\ \hline F_{21} & F_{22}\end{array}\right] \tag{13}$$

where F_{11} is an $(n+m)\times(n+m)$ matrix and $F_{21}=F_{12}^T$, then (12c) yields the expression for the gradient of the implicit mapping:

$$\nabla_c x^*(c)=-\rho[I\,|\,\Theta]F_{12}. \tag{14}$$

The iteration (9) then becomes

$$c_{k+1}=c_k-[I+\rho[I\,|\,\Theta]F_{12}]^{-1}[c_k-x^*(c_k)] \tag{15}$$

In particular, if $[\nabla^2_{x,u}L_\rho(x_k^*,u_k^*,\lambda_k^*)]$ is nonsingular, we have

$$F_{12}=[\nabla^2_{x,u}L_\rho]^{-1}[\nabla_{x,u}f-\rho(I\,|\,\Theta)]^T \times\left[[\nabla_{x,u}f-\rho(I\,|\,\Theta)][\nabla^2_{x,u}L_\rho]^{-1}[\nabla_{x,u}f-\rho(I\,|\,\Theta)]^T\right]^{-1}. \tag{16}$$

It is worth noting that the method of successive approximations would set $\Delta c = c_{k+1}-c_k = x^*(c_k)-c_k$, which one should compare with (15).

To summarize, the proposed algorithm can be conceptually expressed as:

Algorithm 1.

Step 1. Choose c_0, and set $k=0$.

Step 2. Solve the mathematical programming problem

$$\begin{aligned}&\text{minimize} && l(x_k,u_k)\\ &\text{subject to} && f(x_k,u_k)-\rho(x_k-c_k)=\theta.\end{aligned}$$

Step 3. If $x_k^*=c_k$, stop. Otherwise, find

$$c_{k+1}=c_k+\Delta c=c_{k+1}=c_k-[I+\rho[I\,|\,\Theta]F_{12}]^{-1}[c_k-x_k^*]$$

(where F_{12} is given by (13)).

Step 4. Increment k by one and return to Step 2.

4. Horizontal and vertical iterations for fixed-point problems. One recognizes the iteration (9) as Newton's method applied to the equation $c-x^*(c)=\theta$. However, the

fact that the mapping $x^*(c)$ is known only implicitly alters the manner in which Newton's method operates. Consider the problem of solving the (scalar) equation $f(x) = 0$. This is equivalent to finding a fixed-point of the function $g(x) = f(x) + x$. The Newton iteration for this problem is given by ($'$ denotes differentiation):

$$x_{k+1} = x_k - [1 - g'(x_k)]^{-1}(x_k - g(x_k)).$$

For each step of the iteration, we require x_k, $g(x_k)$, and $g'(x_k)$. Suppose that it is difficult to obtain $g(x_k)$ for some reason. Then an alternate approach to this problem may be attempted. Let us introduce a parameter, c, and consider the equation $g[x(c)] = c$. A fixed-point of g is then a fixed point of the implicit function $x(c)$. (The function g is analogous to the mathematical programming problem (3), and $x(c)$ is analogous to the solution of that problem.)

Consider the Newton iteration for solution of the (scalar) equation $c - x(c) = 0$, given by

$$c_{k+1} = c_k - [\ - x'(c_k)]^{-1}(c_k - x(c_k))$$

which, in terms of g, becomes

$$c_{k+1} = c_k - [g'[x(c_k)] - 1]^{-1} g'[x(c_k)](c_k - x(c_k)).$$

This iteration eliminates the need to evaluate $g[x(c_k)]$ since, by the definition of $x(c_k)$, $g[x(c_k)] = c_k$. Instead, the value $x(c_k)$ is required. Now $x(c_k)$ is found by inverting g, $x(c_k) = g^{-1}(c_k)$.

Generally, this is a difficult problem in itself, since it is typically easier to evaluate a function than to invert it. However, for the implicit programming problem, it is natural to consider this approach, since the inversion of g is exactly the process of solving the mathematical programming problem (3) for $c = c_k$.

The two approaches are compared in Figure 1. The intersection of the curve $y = g$ and the line $y = x$ is the required fixed-point. The standard Newton approach, where now the initial estimate is denoted c_k, proceeds in a "vertical" manner, as shown. The implicit approach, beginning from the same point, c_k, inverts the function g to find $x(c_k)$. The point c_{k+1} is determined by projecting along the direction of the slope $g'[x(c_k)]$ until the line $y = x$ is intersected. Then, g is again inverted to find $x(c_{k+1})$, and the iteration proceeds in this "horizontal" manner.

It is clear that by reflection through the line $y = x$, the parameter c may be thought of as an independent variable that maps into the value $x(c)$. Then this new method may be depicted in a "vertical" manner, considering the iterate pairs $(c_k, x(c_k))$. This is illustrated in Figure 2, where the space $\mathbb{R}^n \times \mathbb{R}^n$ of co-ordinate pairs (c, x) is represented by a two-dimensional diagram, and the graph of the mapping x^*, the set $\{(c, x) : x = x^*(c)\}$, is represented by the curve.

The intersection of the n-plane $\{(c, x) : x = c\}$ and the n-surface $\{(c, x) : x = x^*(c)\}$ in the $2n$-space $\mathbb{R}^n \times \mathbb{R}^n$ results typically in a point, denoted by (c^*, x^*), such that $c^* = x^*$.

As shown in Figure 2, given a point c_k, solve problem (3) to find x_k^*, the value of $x^*(c_k)$. From the point $(c_k, x^*(c_k))$ project along the gradient $\nabla_c x^*(c_k)$ until the plane $\{(c, x) : x = c\}$ is intersected. This determines c_{k+1}. The process repeats as shown in the figure to find c_{k+2}.

The projection (or "step") along the gradient $\nabla_c x^*(c_k)$ is given by

$$x - x^*(c_k) = \nabla_c x^*(c_k)(c - c_k).$$

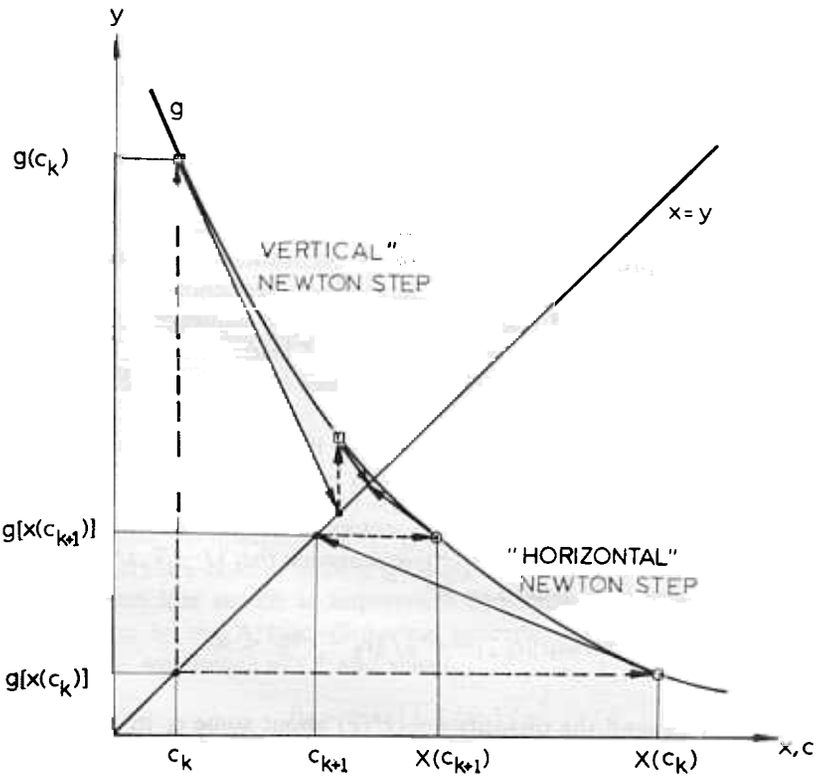

FIGURE 1

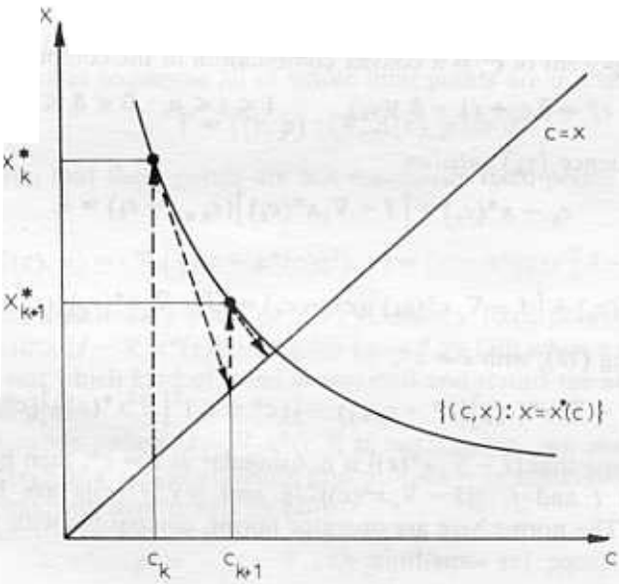

FIGURE 2. Geometric interpretation of the algorithm

Adding the intersection condition, $x = c = c_{k+1}$, we have

$$\begin{aligned} c_{k+1} - x^*(c_k) &= \nabla_c x^*(c_k)(c_{k+1} - c_k) \quad \text{or} \\ \left[I - \nabla_c x^*(c_k)\right](c_{k+1} - c_k) &= -(c_k - x^*(c_k)). \end{aligned} \tag{17}$$

If the matrix $[I - \nabla_c x^*(c_k)]$ is nonsingular, then (17) is identical to (9), the Newton iteration.

5. Local convergence and globalization of the algorithm. It is straightforward to show that for a linear-quadratic problem, where $l = \frac{1}{2}(x^T Q x + u^T R u)$, $Q > 0$, $R > 0$, and $f = Ax + Bu + d$, Algorithm 1 converges to the solution in one step. This suggests that for more general problems the algorithm is quadratically convergent, as we shall prove below. The proposed algorithm generates two sequences, $\{c_k\}$ and $\{(x_k^*, u_k^*, \lambda_k^*)\}$. We are chiefly concerned with the behavior of the sequence $\{c_k\}$, since it is the fixed-point property of the solution that is essential. Clearly, the condition $c_{k+1} = c_k$ is equivalent to $x_k^* = c_k$. Furthermore, the convergence of the sequence $\{c_k\}$ to a point c^* implies that $c^* - x^*(c^*) = \theta$.

In the following theorem, we will show that if the sequence generated by the algorithm converges, the order of convergence is two, as expected.

THEOREM 2. *Let the sequence $\{c_k\}$ of vectors in $\mathbb{R}^n$ be defined by Algorithm 1. Suppose that $\{c_k\} \to c^*$, where $c^* = x^*(c^*)$, and assume that $[I - \nabla_c x^*(c^*)]$ is nonsingular. Then, for any norm $\|\cdot\|$ defined on $\mathbb{R}^n$,*

$$\limsup_{k \to \infty} \|c_{k+1} - c^*\| / \|c_k - c^*\|^2 < \infty.$$

PROOF. Let us expand the quantity $c - x^*(c)$ about some c_k in a small neighborhood of c:

$$\begin{aligned} c - x^*(c) = c_k - x^*(c_k) &+ \left[I - \nabla_c x^*(c_k)\right](c - c_k) \\ &- \tfrac{1}{2}(c - c_k)^T \left[\nabla^2 x^*(c^\circ)\right](c - c_k), \end{aligned} \tag{18}$$

where each component of c° is a convex combination of the components of c and c_k,

$$c_i^\circ = \delta_i c_i + (1 - \delta_i)(c_k)_i, \qquad 1 \leqslant i \leqslant n, \quad 0 \leqslant \delta_i \leqslant 1.$$

By (17), the sequence $\{c_k\}$ satisfies

$$c_k - x^*(c_k) + \left[I - \nabla_c x^*(c_k)\right](c_{k+1} - c_k) = \theta.$$

Hence

$$c_k - x^*(c_k) + \left[I - \nabla_c x^*(c_k)\right](c^* - c_k) = \left[I - \nabla_c x^*(c_k)\right](c^* - c_{k+1}).$$

Thus, substituting (18), with $c = c^*$,

$$\theta = \left[I - \nabla_c x^*(c_k)\right](c^* - c_{k+1}) - \tfrac{1}{2}(c^* - c_k)^T \left[\nabla^2 x^*(c^\circ)\right](c^* - c_k).$$

Since we assume that $[I - \nabla_c x^*(c)]$ is nonsingular at $c = c^*$, then by the continuity assumptions on l and f, $\|[I - \nabla_c x^*(c)]^{-1}\|$ and $\|[\nabla_c^2 x^*(c)]\|$ are bounded in the neighborhood. (The norms here are operator norms, compatible with the vector norm chosen for $\mathbb{R}^n$.) Hence, for some finite K,

$$\|c_{k+1} - c^*\| \leqslant K \|c_k - c^*\|^2. \quad \blacksquare$$

In a neighborhood of a fixed-point of the mapping x^* the algorithm exhibits

quadratic convergence, typical of Newton's method. For an arbitrary initial point, c_0, however, the algorithm must be modified by a stepsize relaxation scheme to assure that a sequence is generated that converges to a fixed-point. Specifically, we choose

$$c_{k+1} = c_k + \beta_k p_k \tag{19}$$

where β_k is a stepsize parameter chosen to ensure global convergence and p_k is the solution of

$$\left[I - \nabla_c x^*(c_k)\right] p_k = -(c_k - x^*(c_k)). \tag{20}$$

If $[I - \nabla_c x^*(c_k)]$ is nonsingular, the modified iteration becomes

$$c_{k+1} = c_k - \beta_k \left[I - \nabla_c x^*(c_k)\right]^{-1} \left[c_k - x^*(c_k)\right], \tag{21}$$

replacing (9), the Newton iteration.

It is natural to measure progress toward a solution, a fixed point of x^*, by the function

$$Z(c) = \tfrac{1}{2} \|c - x^*(c)\|^2 \tag{22}$$

since for a fixed point c^*, $Z(c^*) = 0$, with $Z > 0$ otherwise. Thus, c_{k+1} will be a superior estimate, compared with c_k, if $Z(c_{k+1}) < Z(c_k)$. Since $\langle \nabla_c Z(c_k), p_k \rangle < 0$, it follows that (21) is a *descent method* ([13, 8.2.1]). Since the mapping x^* is known only implicitly, exact line search is impractical for stepsize selection. An alternative approach is given by the Armijo–Goldstein procedure ([2], [9], [13, p. 491]). Following this procedure, we choose $\gamma > 1$ and select

$$\beta_k = \max\{1, 1/\gamma, 1/\gamma^2, \dots\}$$

such that $c_{k+1} = c_k + \beta_k p_k$ satisfies the Goldstein criterion:

$$\left[Z(c_k + \beta_k p_k) - Z(c_k)/\beta_k \langle \nabla_c Z(c_k), p_k \rangle\right] > \sigma$$

for some $\sigma \in (0, \frac{1}{2})$, and $\langle \nabla_c Z(c_k), p_k \rangle < 0$. This procedure is known to converge to a stationary point of $Z(c)$ (see [13, 14.2.15], and [8, II.3.3 and II.3.5]). That is, the algorithm generates sequences all of whose limit points are in the set

$$\Gamma = \{(c, p) : \langle \nabla_c Z(c), p \rangle = 0\}. \tag{23}$$

Note, however, that these points are not necessarily fixed-points of the mapping x^*. Since

$$\langle \nabla_c Z(c), p \rangle = \langle \nabla_c \left(\tfrac{1}{2} \|c - x^*(c)\|^2\right), p \rangle = (c - x^*(c))^T \left[I - \nabla_c x^*(c)\right] p, \tag{24}$$

it follows that a stationary point of $Z(c)$ is either a fixed point of x^*, or a point at which the matrix $[I - \nabla_c x^*(c)]$ is singular ($p \neq \theta$ by (20) when c is not a fixed point). It is easy to test which kind of point is obtained and restart the algorithm if the result is not a fixed point.

At any iteration, when $[I - \nabla_c x^*(c_k)]$ is nonsingular, the matrix $[I - \nabla_c x^*(c_k)]^T \times [I - \nabla_c x^*(c_k)]$ is a positive-definite approximation to the Hessian. In this case, the vector p_k is uniquely determined by (20):

$$p_k = -\left[I - \nabla_c x^*(c_k)\right]^{-1} (c_k - x^*(c_k)).$$

We may then proceed by selecting β_k by the Armijo–Goldstein rule discussed above. However, when the matrix $[I - \nabla_c x^*(c_k)]$ is near singular, the modified iteration (21) breaks down, either because the inverse matrix does not exist or because the stepsize β_k

approaches zero. To overcome this difficulty, we further modify the algorithm using a Levenberg-type damping [11]. We replace iteration (21) with

$$c_{k+1} = c_k - \left[\epsilon_k I + H_k\right]^{-1}\left[I - \nabla_c x^*(c_k)\right]^T (c_k - x^*(c_k)), \tag{25}$$

where $H_k = [I - \nabla_c x^*(c_k)]^T[I - \nabla_c x^*(c_k)]$ and $\epsilon_k > 0$; ϵ_k can be chosen to yield descent ([13, 14.4.6]).

6. Numerical example: A neoclassical economic growth model with wealth effects. This example was analyzed by Kurz [10], who pointed out the interesting feature that there exist multiple optimal stationary points, which we consider as multiple local solutions of the implicit programming problem. In this example, the implicit programming problem is

$$\begin{aligned} &\text{minimize} && -U(k,c) && (26a)\\ &\text{subject to} && f(k) - c - \mu k - \rho(k - k^*) = 0, && (26b)\\ & && 0 \leqslant c \leqslant f(k), && (26c)\\ & && k \geqslant 0. \end{aligned}$$

The function $U(k,c)$ is a utility function, defined on capital stock, k, and consumption, c. The function $f(k)$ is a production function, and μ is the rate of depreciation of capital. The dynamic optimization problem that serves as a model for economic growth in the neoclassical model is

$$\begin{aligned} &\text{maximize} && \int_0^\infty e^{-\rho t} U(k(t), c(t))\, dt && (27a)\\ &\text{subject to} && \dot{k}(t) = f(k(t)) - c(t) - \mu k(t), && (27b)\\ & && k(0) = k_0, && (27c)\\ & && 0 \leqslant c(t) \leqslant f(k(t)). && (27d) \end{aligned}$$

The formulation of the implicit programming problem (26) follows from the structure of (27).

As a numerical example, we used the following data:

$$U(k,c) = 0.25\, k^{0.8} + c^{0.3}, \tag{28a}$$

$$f(k) = 0.3\, k^{0.45} - 0.01k - c, \tag{28b}$$

$$\rho = 0.3706. \tag{28c}$$

The implicit programming problem was solved for this case using an APL implementation of Algorithm 1. We selected an initial estimate $(k_0^*, c_0^*, \lambda_0^*) = (483.80406, 6.1646288 \times 10^{-2}, 10.572712)$, which is the solution of the implicit programming problem with $\rho = 0$. Table 1 exhibits the convergence of the algorithm. The left-hand column is the sequence of parameters, ξ_n, that define the sequence of constraints in the embedding family of mathematical programming problems, i.e., $f(k) - c - \mu k - \rho(k - \xi_n) = 0$.

For comparison we also solved the steady-state Euler–Lagrange equations for the original dynamic problem directly, using Newton's method (applied to the appropriate system of equations). The results are given in Table 2. In this case, Newton's method converges, although some experimentation indicated that this latter method was far more sensitive to choice of initial estimate than the implicit programming problem algorithm.

TABLE 1

Convergence of the implicit programming algorithm

ξ_n	k_n^*	c_n^*	λ_n^*
483.8040589	476.984605	2.570802162	0.1549077676
118.5037391	117.6892059	1.689178917	0.2078491717
51.98681055	51.84351784	1.307771277	0.2486274057
31.81941779	31.78913847	1.116139433	0.2777911069
23.95347608	23.94676563	1.015532527	0.2967806407
20.74255933	20.74128088	0.9671398309	0.3070992571
19.74953863	19.7494127	0.9510251414	0.3107326265
19.62770519	19.62770329	0.9490059023	0.31119529
19.62580904	19.62580904	0.9489744006	0.3112025212
19.62580858	19.62580858	0.9489743930	0.3112025229

TABLE 2

Convergence of Newton's method (applied to the first-order necessary conditions)

k_n^*	c_n^*	λ_n^*
483.8040589	0.006164628772	1.057271206
482.2246513	0.01484231429	0.1547823512
478.4632882	0.03547133794	0.1548075365
469.6508584	0.08360944604	0.1553978847
449.6033874	0.1920641418	0.1567757748
406.3939177	0.4203553561	0.1599303372
322.2974402	0.8386270257	0.1669956778
189.3469782	1.392467063	0.1824618190
63.48098920	1.591183674	0.2141729336
50.24867151	1.257602871	0.2485340339
28.11336075	1.120634220	0.2750105853
25.89316066	1.039494268	0.2910506423
21.16458566	0.9790827962	0.3038530200
20.14351847	0.9576445882	0.3091389635
19.65973872	0.9496093919	0.3110437868
19.62613253	0.9489800748	0.3112011373
19.62580860	0.9489743933	0.3112025228
19.62580858	0.9489743930	0.3112025229

After restarting the algorithm, other local solutions were found, for a total of three:

k_p^*	c_p^*	λ_p^*
19.625809	0.94897439	0.31120252
10.971532	0.77182366	0.35963222
6.3785408	0.62684526	0.41601502

These agree with the results reported by Kurz [10, Example 1].

It should be pointed out that comparing the number of iterations in Tables 1 and 2 is not meaningful since each iteration of Algorithm 1 involves an optimization problem. In this example, and in general, the main advantage of the new algorithm is its robustness rather than its speed.

7. Conclusions. In this paper, we have presented a variant of Newton's method for the solution of the implicit programming problem. The implicit programming

problem is formulated so that its solution is the set of optimal steady-states of a dynamic optimization problem. The algorithm responds to the essential structural feature of the implicit programming problem: a local solution to the problem is a fixed point of the mapping defined implicitly by the problem itself. One virtue of the algorithm is its robustness. In general, we cannot rely on Newton's method applied to the steady-state Euler–Lagrange equations to generate the steady-states of the dynamic problem. This is because the Jacobian matrix of those equations may be singular, in which case Newton's method breaks down.

References

[1] Apostol, T. (1960). *Mathematical Analysis*. Addison-Wesley, Reading, Massachusetts.

[2] Armijo, L. (1966). Minimization of Functions Having Lipschitz-Continuous First Partial Derivatives. *Pacific J. Math.* **16** 1–3.

[3] Arrow, K. J. and Kurz, M. (1970). *Public Investment, the Rate of Return, and Optimal Fiscal Policy*. Johns Hopkins Press, Baltimore.

[4] Cass, D. (1966). Optimum Growth in an Aggregative Model of Capital Accumulation: A Turnpike Theorem. *Econometrica* **34** 833–850.

[5] ——— and Shell, K., eds. (1976). *The Hamiltonian Approach to Dynamic Economics*. Academic Press, New York.

[6] Feinstein, C. D. and Luenberger, D. G. (1981). Analysis of the Asymptotic Behavior of Optimal Control Trajectories: The Implicit Programming Problem. *SIAM J. Control Optim.* **19** 561–585.

[7] Fiacco, A. V. (1976). Sensitivity Analysis for Nonlinear Programming Using Penalty Methods. *Math. Programming* **10** 287–311.

[8] Garcia-Palomares, U. (1973). Superlinearly Convergent Quasi-Newton Methods for Non-Linear Programming. Doctoral dissertation, University of Wisconsin.

[9] Goldstein, A. A. (1965). On Steepest Descent. *SIAM J. Control* **3** 147–151.

[10] Kurz, M. (1968). Optimal Economic Growth and Wealth Effects. *Internat. Econom. Rev.* **9** 348–357.

[11] Levenberg, K. (1944). A Method for the Solution of Certain Nonlinear Problems in Least-Squares. *Quart. Appl. Math.* **2** 164–168.

[12] Luenberger, D. G. (1973). *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, Reading, Massachusetts.

[13] Ortega, J. M. and Rheinboldt, W. C. (1970). *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York.

[14] Ramsey, F. P. (1928). A Mathematical Theory of Saving. *Econom. J.* **38** 543–549.

[15] Rockafellar, R. T. (1976). Saddle Points of Hamiltonian Systems in Convex Lagrange Problems Having a Non-Zero Discount Rate. Essay IV in *The Hamiltonian Approach to Dynamic Economics*, D. Cass and K. Shell, eds. Academic Press, New York.

[16] Samuelson, P. A. (1965). A Catenary Turnpike Theorem Involving Consumption and the Golden Rule. *Amer. Econom. Rev.* **55** 486–496.

FEINSTEIN: DEPARTMENT OF DECISION AND INFORMATION SCIENCES, UNIVERSITY OF SANTA CLARA, SANTA CLARA, CALIFORNIA 95053

OREN: DEPARTMENT OF INDUSTRIAL ENGINEERING AND OPERATIONS RESEARCH, UNIVERSITY OF CALIFORNIA, BERKELEY, BERKELEY, CALIFORNIA 94720