SIAM REVIEW
Vol. 41, No. 4, pp. 795–805

© 1999 Society for Industrial and Applied Mathematics

Pseudolinear Programming*

Serge Kruk†
Henry Wolkowicz†

Abstract. This short note revisits an algorithm previously sketched by Mathis and Mathis [*SIAM Rev.*, 37 (1995), pp. 230–234] and used to solve a nonlinear hospital fee optimization problem. An analysis of the problem structure reveals how the Simplex algorithm, viewed under the correct light, can be the driving force behind a successful algorithm for an almost linear problem. This presentation is intended for students who have been exposed to the Simplex method for linear programming and are progressing, via the Karush–Kuhn–Tucker conditions, toward nonlinear optimization.

Key words. nonlinear programming, Simplex method, pseudoconvexity, fractional programming

AMS subject classifications. 90C08, 90C30, 49M37

PII. S0036144598335259

1. A Seemingly Nonlinear Program.

In a past Classroom Notes column [6], Mathis and Mathis introduced an interesting optimization problem. Their model was practical, in this era of tightening budget constraints: it described a facet of hospital revenue used by managers in Texas to help set fees for procedures. Even more interesting, for the theoretically minded, is the fact that a trivial algorithm apparently solved, albeit without a convergence proof, a nonlinear, arguably difficult problem.

We revisit this problem to give a strong mathematical foundation to a slightly modified algorithm and to explain, along the way, why the problem is much simpler than expected at first glance. To do so, we introduce the concept of pseudolinear functions and view the Simplex method as an active set algorithm. The historically inclined will notice that the results used for this analysis are all 30 years old.

Mathis and Mathis gave a complete description of the problem, to which we refer the reader [6]. We just repeat the more important points of the model formulation.

- There are d departments in the hospital.
- Department i performs p_i procedures.
- Charges are assigned to procedure j in department i by:
  - $m_{ij} \geq 0$ represents Medicare/Medicaid charges;
  - $o_{ij} \geq 0$ represents other charges;
  - $c_{ij} > 0$ represents total charges ($c_{ij} = m_{ij} + o_{ij}$).
- The government-fixed outpatient cost is $C_i > 0$ for department i.
- The decision variable r_{ij} represents the fraction of increase in the charge for procedure j, department i.
- The charge increase is upper and lower bounded by $l_{ij} \leq r_{ij} \leq u_{ij}$.

*Received by the editors March 6, 1998; accepted for publication (in revised form) May 1, 1999; published electronically October 20, 1999.
http://www.siam.org/journals/sirev/41-4/33525.html

†University of Waterloo, Department of Combinatorics and Optimization, Waterloo, ON N2L 3G1, Canada (sgkruk@acm.org, henry@orion.uwaterloo.ca).

- The overall charge increase is a constant $q \times 100$ percent.

Their model then produces the following maximization problem:

$$\max F(r) := \sum_{i=1}^{d} \left[\sum_{j=1}^{p_i} o_{ij}(1+r_{ij}) + C_i \frac{\sum_{j=1}^{p_i} m_{ij}(1+r_{ij})}{\sum_{j=1}^{p_i} c_{ij}(1+r_{ij})} \right]$$

$$\text{subject to} \sum_{i=1}^{d} \sum_{j=1}^{p_i} c_{ij} r_{ij} = q \sum_{i=1}^{d} \sum_{j=1}^{p_i} c_{ij}, \qquad l_{ij} \le r_{ij} \le u_{ij}.$$

This results in Algorithm 1 for solving the maximization.

ALGORITHM 1 (Mathis and Mathis [6]).

Set all r_{ij} to q and sort the procedure in ascending order of $Q_{ij} := (\frac{\partial F}{\partial r_{ij}})(\frac{1}{c_{ij}})$.
while the sort produces a different ordering **do**
Set all r_{ij} to u_{ij}.
Beginning with the smallest Q_{ij}, assign l_{ij} to r_{ij} until the solution is nearly feasible. Then adjust the last r_{ij} to make it so.
Compute the new Q_{ij} and sort.
end while

The algorithm is unexpectedly simple, especially when contrasted to the problem it pretends to solve. But it begs a convergence proof. The authors checked the results against a well-known nonlinear solver to gain confidence in the procedure but stopped short of a proof. Moreover, one characteristic of the algorithm is nagging: looking at the feasible region as the polytope that it is, we see that iterates proceed from vertex to vertex.

To observe this behavior, let n be the number of variables ($n := \sum_{i=1}^{i=d} p_i$) and notice that the program has 1 equality constraint and $2n$ inequalities. Recall that a vertex is a basic feasible solution and that the algorithm, forcing all variables except one to either their upper or lower bound, will satisfy with equality exactly $n-1$ of the inequalities. Adding the single equality constraint, we obtain n constraints in n-space, a vertex.

It should have struck the reader as odd that a continuous nonlinear program would attain its optimal solution at a vertex. Moreover, the algorithm seems never to remain stuck at a local optimum; it finds, according to the authors, the global optimum. "Curiouser and curiouser," said Alice. Because of these surprises, we will reformulate the problem to highlight the characteristics that explain the algorithm's behavior.

2. A Somewhat Less Nonlinear Program. How should one attack a problem like the one above? The objective function is nonlinear, which eliminates a blind application of the well-known Simplex method for linear programming (see [1]). Interior-point methods (see [8]) usually assume convex functions, which does not appear to be the case here. Yet a full-featured nonlinear solver, possibly based on sequential quadratic programming (see [9]), seems to be overkill. In the hope of recognizing special characteristics of the problem and of simplifying the analysis, we now reformulate and get rid of the noise, a valuable step in any modeling problem. First we let the solution space be $\Re^n := \Re^{p_1} \oplus \cdots \oplus \Re^{p_d}$, where the p_i, we recall, indicate the number of procedures per department, to obtain

$$\begin{array}{lll} \widetilde{x_i} \in \Re^{p_i}, & \widetilde{x_i} & := [1+r_{i1}^t, 1+r_{i2}^t, \ldots, 1+r_{ip_i}^t]^t, \\ \widetilde{x} \in \Re^n, & \widetilde{x} & := [\widetilde{x_1}^t, \widetilde{x_2}^t, \ldots, \widetilde{x_d}^t]^t. \end{array}$$

Note that the decision variable is now nonnegative. We will use $\widetilde{x}$ to refer to the whole vector, $\widetilde{x_i}$ to refer to the subvector corresponding to department i, and x_i to refer to a component of $\widetilde{x}$, according to the property of the solution that we wish to highlight. We make corresponding substitutions to the other parameters.

$$\begin{aligned}
\widetilde{a_i} \in \Re^{p_i}, \quad \widetilde{a_i} &:= [o_{i1}^t, o_{i2}^t, \ldots, o_{ip_i}^t]^t, \\
\widetilde{b_i} \in \Re^{p_i}, \quad \widetilde{b_i} &:= C_i[m_{i1}^t, m_{i2}^t, \ldots, m_{ip_i}^t]^t, \\
\widetilde{c_i} \in \Re^{p_i}, \quad \widetilde{c_i} &:= [c_{i1}^t, c_{i2}^t, \ldots, c_{ip_i}^t]^t, \\
\widetilde{l_i} \in \Re^{p_i}, \quad \widetilde{l_i} &:= [1 + l_{i1}^t, 1 + l_{i2}^t, \ldots, 1 + l_{ip_i}^t]^t, \\
\widetilde{u_i} \in \Re^{p_i}, \quad \widetilde{u_i} &:= [1 + u_{i1}^t, 1 + u_{i2}^t, \ldots, 1 + u_{ip_i}^t]^t, \\
t &:= (1+q) \textstyle\sum_{i=1}^{i=d} \sum_{j=1}^{j=p_i} c_{ij},
\end{aligned}$$

where the coefficients now satisfy $\widetilde{c_i} = \widetilde{a_i} + C_i^{-1}\widetilde{b_i}$.

After these substitutions the problem is revealed as

$$\text{(P)} \qquad \max \quad \left\{ \sum_{i=1}^{d} \left[\widetilde{a_i}^t \widetilde{x_i} + \frac{\widetilde{b_i}^t \widetilde{x_i}}{\widetilde{c_i}^t \widetilde{x_i}} \right] \;\middle|\; \sum_{i=1}^{d} \widetilde{c_i}^t \widetilde{x_i} = t, \quad l \le x \le u \right\},$$

a *fractional program*. Problems of this form have been extensively studied, mostly because of their importance in finance (see [11]). There also have been recent attempts at developing interior-point algorithms specialized to fractional programs (see [3, 7]).

With our formulation, the first useful characteristic of the problem comes to light: The objective function is separable by department. It is a sum of functions, each concerned with different vectors. Analyzing each term yields the following information:

- The product $\widetilde{c_i}^t \widetilde{x_i}$ must be strictly positive.
- The feasible region is the intersection of a hyperplane with a box.
- Each term is either a linear ($\widetilde{a_i}^t \widetilde{x_i}$) or a *linear fractional* ($\frac{\widetilde{b_i}^t \widetilde{x_i}}{\widetilde{c_i}^t \widetilde{x_i}}$) transformation (also known as a *projective transformation*).

We call the reader's attention to the linearity of the feasible region and to what we will call the *pseudolinearity* of each term of the objective function. To explain this term, we need two definitions. A differentiable function f is *pseudoconvex* if

$$\nabla f(z)^t (y - z) \ge 0 \Rightarrow f(y) \ge f(z) \quad \forall y, z,$$

and *pseudoconcave* if

$$\nabla f(z)^t (y - z) \le 0 \Rightarrow f(y) \le f(z) \quad \forall y, z.$$

The geometric interpretation of pseudoconvexity is the following: if the directional derivative in a given direction is nonnegative, then the function is nondecreasing in that direction. For example, $f(x) = x + x^3$ is a pseudoconvex function: for any y larger than z, since the first derivative is everywhere positive, the product $f'(z)(y - z) = (1 + 3z^2)(y - z)$ will be positive. Pseudoconcavity has the mirror interpretation.

We will use *pseudolinear* to describe a function satisfying both the pseudoconvexity and pseudoconcavity conditions. The reader might want to verify that a linear function, for example, satisfies both conditions. But linear functions are not the only members of this class. An exercise in Mangasarian's classic text of the late 1960s [5,

Fig. 1 *Projective transformation* $f(x) = \frac{4x+3}{2x+1}$.

Chapter 9, problem 3] required that projective transformations be shown to be both pseudoconvex and pseudoconcave.

The interest we have in these conditions stems from the fact that hunting for the optimal solution of a linearly constrained pseudolinear objective function is as easy as hunting for a linear program. The graph of the fractional transformation $f(x) = \frac{4x+3}{2x+1}$ in Figure 1 suggests that, as far as stationary points are concerned, a pseudolinear function is as good as linear. The following lemma formalizes this vague statement.

LEMMA 2.1. *If the directional derivative of a pseudolinear function vanishes in a direction d, the function is constant on the line containing d.*

Proof. Say that, at some point z, we have $\nabla f(z)^t d = 0$. Then both inequalities describing pseudolinearity apply, and we have that $f(z + \alpha d) = f(z)$ for all α. The function f is constant along the line containing d. □

This implies that a pseudolinear function has no stationary point unless it is constant. This is critical to justifying the algorithm's behavior: a pseudolinear function, optimized over a polytope, only has global optima and attains its extrema at vertices. This is clear from the graph in Figure 1: If we optimize over the interval $x \in [0, 5]$, then clearly the maximum is at 0, and the minimum at 5. We will expand on this characteristic and make it more precise in the next section. But the crucial unstated assumption the algorithm makes is that the sum $\widetilde{a_i}^t \widetilde{x_i} + \frac{\widetilde{b_i}^t \widetilde{x_i}}{\widetilde{c_i}^t \widetilde{x_i}}$ maintains its (pseudo)convexity properties over the feasible region.

The reader will find a complete characterization of where such sums remain either quasi-convex or quasi-concave in [10], but the main point, for our purposes, is that the characterization will depend on the values of $\widetilde{c_i}, \widetilde{b_i}$, and $\widetilde{a_i}$ and that, in the absence of formal restrictions on these values, we cannot claim that the sum of linear and linear-fractional transformations remains pseudolinear. Since this condition is ignored by Mathis and Mathis, while some convexity assumption is essential to their algorithm, one can easily construct examples of failure.

We could decide to attack a larger class of problems but the algorithm might lose its amazing simplicity. More simply, we could check the data to ensure the condition. For the moment, we choose to believe, based on the practical experience of Mathis and Mathis with hospital management, that the assumption holds, and we proceed to give a solid mathematical foundation to the algorithm. After we have described the algorithm, we will see how to handle a larger class of problems with little additional work.

3. Characterization of a Global Solution. The usual goal of a constrained optimization algorithm is a Karush–Kuhn–Tucker (KKT) point, since such a point characterizes, under a constraint qualification, a necessary condition for optimality (see [5, Chapter 7, section 3]). This search is easily justified in our case.

LEMMA 3.1. *An optimal solution of* (P) *is a KKT point.*

Proof. The feasible region is defined by affine functions. This implies the KKT constraint qualification. □

No more than the recognition of the shape of the feasible region is required to obtain the constraint qualification. In particular, we need not insist on the linear independence of the gradients of the active constraints.

We are therefore justified in looking for a KKT point. The next question is whether there can be any spurious points not optimal for the problem. We can answer in the negative.

LEMMA 3.2. *Under the pseudolinearity assumption, any point satisfying the KKT conditions is a solution of* (P).

Proof. Since the constraints are affine, they are convex. Under the assumption of pseudoconcavity of the objective function, by a theorem of Mangasarian [5, Theorem 10.1.1], the KKT conditions are sufficient for optimality. □

This has been known for over 30 years. Everyone remembers that convexity of all the functions implies sufficiency of the KKT conditions, but Mangasarian showed that pseudoconvexity of the objective function and convexity of the level sets of the constraints (known as quasiconvexity, an even weaker condition) is all one really needs for sufficiency of the KKT conditions to hold.

We now know that the KKT conditions are both necessary and sufficient for optimality of our problem. The last ingredient, surprising in the framework of nonlinear optimization if usual in the context of linear programming, is that we can restrict our search for KKT points to the vertices of the polytope.

LEMMA 3.3. *Under the pseudolinearity assumption, if* (P) *is feasible, it has an optimal solution at a vertex.*

Proof. Assume that x^* is optimal for (P) and that x^* lies on an edge described by the interval $[x^* - \alpha_1 d, x^* + \alpha_2 d]$ for some nonzero direction d and $\alpha_1, \alpha_2 > 0$. By optimality, f must be nondecreasing from x^*; that is,

$$\nabla f(x^*)^t(x^* - \alpha_1 d - x^*) \geq 0 \quad \text{and} \quad \nabla f(x^*)^t(x^* + \alpha_2 d - x^*) \geq 0,$$

which implies

$$-\nabla f(x^*)^t d \geq 0 \quad \text{and} \quad \nabla f(x^*)^t d \geq 0.$$

This implies that $\nabla f(x^*)^t d = 0$. By Lemma 2.1, f is constant along d, and therefore the vertices adjacent to x^* are also optimal. □

We therefore have a complete characterization of optimal solutions and a finite subset of points of the feasible region that we need to investigate. The reader, well versed in linear programming, will have realized that all the conditions required for a successful application of the Simplex method are accounted for. A blind application of a Simplex code for linear programs will fail, of course. But viewed under the correct light, which in this case is an active set approach, the Simplex method is easy to transpose to our problem, which we now proceed to do.

4. A Pseudolinear Simplex Algorithm. We now describe more formally our implementation of the Simplex method for pseudolinear program (P). For the interested

reader, we provide a MATLAB implementation.[1]

4.1. Variable Declarations. The algorithm is described in consecutive sections, starting with the parameter declarations in Algorithm 2.

ALGORITHM 2. Pseudolinear Simplex method, parameters.

```
plSimplex(d, p, c, m, a, lb, ub, t)
integer d                                   {Number of departments}
integer p[1..d]                             {Number of procedures per department}
integer nb := sum_{i=1}^{d} p[i]            {Total number of procedures}
real c[1..nb]                               {Total charges}
real b[1..nb]                               {Transformed Medicare charges}
real a[1..nb]                               {Other charges}
real l_b[1..nb]                             {Lower bound on x}
real u_b[1..nb]                             {Upper bound on x}
real t                                      {Transformed total charge increase}
```

4.2. Finding an Initial Feasible Vertex. The first problem faced in any Simplex-type approach is the initial vertex. In our case, with only a box constraint intersecting a hyperplane, a basic feasible solution is within easy reach. The first step described in Algorithm 1 will work: set all variables to their upper (or lower) bound and decrease (or increase) them one by one until equality is satisfied.

ALGORITHM 3. Pseudolinear Simplex method, phase I.

```
real x[1..nb]                               {Decision variable}
integer k                                   {Index of inactive constraint}
if (c^t l_b > t ∨ c^t u_b < t) then
   return ∅;                                {Program is infeasible}
else
   x = l_b;                                 {Try variable at lower bound}
   for i = 1..nb do
      x[i] = u_b[i];                        {Move component to upper bound}
      if (c^t x > t) then
         x[i] = u_b[i] − ((c^t x − t)/c[i]); {Adjust to feasibility}
         k = i;                             {Record inactive constraint}
      end if
   end for
end if
```

Algorithm 3 will detect infeasibility or provide a feasible vertex and record which constraint (there is exactly one) is not an element of the *active set*, the set of constraints forced to equality. This so-called *inactive* constraint might not actually be slack. By chance, it might also be satisfied with equality. We will say that it is *saturated*. We will return to the case of such a *degenerate vertex* in section 4.4. Some heuristic for choosing the ordering of the variables based on the vectors m and c might prove effective but is not required.

4.3. Iterating to the Optimal Vertex. Now we move on to the crux of the algorithm. We compute the Lagrange multiplier estimates by trying to solve for dual feasibility. In fact, we will solve for everything but nonnegativity of the multipliers.

[1] http://orion.math.uwaterloo.ca/∼hwolkowi/henry/software/linpr.d/pseudolp.d.

Let the Lagrange multipliers be denoted by

$$\begin{aligned}
&\widetilde{\lambda}_i \in \Re_+^{p_i}, && \text{Upper bound multipliers of department } p_i,\\
&&& \lambda := [\widetilde{\lambda_1}^t, \widetilde{\lambda_2}^t, \ldots, \widetilde{\lambda_d}^t]^t;\\
&\widetilde{\gamma}_i \in \Re_+^{p_i}, && \text{Lower bound multipliers of department } p_i,\\
&&& \gamma := [\widetilde{\gamma_1}^t, \widetilde{\gamma_2}^t, \ldots, \widetilde{\gamma_d}^t]^t;\\
&\mu \in \Re, && \text{Multiplier of hyperplane constraint;}
\end{aligned}$$

and the Lagrangian be

$$\mathcal{L} := \sum_{i=1}^{d} \left[\widetilde{a_i}^t \widetilde{x_i} + \frac{\widetilde{b_i}^t \widetilde{x_i}}{\widetilde{c_i}^t \widetilde{x_i}} \right] + \sum_{i=1}^{d} \widetilde{\lambda_i}^t (\widetilde{u_i} - \widetilde{x_i}) + \sum_{i=1}^{d} \widetilde{\gamma_i}^t (\widetilde{x_i} - \widetilde{l_i}) + \mu \left(\sum_{i=1}^{d} (\widetilde{c_i}^t \widetilde{x_i} - t) \right).$$

Stationarity of the Lagrangian, together with complementarity, yields the following system, which the algorithm will have to solve:

$$\widetilde{a_i} + \frac{\widetilde{b_i}(\widetilde{c_i}^t \widetilde{x_i}) - \widetilde{c_i}(\widetilde{b_i}^t \widetilde{x_i})}{(\widetilde{c_i}^t \widetilde{x_i})^2} - \widetilde{\lambda_i} + \widetilde{\gamma_i} + \mu \widetilde{c_i} = 0, \qquad 1 \le i \le d, \tag{1}$$

$$\widetilde{\lambda_i}^t (\widetilde{x_i} - \widetilde{u_i}) = 0, \qquad 1 \le i \le d, \tag{2}$$

$$\widetilde{\gamma_i}^t (-\widetilde{x_i} + \widetilde{l_i}) = 0, \qquad 1 \le i \le d. \tag{3}$$

The system (1)–(3) turns out to be extremely simple to solve, partly because the objective is separable, partly because of the box constraints: a variable cannot be both at its upper bound and at its lower bound. (If it is, we can take the variable out of the problem altogether). Therefore at least half of the multipliers λ and γ must be zero and we know which ones.

Our active set consists of one for every pair of box constraints except exactly one, usually corresponding to a slack primal variable, a component x_k that is at neither its lower nor its upper bound. Although this constraint could be saturated (what we will call the *degenerate case*), we can force its two corresponding multipliers to zero. If the constraint is truly slack, then we have no choice. In either case, this leads to one pair $\lambda_k = \gamma_k = 0$, and we can solve first for μ and then for every other multiplier by simple substitution.

We then consider the sign of the multipliers. If all of them are nonnegative, we have a KKT point and, therefore, an optimal solution. If not, we need to move to another vertex. The classical way to do this, somewhat different from the Mathis and Mathis approach, is to choose the largest multiplier of the wrong sign (either λ_l or γ_l), drop the corresponding constraint ($x_l \le u_l$ or $x_l \ge l_l$), and move to the adjacent vertex. This choice, in the context of linear programming, is usually known as Dantzig's rule but it makes sense in much more general settings.

To motivate this choice, consider the program

$$\min\{f(x) \mid g(x) \le b\}.$$

Under appropriate conditions, we can write the optimal solution and the multipliers in terms of the right-hand side as $x(b)$ and $\lambda(b)$. Consider a small perturbation of the right-hand side of active constraint k, namely, b_k. A well-known sensitivity result (see [4, pp. 170, 323]) relates the change in the objective function to the change in the right-hand side, under appropriate conditions, as

$$\nabla f(x(b)) = -\lambda(b).$$

The interpretation of this result is that if $\lambda_k(b) < 0$, then a change of b_k to $b_k - \epsilon$ or, equivalently, a move of distance ϵ away from the constraint k, within the feasible region, will yield a first-order change in the objective function of

$$f(x(b + \epsilon e_k)) - f(x(b)) \approx -\epsilon \nabla f(x(b))^t e_k = \epsilon \lambda_k < 0,$$

where e_k has one in position k and zeros elsewhere. We therefore have a decrease in the objective function proportional to the magnitude of the multiplier.

To find the direction in which to move or, equivalently, which constraint to pick up, since we are moving from vertex to vertex, we need to solve an even simpler system. We need a direction d, satisfying $d_j = 0$ for all active constraints (all the currently active ones except the dropped constraint). Yet we need to remain feasible, which translates into $\sum_{i=1}^{d} \widetilde{c}_i^t d_i = 0$. Since we have one component corresponding to the constraint not in the active set (x_k) and one component corresponding to the dropped constraint (x_l), the condition reduces to

$$c_k x_k + c_l x_l = 0,$$

and the step length is just enough to get to the next vertex.

ALGORITHM 4. Pseudolinear Simplex method, phase II.

real $\gamma[1..nb]$ {Multiplier upper bound constraints}
real $\lambda[1..nb]$ {Multiplier lower bound constraints}
while 1 **do**
  Given x and $\lambda_k = \gamma_k = 0$, solve system (1)–(3) for λ, γ, μ
  Say $\lambda[l_\lambda] = \min\{\lambda\}; \gamma[l_\gamma] = \min\{\gamma\};$ {Find most negative multipliers}
  if $(\lambda[l_\lambda] < 0 \wedge \lambda[l_\lambda] < \gamma[l\gamma])$ {We should drop an upper bound} **then**
    if $((x[k] - u_b[k])c[k]/c[l_\lambda] > l_b[l_\lambda] - x[l_\lambda])$ **then**
      $x[l_\lambda] = x[l_\lambda] - c[k](u_b[k] - x[k])/c[l_\lambda]; x[k] = u_b[k]; k = l_\lambda;$
    else
      $x[k] = x[k] - c[l_\lambda](l_b[l_\lambda] - x[l_\lambda])/c[k]; x[l_\lambda] = l_b[l_\lambda];$
    end if
  else if $(\gamma[l\gamma] < 0)$ {We should drop a lower bound} **then**
    if $((l_b[k] - x[k])c[k]/c[l_\gamma] > x[l_\gamma] - u_b[l_\gamma])$ **then**
      $x[k] = x[k] - c[l_\gamma](u_b[l_\gamma] - x[l_\gamma])/c[k]; x[l_\gamma] = u_b[l_\gamma];$
    else
      $x[l_\gamma] = x[l_\gamma] - c[k](l_b[k] - x[k])/c[l_\gamma]; x[k] = l_b[k]; k = l_\gamma;$
    end if
  else
    return x; {We are optimal}
  end if
end while

This algorithm will increase the objective function at each step where we take a nonzero step. This can be formalized in the following lemma.

LEMMA 4.1. *The objective function f increases at each nondegenerate step of the algorithm.*

Proof. Say that k is the index of x corresponding to the constraints not in the active set and that l is the index of x corresponding to the dropped constraint (because either $x_k = l_k$ or $x_k = u_k$). The algorithm solves the system (1)–(3), from which we have that

$$\nabla f(x) - \lambda + \gamma + \mu c = 0, \quad \lambda_k = \gamma_k = 0,$$

and either of

$$\lambda_l = 0 \quad \text{or} \quad \gamma_l = 0,$$

whether we are dropping an upper or a lower bound constraint. Moreover, to compute a direction in which to move, the algorithm ensures that

$$c_k d_k + c_l d_l = 0.$$

Moving in the direction d (for a small step), we can estimate the change in the objective function by

$$\begin{aligned}\Delta f &= f(x+d) - f(x)\\ &= \nabla f(x)^t d + o(||d||)\\ &= (\lambda - \gamma - \mu c)^t d + o(||d||)\\ &= (\lambda_l - \gamma_l)d_l + (\lambda_k - \gamma_k)d_k - \mu(c_k d_k + c_l d_l) + o(||d||)\\ &= (\lambda_l - \gamma_l)d_l + o(||d||).\end{aligned}$$

Now we must distinguish two cases.

1. We dropped an upper bound. Then $\lambda_l < 0$, $\gamma_l = 0$, and x_l decreased so that d_l is negative. Then the last line above reads $\Delta f = \lambda_l d_l > 0$.
2. We dropped a lower bound. Then $\lambda_l = 0$, $\gamma_l < 0$, and x_l increased so that d_l is positive. Then the last line above reads $\Delta f = -\gamma_l d_l > 0$.

In both cases, the objective function increases in the direction away from the dropped constraint, and since the directional derivative cannot vanish on the edge we are following (by Lemma 2.1), the objective function must increase monotonically from the current vertex to the next one along the edge. □

4.4. Degeneracy. For this discussion to be complete, we need to discuss degeneracy, an uncommon yet possible situation. Recall that we defined a degenerate vertex as a vertex where a constraint not in the active set is nevertheless saturated. As an example, consider the simple three-dimensional case where all vertices are degenerate and the objective function is linear. This is clearly a special case of (P):

$$\max\left\{x_1 + 2x_2 + 3x_3 \;\middle|\; 2.4x_1 + 2.4x_2 + 2.4x_3 = 12, 1 \le x_i \le 3\right\}.$$

Phase I of the algorithm (Algorithm 3) will, as coded, produce $[3,1,1]^t$ as the initial vertex and consider the constraint $1 \le x_2$ as outside of the active set ($k = 2$) even if $x_2 = 1$. The iterations will be as shown in Table 1, and the vector $[1,1,3]^t$ is optimal since all multipliers are nonnegative. The first iteration is usually known as a degenerate pivot. We did not move in the primal space. The algorithm tried to increase both x_2, corresponding to the inactive constraint, and x_3, corresponding to the dropped constraint ($1 \le x_3$), but that is not possible while ensuring feasibility. The consequence was that we picked up a new constraint ($1 \le x_1$). The next iteration works.

An alternative to this would have been to recognize a degenerate vertex and react accordingly. In general, the simplex method for linear programming can be made to handle degenerate vertices by rules governing the choice of inclusion of components into the basis. Such rules can be shown to work but generally degrade performance. Because of the special structure of our feasible region, we can simplify the degeneracy handling by ensuring that if our inactive component x_k is actually at its upper

Table I

Iter	μ	x^t	k	l	λ^t	γ^t
0	-2/2.4	[3,1,1]	2	3	[-1,0,0]	[0,0,-1]
1	-3/2.4	[3,1,1]	3	1	[-2,0,0]	[0,1,0]
2	-3/2.4	[1,1,3]	3		[0,0,0]	[2,1,0]

(respectively, lower) bound, we choose to drop a lower (respectively, upper) bound constraint (corresponding, possibly, to the next most negative multiplier). In this way, we guarantee improvement of the objective function. One way to implement this is to replace the third line in Algorithm 4 by

$$\textbf{if } (\lambda[l_\lambda] < 0 \wedge x[k] < u[k]).$$

This chooses to drop a lower bound constraint if our slack variable is at its upper bound and therefore allows us to move. Like most degeneracy-avoiding routines, this slows down the algorithm.

5. Example. The original paper of Mathis and Mathis [6] ends with a small example. After the transformation of variables we suggested at the outset, the example reads as follows.

The overall charge increase is 10% ($q = 0.10$). We have four departments ($d = 4$), each with five procedures ($p_i = 5, 1 \leq i \leq d$). The fraction of increase must be between -5% and 15%, which translates to the box constraints, for all $1 \leq i \leq d$,

$$\widetilde{l_i} = 0.95e \leq \widetilde{x_i} \leq \widetilde{u_i} = 1.15e,$$

where e is the all-ones vector. The right-hand side of the equality constraint is $t = 999.966$.

The charges, suitably transformed, are given in Table 2, along with the feasible solution produced by phase I and the optimal solution produced by phase II. This solution, transformed back to the problem space, is the same as the one provided by Mathis and Mathis, modulo some rounding of values in their original table.

6. Conclusion. The algorithm we present here is not very different from that of Mathis and Mathis. It differs in the choice of constraints to drop and pick up. Or, in the usual language of the Simplex method, it differs in the choice of entering and leaving variables. But different pivoting rules hardly make a different algorithm, though they can drastically alter performance. (The steepest-descent edge is a case in point [2].)

The main differences in our description of the algorithm lie in the approach used to derive it and the fact that we have explained its behavior and proved its correctness under certain stated assumptions on the problem data.

What we have done is to look at an instance of the hospital management problem described by Mathis and Mathis and to identify as pseudolinear the class of problems into which it fell. We then recalled the characterization of these problems and their optimal solutions, allowing us to recognize that the Simplex method, viewed under the light of an active set method, would *provably* solve all instances of the problem. All of this comes from work done 30 and 40 years ago.

We end with a one-sentence conclusion: The Simplex method is much more than its tableau representation, and the class of problems to which it applies is much larger than that of linear programs.

Table 2

Charges			Solutions ($\widetilde{x}$)	
$\widetilde{a}$	$\widetilde{b} \times 10$	$\widetilde{c}$	Phase I	Optimal
16.35	901.2907	64.89	1.15	1.15
74.12	187.7225	84.23	1.15	1.15
39.26	43.4491	41.60	1.15	1.15
48.20	105.6519	53.89	1.15	1.15
12.63	133.6896	19.83	1.15	1.15
13.22	576.0705	61.66	1.15	1.0473
12.67	248.9766	33.61	1.15	1.15
22.71	772.2555	87.66	1.15	1.15
5.44	272.0432	28.32	1.15	0.95
7.24	207.9561	24.73	1.15	1.15
12.66	182.5256	26.26	1.15	1.15
6.84	93.5444	13.81	1.15	1.15
3.72	300.4962	26.11	1.15	0.95
6.66	740.5708	61.84	1.15	0.95
15.89	851.4282	79.33	1.0845	0.95
33.03	233.8502	47.07	0.95	1.15
0.18	117.7579	7.25	0.95	1.15
5.23	22.1525	6.56	0.95	1.15
7.60	691.7237	49.13	0.95	1.15
67.54	395.4134	91.28	0.95	1.15

REFERENCES

[1] G. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[2] J. J. Forrest and D. Goldfarb, *Steepest-edge simplex algorithms for linear programming*, Math. Programming, Ser. A, 57 (1992), pp. 341–374.

[3] R. W. Freund, F. Jarre, and S. Schaible, *On self-concordant barrier functions for conic hulls and fractional programming*, Math. Programming, Ser. A, 74 (1996), pp. 237–246.

[4] P.E. Gill, W. Murray, and M.H. Wright, *Practical Optimization*, Academic Press, New York, London, Toronto, Sydney, San Francisco, 1981.

[5] O.L. Mangasarian, *Nonlinear Programming*, McGraw–Hill, New York, 1969.

[6] F.H. Mathis and L.J. Mathis, *A nonlinear programming algorithm for hospital management*, SIAM Rev., 37 (1995), pp. 230–234.

[7] A.S. Nemirovski, *The long-step method of analytic centers for fractional problems*, Math. Programming, Ser. B, 77 (1997), pp. 191–224.

[8] Y.E. Nesterov and A.S. Nemirovski, *Interior Point Polynomial Algorithms in Convex Programming*, SIAM, Philadelphia, 1994.

[9] M.J.D. Powell and Y. Yuan, *A recursive quadratic programming algorithm that uses differentiable penalty functions*, Math. Programming, 7 (1986), pp. 265–278.

[10] S. Schaible, *A note on the sum of a linear and a linear-fractional function*, Naval Res. Logist., 24 (1977), pp. 691–693.

[11] S. Schaible, *Fractional programming*, in Handbook of Global Optimization, Nonconvex Optim. Appl. 2, Kluwer, Dordrecht, the Netherlands, 1995, pp. 495–608.