Pseudo-Linear Programming

Serge Kruk [*] Henry Wolkowicz [†]

February 26, 1998

University of Waterloo
Department of Combinatorics and Optimization
Waterloo, Ontario N2L 3G1, Canada

Key words: Nonlinear Programming, Simplex Method, Pseudo-convexity, Fractional Programming.

Abstract

This short note revisits an algorithm previously sketched by Mathis and Mathis, Siam Review 1995, 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 a nonlinear problem.

1 A seemingly nonlinear program

In a past Classroom Notes column, [6] Mathis and Mathis introduces an interesting optimization problem. Practical, in this era of budget constraints, their model describes a facet of hospital revenue and is used by managers in Texas to help in decision making. Even more interesting, for the theoretically minded, is the fact that a trivial algorithm seems to solve, 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 explain, along the way, why the problem is much simpler than expected at first glance. The historically inclined will notice that the results used for this analysis are all thirty years old.

Mathis and Mathis give a complete description of the problem to which we refer the reader. We just repeat the more important points of the model formulation.

[*]Email sgkruk@acm.org
[†]Email henry@orion.uwaterloo.ca.

- 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 increase is upper and lower bounded by $l_{ij} \leq r_{ij} \leq u_{ij}$.
- The overall charge increase is $q \times 100$ percent.

Their model then produces the following maximization problem

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

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

And their algorithm for solving the maximization is

Algorithm 1 : Mathis & Mathis

Set all r_{ij} to q and sort the procedure in ascending order of $Q_{ij} := \frac{o_{ij}}{q_{ij}}$.
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 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. It should have struck the reader as odd that a continuous nonlinear program would attain its optimal solution at a vertex. 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. Yet a full-features nonlinear solver, possibly based on Sequential Quadratic Programming (see [9]) seems an 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. First we let the solution space be $\Re^n := \Re^{p_1} \oplus \ldots \oplus \Re^{p_d}$, where the p_i, we recall, indicate the number of procedures per department, to obtain

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

Note that the decision variable is now non-negative. We will use $\widetilde{x}$ to refer to the whole vector, $\widetilde{x}_i$ to refer to the sub-vector 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.

$$\begin{array}{lll} \widetilde{a}_i \in \Re^{p_i}, & \widetilde{a}_i & := [o^t_{i1}, o^t_{i2}, \ldots, o^t_{ip_i}]^t, \\ \widetilde{b}_i \in \Re^{p_i}, & \widetilde{b}_i & := C_i[m^t_{i1}, m^t_{i2}, \ldots, m^t_{ip_i}]^t, \\ \widetilde{c}_i \in \Re^{p_i}, & \widetilde{c}_i & := [c^t_{i1}, c^t_{i2}, \ldots, c^t_{ip_i}]^t, \\ \widetilde{l}_i \in \Re^{p_i}, & \widetilde{l}_i & := [1 + l^t_{i1}, 1 + l^t_{i2}, \ldots, 1 + l^t_{ip_i}]^t, \\ \widetilde{u}_i \in \Re^{p_i}, & \widetilde{u}_i & := [1 + u^t_{i1}, 1 + u^t_{i2}, \ldots, 1 + u^t_{ip_i}]^t, \\ & t & := (1 + q) \sum_{i=1}^{i=d} \sum_{j=1}^{j=p_i} c_{ij}, \end{array}$$

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

$$\text{(P)} \qquad \max \quad \left\{ \sum_{i=1}^{d} \left[\widetilde{a}_i^t \widetilde{x}_i + \frac{\widetilde{b}_i^t \widetilde{x}_i}{\widetilde{c}_i^t \widetilde{x}_i} \right] \;\middle|\; \sum_{i=1}^{d} \widetilde{c}_i^t \widetilde{x}_i = t, \quad l \leq x \leq u \right\},$$

a *Fractional Program*. Problems of this form have been extensively studied, mostly because of their importance in finance (See [11]). There also has 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 (*projective transformation*).

We wish to attract the reader's attention to the linearity of the feasible region and the pseudo-linearity of each term of the objective function. That a projective transformation is both pseudo-convex and pseudo-concave was an exercise in Mangasarian's classic text of the late sixties [5] (Chapter 9, problem 3). We recall the basic definitions. A function f is pseudo-convex if

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

and pseudo-concave if

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

We will use *pseudo-linear* to describe a function satisfying both conditions. Hunting for an optimal solution for this class of functions is as easy as for a linear function, as the following result helps to understand.

Lemma 2.1 *If the directional derivative of a pseudo-linear 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 pseudo-linearity apply, and we have that $f(z + \alpha d) = f(z)$

for all α. The function f is constant along the line containing d. $\square$

This implies that a pseudo-linear function has no stationary point unless it is constant. This is critical to justify the algorithm's behavior: A pseudo-linear function, optimized over a polytope, only has global optima and attains its extrema at vertices. We will make this 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 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 remain pseudo-linear. 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 loose it's amazing simplicity. More simply, we could check the data to ensure the condition. For the moment, we choose to believe, based on Mathis and Mathis' practical experience 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 point (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. $\square$
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 pseudo-linearity 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 pseudo-concavity 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 thirty years. Everyone remembers that convexity of all the functions implies sufficiency of the KKT conditions but Mangasarian showed that pseudo-convexity of the objective function and convexity of the level sets of the constraints (known as quasi-convexity, 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 pseudo-linearity 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 non-decreasing from x^*, that is

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

which implies

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

This implies that $\nabla f(x^*)^t d = 0$ and 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, as we now proceed to do.

4 A pseudo-linear Simplex algorithm

4.1 Variable declarations

We intend to describe in detail our implementation of the Simplex method for pseudo-linear program (P). We will also provide a MATLAB implementation[1] for the interested reader. The algorithm will be described in consecutive sections, starting with the declarations in Algorithm 2.

Algorithm 2 : Pseudo-linear Simplex method. Declarations.

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 $m[1..nb]$	{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	{Total charge increase}
integer k	{Index of inactive constraint}
integer l	{Index of dropped constraint}
real $\gamma[1..nb]$	{Multiplier upper bound constraints}
real $\lambda[1..nb]$	{Multiplier lower bound constraints}

4.2 Finding an initial feasible vertex.

The first problem faced with 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 the equality is satisfied.

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

Algorithm 3 : Pseudo-linear Simplex method. Phase I.

if $(c^t l_b > t \vee c^t u_b < t)$ **then**
  return $\emptyset$; {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 is not an element of the active set. Note this constraint might not actually be slack. In case of such a *degenerate vertex*, the constraint could be saturated. We will return to this in Section 4.4. Some heuristics 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 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. Let the Lagrange multipliers be denoted by

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

and the Lagrangean be

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

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

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

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

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

The system (1,2,3) 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 and at its lower bound. (If it is, we can take the variable out of the problem altogether.) So that half of the multipliers λ and γ must be zero and we know which ones.

Our active set consists of every constraint except exactly one, usually corresponding to a slack primal variable, a component x_k that is neither at its lower or 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 (see [4]), somewhat different from 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.

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 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.

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

Algorithm 4 : Pseudo-Linear Simplex method. Phase II.

Given x and $\lambda_k = \gamma_k = 0$, solve system (1,2,3) 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

Lemma 4.1 *The objective function f increases at each non-degenerate 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 (1,2,3), 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

$$\Delta f \quad = \quad f(x + d) - f(x)$$

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

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), it must be that the objective function increases 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 as 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).

$$
\max \left\{ x_1 + 2x_2 + 3x_3 \mid 2.4x_1 + 2.4x_2 + 2.4x_3 = 12, 1 \leq x_i \leq 3 \right\}.
$$

Phase I of the algorithm will, as coded, produces $[3,1,1]^t$ as the initial vertex and considers the constraint $1 \leq x_2$ as outside of the active set ($k = 2$) even if $x_2 = 1$. The iterations will be

Iter	μ	x	k	l	λ	γ
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]

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 $(1 \leq x_3)$ but that is not possible while ensuring feasibility. The consequence was that we picked up a new new constraint $(1 \leq x_1)$. 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 (resp. lower) bound, we choose to drop a lower (resp. 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 routine, this slows down the algorithm.

5 Conclusion

The algorithm we present here is not very different from 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 it's behavior and proved it's correctness.

What we have done is to look at an instance of the hospital management problem described by Mathis and Mathis and identify as pseudo-linear the class of problems into which it fell. We then recalled the characterization these problems and their optimal solutions, allowing us to recognize that the Simplex method, viewed under the proper light of an active set method,

would *provably* solve all instances of the problem. All of this from work done thirty and forty years ago.

The one-sentence conclusion, perfect for a pedagogical column: The Simplex method is much more than its tableau representation and the class of problems to which it applies is much larger than linear programs.

References

[1] G. DANTZIG. *Linear Programming and Extensions.* Princeton University Press, Princeton, New Jersey, 1963.

[2] John J. Forrest and Donald Goldfarb. Steepest-edge simplex algorithms for linear programming. *Math. Programming*, 57(3, Ser. A):341–374, 1992.

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

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

[5] O.L. MANGASARIAN. *Nonlinear Programming.* McGraw-Hill, New York, NY, 1969.

[6] Frank H. Mathis and Lenora Jane Mathis. A nonlinear programming algorithm for hospital management. *SIAM Rev.*, 37(2):230–234, 1995.

[7] A. Nemirovski. The long-step method of analytic centers for fractional problems. *Math. Programming*, 77(2, Ser. B):191–224, 1997. Semidefinite programming.

[8] Y. E. NESTEROV and A. S. NEMIROVSKY. *Interior Point Polynomial Algorithms in Convex Programming : Theory and Algorithms.* SIAM Publications. SIAM, Philadelphia, USA, 1994.

[9] M.J.D. POWELL and Y. YUAN. A recursive quadratic programming algorithm that uses differentiable penalty functions. *Math. Prog.*, 7:265–278, 1986.

[10] S. Schaible. A note on the sum of a linear and a linear-fractional function. *Naval Research Logistics Quarterly*, 24:691–693, 1977.

[11] S. Schaible. Fractional programming. In *Handbook of global optimization*, volume 2 of *Nonconvex Optim. Appl.*, pages 495–608. Kluwer Acad. Publ., Dordrecht, 1995.