CANTER

9406

Department of Economics

UNIVERSITY OF CANTERBURY

CHRISTCHURCH, NEW ZEALAND

ISSN 1171-0705

LINEAR PROGRAMMING WITH MATHEMATICA

Michael Carter

Discussion Paper

No. 9406

Department of Economics, University of Canterbury
Christchurch, New Zealand

Discussion Paper No. 9406

November 1994

LINEAR PROGRAMMING WITH MATHEMATICA

Michael Carter

Abstract

We exploit the symbolic manipulation capability of *Mathematica* to elucidate the simplex algorithm of linear programming clearly and intuitively. We then develop a set of tools for conducting sensitivity analysis of the optimal solution. In order to utilize *Mathematica*'s efficient linear programming routine, we develop a function which can deduce the final tableau from the Spartan output of ConstrainedMax. This final tableau contains all the information usually provided by a good linear programming package, which can than be explored using the techniques of sensitivity analysis developed in the paper. This enables the package to be applied to a substantive problem, relying on *Mathematica*'s native code for intensive computation. To illustrate, we apply the package to analyse a classic problem in efficient nutrition.

Address for correspondence: Michael Carter
Department of Economics
University of Canterbury
Christchurch, New Zealand

internet: m.carter@econ.canterbury.ac.nz

Linear Programming with Mathematica

Michael Carter
Department of Economics
University of Canterbury
Christchurch, New Zealand

The subject of linear programming is surrounded by notational and terminological thickets. Both of these thorny defenses are lovingly cultivated by a coterie of stern acolytes who have devoted themselves to the field. Actually, the basic ideas of linear programming are quite simple.

Press, Teukolsky, Vetterling and Flannery (1992: 431)

1 Introduction

In his book *Methods of Mathematical Economics*, Joel Franklin relates a visit to the headquarters of the Mobil Oil Corporation in New York in 1958. The purpose of his visit was to study Mobil's use of computers. In those days, computers were rare and expensive and Mobil's installation had cost millions of dollars. Franklin recognized the person in charge; they had been post-doctoral fellows together. Franklin asked his former colleague how long he thought it would take to pay off this investment. "We paid it off in about two weeks" was the surprise response. Elaborating, he explained that Mobil were able to make massive cost savings by optimizing production decisions using linear programming, decisions which had previously been made heuristically.

It would be hard to exaggerate the importance of linear programming in practical optimization, in applications such as production scheduling, transportation and distribution, inventory control, job assignment, capital budgeting and portfolio management. Franklin's anecdote highlights the enormous benefits which can accrue from optimizing recurrent decisions.

There are two main reasons for the practical success of linear programming. First, many production processes and economic systems are linear or nearly so. Linear programming provides an appropriate mathematical model for such

*I gratefully acknowledge the assistance of John George, Department of Management, University of Canterbury in negotiating the thickets of linear programming. He is of course completely exonerated for any breach in my defenses. I also acknowledge my considerable debt to the book by Vašek Chvátal (1983), whose refreshing approach to linear programming illuminated my path.

processes. Second, there exists of a very efficient algorithm (the simplex algorithm) for solving most linear programming problems. The postwar conjunction of the availability of digital computers and the discovery of the simplex algorithm by George Dantzig paved the way for successful industrial application as exemplified by Mobil's experience.

Linear programming is also important to economists and game theorists. Although most economic models are nonlinear, linear models of exchange, production and capital accumulation serve an important didactic role (Dorfman, Samuelson and Solow, 1958). Furthermore, an understanding of the simplex algorithm and the duality theorem enhances comprehension of nonlinear optimization, mastery of which is central to economic analysis. In game theory, the solution of zero sum games is a linear programming problem and the minimax theorem is formally equivalent to the fundamental duality theorem of linear programming. The efficient solution of non-zero sum games uses a modification of the simplex algorithm called the complementary pivot algorithm (Lemke and Howson, 1964; Wilson, 1992). Much of cooperative game theory reduces to the application of linear programming theory and techniques (Carter, 1993).

Despite recent discoveries alternative "interior point" algorithms, the simplex method and its variants remain by far the most common practical method for solving linear programming problems. The simplex algorithm is based upon a very simple intuitive idea of successive improvement. However, because most textbook treatments aim at describing a mathematical formulation suitable for implementation by conventional programming languages, their discussion tends to hide the intuitive simplicity of the algorithm. The symbolic manipulation capability of *Mathematica* enables the simplex algorithm to be elucidated more clearly and intuitively, and significance of all the results understood. This is the first objective of this paper.

Mathematica already includes an implementation of the simplex method (**ConstrainedMax**) as part of its native code. Though fast, the output from this routine is rather Spartan. The second objective of this paper is to work backwards, manipulating the output of **ConstrainedMax** to recover the "final tableau" and hence to deduce all the information which is usually provided by a good linear programming package. This can then be used to conduct a *sensitivity analysis* of the optimal solution. In this way, the paper enhances the utility of the in-built linear programming facility.

Since our implementation of the simplex algorithm is designed for explanation rather than execution, our code emphasizes clarity rather than efficiency.

2 The problem

To make the exposition easier to follow, we start with a simple specific example. A furniture maker can produce three products, bookcases, chairs and desks. Each product requires machining, finishing and some labor. The supply of these resources is limited. Unit profits and resource requirements are listed in the following table.

	Bookcases	Chairs	Desks	Capacity
Finishing	2	2	1	30
Labor	1	2	3	25
Machining	2	1	1	20
Net profit	3	1	3	

The problem of maximizing profit can be specified as the following linear program. Maximize

```
profit = 3x[b] + x[c] + 3x[d];
```

subject to the

```
constraints =
{ 2x[b] + 2x[c] +  x[d] <= 30,
   x[b] + 2x[c] + 3x[d] <= 25,
  2x[b] +  x[c] +  x[d] <= 20
  };
```

and $x_i \geq 0$

For analysis, inequalities are more difficult to manipulate than equations. By introducing *slack variables*, we can transform the inequalities into a corresponding system of equations. This is called the *standard form.* The slack variables measure the unused capacity of a resource at any given production plan. For convenience, we label the spare capacities of finishing, labor and machining s_f, s_l and s_m respectively.

```
StandardForm[constraints_, labels_List] :=
Transpose @
   {labels,constraints} /.
{i_, lhs_ <= rhs_}:> s[i] == rhs - lhs
```

```
StandardForm[constraints, {f,l,m}]
```

```
{s[f] == 30 - 2 x[b] - 2 x[c] - x[d],

 s[l] == 25 - x[b] - 2 x[c] - 3 x[d],

 s[m] == 20 - 2 x[b] - x[c] - x[d]}
```

For example, if the furniture maker was to produce 1 bookcase, 2 chairs and 3 desks, the unused capacities would be

```
% /. {x[b] -> 1, x[c] ->2, x[d] -> 3}
```

```
{s[f] == 21, s[l] == 11, s[m] == 13}
```

The linear programming problem can be succinctly summarized by prepending the objective function to the constraints. To ensure that the result is nicely formatted, we can set **$PrePrint** to apply **MatrixForm** automatically to any system of equations (tableau).

```
TableauQ[_] := False
TableauQ[{_Equal..}] := True
$PrePrint = If[TableauQ[#], TableForm[#],#]&;
```

```
problem=Prepend[StandardForm[constraints, {f,l,m}],
z == profit]
```

```
z == 3 x[b] + x[c] + 3 x[d]

s[f] == 30 - 2 x[b] - 2 x[c] - x[d]

s[l] == 25 - x[b] - 2 x[c] - 3 x[d]

s[m] == 20 - 2 x[b] - x[c] - x[d]
```

We can also use **TableauQ** to preclude applying **StandardForm** to a system which is already in standard form.

```
StandardForm[constraints_?TableauQ,labels_List] := constraints
```

PROGRAMMING NOTE: We use *Mathematica* functions, for example $x[b]$, $s[l]$, to represent variables rather than simple variable names like $x1, x2$ or b, c, d. This faciltates manipulation in *Mathematica*, for example in the use of pattern matching and automatic generation of variable names. It more closely resembles the subscript notation used in many presentations and would enable the use of more descriptive variable names, such as $x[desks]$ if required. We adopt the convention of using $x[j]$ to denote decision variables and $s[i]$ to denote slack variables. We rely on this convention occasionally in the code.

3 Searching for a solution

Are there any feasible solutions to the production problem? Yes, an obvious possibility is to produce nothing ($x_i = 0$) but it is not very profitable. Another feasible solution was cited in the previous section. The production plan

```
productionPlan = {x[b] -> 1, x[c] -> 2, x[d] -> 3};
```

is feasible

```
constraints /. productionPlan
```

```
{True, True, True}
```

and produces a profit of

```
profit /. productionPlan
```

```
14
```

A better plan is

```
{profit, constraints} /.
(productionPlan = {x[b] -> 2, x[c] -> 4, x[d] -> 5})
```

```
{25, {True, True, True}}
```

Unfortunately, we cannot produce

```
{profit, constraints} /.
(productionPlan = {x[b] -> 5, x[c] -> 4, x[d] -> 5})
```

```
{34, {True, False, True}}
```

since the requirements of 5 bookcases, 4 chairs and 5 desks exceeds the available labor supply. If we eliminate the chairs, we can produce

```
{profit, constraints} /.
(productionPlan = {x[b] -> 5, x[c] -> 0, x[d] -> 5})
```

```
{30, {True, True, True}}
```

This is the most profitable plan which we have considered so far, but is it the best? How can we tell? We could continue to explore various permutations, but it is not clear how we would know when we have reached the optimum plan or even that an optimum plan exists. Clearly, we need some systematic way in which to explore the various alternatives.

A sensible starting point would be to produce as much as possible of the most profitable item. Bookcases (x_b) and desks (x_d) are equally profitable. Arbitrarily, let us choose bookcases. What is the maximum number of bookcases x_b which can be produced with the available resouces? Inspection reveals that machining capacity is the limiting factor in the production of bookcases. Each bookcase requires 2 hours of machining. A total of 20 hours is available. Therefore, we can produced a maximum of 10 bookcases. Our first tentative production plan is

```
productionPlan = {x[b] -> 10, x[c] -> 0, x[d] -> 0};
```

This plan exhausts the machining capacity but leaves spare capacity in finishing and spare labor, resulting in a profit of $30.

```
problem /. productionPlan
```

```
z == 30

s[f] == 10

s[l] == 15

s[m] == 0
```

Is this an optimal solution? It is not immediately obvious. Sure, we have spare resources. But we know that we cannot produce any more bookcases. Furthermore, since the machining constraint is binding, any output of the other goods can only be accomplished by contracting the production of bookcases. In other words, any production of chairs or desks involves a tradeoff against the production of bookcases. Is such a tradeoff profitable?

The tradeoff imposed by the limited supply of machining capacity is represented by the third constraint, which is called the *pivot row.*

```
pivotRow = First @ Cases[problem, s[m] == _]
```

```
s[m] == 20 - 2 x[b] - x[c] - x[d]
```

This equation imposes a constraint on the total production of bookcases which can be highlighted by solving this constraint for x_b.

```
RevisedPivotRow =
    Roots[pivotRow, x[b]] // ExpandAll
```

```
                 s[m]   x[c]   x[d]
x[b] == 10 - ---- - ---- - ----
                  2      2      2
```

This equation expresses the machining constraint in terms of the production of bookcases. It reveals two important technological facts.The intercept is the number of bookcases which can be produced by devoting all of the machining resource to the production of bookcases. The coefficients on the other goods,x_c and x_d indicate that, *while the machining constraint remains binding,* every chair and desk produced reduces the available output of bookcases by 1/2 unit. This is the technological tradeoff imposed by the machining constraint. The revised constraint reveals that (1) a maximum of 10 bookcases can be produced by concentrating on bookcases alone and (2) every chair or desk produced would reduce the potential number of bookcases by 1/2. This is the fundamental technological tradeoff.

The *economics of this tradeoff* is revealed by substituing the technological tradeoff into the objective function. To do this, we first express the revised pivot row as a transformation rule

```
pivotRule = ToRules[RevisedPivotRow]
```

```
                  s[m]    x[c]    x[d]
{x[b] -> 10 - ---- - ---- - ----}
                   2       2       2
```

and use this to transform the objective function

```
RevisedObjective = profit /. pivotRule //
 ExpandAll
```

```
      3 s[m]    x[c]    3 x[d]
30 - ------ - ---- + ------
        2        2        2
```

This revised objective function reflects the *economic position* at the tentative production plan ($x_b] = 10, x_c = 0, x_d = 0$). The intercept 30 represents the net profit earned by this plan. The coefficients on x_c and x_d evaluate the marginal benefit of producing chairs or desks, while taking account of the need to reduce correspondingly the output of bookcases (the technological tradeoff). This reveals that the marginal benefit of substituting chairs for desks is negative, while the marginal benefit of producing a desk, *after accounting for the corresponding reduction in the output of bookcases*, is positive. Some substitution of desks for bookcases is desirable. Producing one less bookcase would enable the firm to produce 2 desks, increasing net profit by \$3. At the margin, each desk is worth \$3/2. Note that bookcases and desks have the same selling price, \$3. The reason that it is more profitable to produce a combination of bookcases and desks (as opposed to bookcases alone) is that desks require less of the scarce machining capacity.

The coefficient of x_d, 3/2, measures the true opportunity cost of the producing all 10 bookcases, which is the foregone opportunity to produce some desks. Similarly, the coefficient $-3/2$ on the slack variable s_m measures the opportunity cost of the machining constraint — the foregone potential output of bookcases and/or desks. Its inverse (negative), 3/2 is called the *shadow price* of machining capacity. It indicates that profit could be increased by 3/2 if this constraint were relaxed by one unit. Similarly, the coefficient 3/2 on the $x[d]$ can be interpreted as the *shadow price* of the negativity constraint on this variable. Shadow prices play a central role in the solution of linear programming problems. As we will shortly demonstrate, they guide the process of sequential improvement leading to the optimal solution. At the optimal solution, the shadow prices provide important information about the sensitivity of the solution to changes in the parameters of the problem.

The revised objective function reveals that our tentative plan $(10, 0, 0)$ is not optimal; some substitution of desks for bookcases is desirable. How many desks should we produce? Linearity implies that we should exploit any profitable substitution as far as resources allow. The impact on machining is implicitly taken account of by the pivot row. We should substitute desks for bookcases until we run out of some other resources. The impact of this substitution on resource requirements can be obtained by using the pivot rule to rewrite the other resource constraints.

```
RevisedConstraints = Rest[problem] /.
{pivotRow :> RevisedPivotRow,
   otherRow_Equal :> (otherRow /. pivotRule)} //
ExpandAll
```

```
s[f] == 10 + s[m] - x[c]

              s[m]   3 x[c]   5 x[d]
s[l] == 15 + ---- - ------ - ------
               2       2        2

              s[m]   x[c]   x[d]
x[b] == 10 - ---- - ---- - ----
               2      2      2
```

The absence of x_d from the first equation indicates that substituting desks for bookcases will have no impact on the utilization of finishing capacity. (Although desks are less demanding of finishing than bookcases, the machining constraint allows us to substitute 2 desks for every bookcase.) The second equation reveals that substitution will utilize spare labor capacity, since desks are considerably more labor intensive than bookcases. Each desk will utilize 5/2 units of spare labor capacity. Available spare capacity is 15 units, allowing the substitution of six desks for three bookcases. Therefore, our second tentative production plan is

```
productionPlan = {x[b] -> 7, x[c] -> 0, x[d] -> 6};
```

```
problem /. productionPlan
```

```
z == 39

s[f] == 10
```

```
s[l] == 0

s[m] == 0
```

This plan exhausts both labor and machining capacity and produces of a profit \$39, compared to \$30 with the earlier plan. This certainly represents an improvement over our first production plan. Is this now an optimal solution? This is the same question we asked of the first production plan and we could repeat the same analysis from a new starting point. This is the pivotal step in linear programming. Let us formalize this procedure.

4 Formalizing this procedure - pivoting

Let us review what we have just done. Faced with the linear programming problem

```
problem

z == 3 x[b] + x[c] + 3 x[d]

s[f] == 30 - 2 x[b] - 2 x[c] - x[d]

s[l] == 25 - x[b] - 2 x[c] - 3 x[d]

s[m] == 20 - 2 x[b] - x[c] - x[d]
```

we proposed a tentative production plan which involved maximum production of the most profitable good, in this case bookcases. Examining the resource constraints, we deduced that machining capacity would most limit the production of bookcases. The insightful step was rewriting the specification of the problem to more clearly represent the technological and economic tradeoffs entailed by the maximum production of bookcases. This rewritten specification clearly indicated that the tentative production plan of 10 bookcases was not optimal, and also indicated the direction of improvement.

The tentative production plan (10, 0, 0) producing as many desks as possible a *basic feasible solution.* It is a basic solution in that no more than the minimal number of variables are non-zero. In this case, the basis variables are x[b], s[1] and s[3]. The process of moving from one basic feasible solution to another and rewriting the specification of the problem to fully reflect the technological and economic tradeoffs from the perspective of a the new basic solution is called *pivoting.* This procedure was carried out in the preceding section and is summarized in the function **Pivot**.

```
Pivot[tableau_?TableauQ,x_[j_],s_[i_]] := Module[{pivotRow},
    pivotRow = First @ Cases[tableau, s[i] == _];
    tableau /. {pivotRow :> Roots[pivotRow,x[j]],
  otherRow_Equal :> (otherRow /.
Flatten @ Solve[pivotRow,x[j]])} //
ExpandAll
]
```

The function **Pivot** returns the revised constraints together with the revised objective function. These summarize the optimization problem from the viewpoint of the tentative production plan (the basic feasible solution). The left hand side of the revised constraints indicates the values of the positive decision variables and the positive slack variables, which measure spare resource capacity. The coefficient of the revised objective function measure the shadow prices of the fully used resources and the zero decision variables. In the linear programming literature, the revised objective function and constraints constitute what is called a *tableau*. The tableau is a very compact representation of the technological and economic tradeoffs which pertain at any tentative solution.[1]

In the production planning example, our first basic feasible solution is obtained by concentrating on the production of bookcases. The maximum number of bookcases is limited by the machining constraint. The revised specification at this basic solution is

```
tableau = Pivot[problem,x[b],s[m]]
```

```
            3 s[m]   x[c]   3 x[d]
z == 30 - ------ - ---- + ------
              2        2       2

s[f] == 10 + s[m] - x[c]

              s[m]   3 x[c]   5 x[d]
s[l] == 15 + ---- - ------ - ------
               2       2        2

              s[m]   x[c]   x[d]
x[b] == 10 - ---- - ---- - ----
               2      2      2
```

Our next step was to substitute some desks for bookcases. The extent of the substitution was limited by the labor resource. Making this substitution leads to the revised tableau.

```
Pivot[tableau,x[d],s[l]] // TableForm
```

```
            3 s[l]   6 s[m]   7 x[c]
z == 39 - ------ - ------ - ------
              5        5        5

s[f] == 10 + s[m] - x[c]
```

[1] In fact, the linear programming literature usually reserves the term "tableau" for the matrix of coefficients of the preceding equations. The full system of equations was called a *dictionary* by Strum (1972), whose approach was fully developed by Chvátal (1983).

```
            2 s[l]    s[m]    3 x[c]
x[d] == 6 - ------ + ---- - ------
              5       5        5

            s[l]    3 s[m]    x[c]
x[b] == 7 + ---- - ------ - ----
             5        5       5
```

The revised objective function (top row) reveals that this is now an optimal production plan, since all the coefficients are negative. Any move which increases the currently zero variables (s_l, s_m, x_c) must reduce profit below \$39. There is no further profitable improvement. The optimal plan requires the production of 7 bookcases and 6 desks. It leaves spare finishing capacity of 10, which cannot be utilized profitably.

The procedure we have followed in seeking an optimal solution is one of sequential improvement. At each tentative production process, we looked for a profitable improvement. The potential for profitable improvement was summarized in the revised objective function. Having identified a potential improvement, we then made that improvement to the full extent possible (which is profitable because of linearity), while accounting for all the tradeoffs necessary to adhere to the binding constraints. Finally, we revised the formulation of the problem to account for any new tradeoffs inherent in the new tentative production plan. This stepwise improvement procedure is known as *pivoting*.

To automate the pivoting process, we need to identify the critical resource which limits the extent of any potential improvement, that is the resource which most limits the extent to which the variable x_i can be increased. This is done by comparing the ratio of the available surplus (the constant term) of the resource to the unit requirements of the activity (the coefficient of x_i) and selecting the smallest. This is implemented in the function **LimitingResource**, which makes use of pattern matching to parse the components of each constraint. ExpansionRatios is a list comprising each resource (represented by teh corresponding slack variable) and its expansion ratio. **MinPairs** selects the mimimum element in this list.

```
MinPair[pairs_List] :=
         Fold[If[#1[[2]] <= #2[[2]],#1,#2]&,
First[pairs],Rest[pairs]]


LimitingResource[tableau_,var_] := Module[{ExpansionRatios},
    ExpansionRatios = (Rest[tableau] /.
{resource_ == (b_ ? NumberQ) + a_. var + x_:>
{resource,b/Abs[a]}/; a < 0,
 resource_ == a_. var + x_ :> {resource,0} /; a < 0,
 resource_ == x_  :> {resource,Infinity}});
    First @ MinPair[ExpansionRatios]
]
```

PROGRAMMING NOTE: This calculation is applied only to the constraints (Rest[tableau]). The usual convention in the linear programming literature is to put the objective function at the bottom of the tableau. We have reversed this convention, putting the objective at the top, to facilitate separating the objective function and the constraints when required.

For example, we found above that the first resource (machining) was the limiting factor in the production of bookcases, while labor is the critical resource in making desks.

```
LimitingResource[problem,x[b]]
```

```
s[m]
```

```
LimitingResource[problem,x[d]]
```

```
s[l]
```

We use **LimitingResource** to augment the function **Pivot** to find the critical resource limiting the extent of any potential improvement.

```
Pivot[tableau_?TableauQ,x_[j_]] :=
Pivot[tableau,x[j],LimitingResource[tableau,x[j]]]
```

In the production planning example, bookcases and desks are equally profitable. We arbitrarily chose to focus initially on bookcases. To illustrate the function **Pivot**, let us investigate what would have happened if we had made the other choice. Concentrating on the production of desks leads to

```
Pivot[problem, x[d]]
```

```
z == 25 - s[l] + 2 x[b] - x[c]

        65   s[l]   5 x[b]   4 x[c]
s[f] == -- + ---- - ------ - ------
        3     3       3        3

        25   s[l]   x[b]   2 x[c]
x[d] == -- - ---- - ---- - ------
        3     3      3       3

        35   s[l]   5 x[b]   x[c]
s[m] == -- + ---- - ------ - ----
        3     3       3       3
```

The revised objective function indicates that this is not optimal, some substitution of bookcases for desks is indicated.

```
Pivot[%,x[b]]
```

$$z == 39 - \frac{3\ s[l]}{5} - \frac{6\ s[m]}{5} - \frac{7\ x[c]}{5}$$

$$s[f] == 10 + s[m] - x[c]$$

$$x[d] == 6 - \frac{2\ s[l]}{5} + \frac{s[m]}{5} - \frac{3\ x[c]}{5}$$

$$x[b] == 7 + \frac{s[l]}{5} - \frac{3\ s[m]}{5} - \frac{x[c]}{5}$$

The top row indicates that this is an optimal solution and indeed it is the same solution as we arrived at earlier.

Now consider the situation if we begin with the production of chairs. Concentrating on the production of chairs yields a profit of only \$12.50.

Pivot[problem, x[c]]

$$z == \frac{25}{2} - \frac{s[l]}{2} + \frac{5\ x[b]}{2} + \frac{3\ x[d]}{2}$$

$$s[f] == 5 + s[l] - x[b] + 2\ x[d]$$

$$x[c] == \frac{25}{2} - \frac{s[l]}{2} - \frac{x[b]}{2} - \frac{3\ x[d]}{2}$$

$$s[m] == \frac{15}{2} + \frac{s[l]}{2} - \frac{3\ x[b]}{2} + \frac{x[d]}{2}$$

This tableau indicates that producing chairs alone does not use the available resources very efficiently. Producing some bookcases or desks would be profitable. Substituting some desks for chairs yields

Pivot[%, x[d]]

$$z == 25 - s[l] + 2\ x[b] - x[c]$$

$$s[f] == \frac{65}{3} + \frac{s[l]}{3} - \frac{5\ x[b]}{3} - \frac{4\ x[c]}{3}$$

$$x[d] == \frac{25}{3} - \frac{s[l]}{3} - \frac{x[b]}{3} - \frac{2\ x[c]}{3}$$

```
        35    s[l]    5 x[b]    x[c]
s[m] == -- + ---- - ------ - ----
        3     3       3         3
```

This looks better, but adding some bookcases would be even better. One more step gives the optimal solution.

```
Pivot[%,x[b]]
```

```
          3 s[l]    6 s[m]    7 x[c]
z == 39 - ------ - ------ - ------
            5         5         5

s[f] == 10 + s[m] - x[c]

           2 s[l]    s[m]    3 x[c]
x[d] == 6 - ------ + ---- - ------
             5        5        5

           s[l]    3 s[m]    x[c]
x[b] == 7 + ---- - ------ - ----
            5        5        5
```

From this experiment, it would seems that all paths of sequential improvement lead eventually to the optimal solution. Later, we will consider whether this is always true. Our experiment also reveals that some paths are quicker than others. Starting with bookcases or desks, we require two steps to reach the optimal solution. Beginning with chairs required three steps. The remaining component of the simplex algorithm is a criterion for guiding the process of sequential improvement efficiently.

5 The simplex algorithm

As we have just seen, the process of sequential improvement leading to an optimal solution can follow different paths. The basic simplex algorithm uses the shadow prices at each stage to guide the process of sequential improvement. The has proved to be an extremely efficient procedure in practice.

The shadow prices are the negatives of the coefficients of the variables in the revised objective function. Consequently, any variable which has a negative shadow price has a positive coefficient in the revised objective function and offers potential for improvement. Arguably, the variable with the lowest (most negative) shadow price offers the greatest potential for improvement.

The function **ShadowPrice** determines the shadow price of any given variable. **ShadowPrices** extracts the shadow prices from the revised objective function, returng a list of variables and their shadow prices. (Since the names are so similar, we reassure *Mathematica* that we have not made a spelling mistake.)

```
Off[General::spell1];
ShadowPrice[objective_,var_]:= -Coefficient[objective,var]
```

```
ShadowPrices[objective_] :=
    {#, ShadowPrice[objective,#]}& /@ Variables[objective]
On[General::spell1];
```

```
ShadowPrices[First[Pivot[problem,x[b]]][[2]]]
```

```
          3            1              3
{{s[m], -}, {x[c], -}, {x[d], -(-)}}
          2            2              2
```

Candidates for pivoting are those variables which have negative shadow prices. **NextPivot** identifies the that variable with the most negative shadow price, which becomes the pivot variable. It returns **None** if all shadow prices are positive. The function **Simplex** iterates the pivoting procedure until there is no further potential for improvement, stopping when there is no variable with a negative shadow price in the revised objective function. It returns the final tableau.

```
NextPivot[tableau_?TableauQ] := Module[
    {minshadow = MinPair[ShadowPrices[First[tableau][[2]]]]},
    If[minshadow[[2]] < 0,minshadow[[1]],None]
]
```

By augmenting the function pivot to select the next pivot when none is specified,

```
Pivot[tableau_?TableauQ] := Pivot[tableau,NextPivot[tableau]]
```

the problem can be solved by repeated application of the function **Pivot**.

```
Pivot @ Pivot[problem]
```

```
           3 s[l]   6 s[m]   7 x[c]
z == 39 - ------ - ------ - ------
             5        5        5

s[f] == 10 + s[m] - x[c]

             2 s[l]   s[m]   3 x[c]
x[d] == 6 - ------ + ---- - ------
               5       5       5

             s[l]   3 s[m]   x[c]
x[b] == 7 + ---- - ------ - ----
              5       5       5
```

Provided we provide an appropriate stopping rule, the simplex algorithm can be encoded surprisingly elegantly.

```
Pivot[tableau_?TableauQ,None] := tableau
```

```
Simplex[tableau_?TableauQ] := FixedPoint[Pivot,tableau]
```

```
finalTableau = Simplex[problem]
```

```
          3 s[l]   6 s[m]   7 x[c]
z == 39 - ------ - ------ - ------
            5        5        5

s[f] == 10 + s[m] - x[c]

             2 s[l]   s[m]   3 x[c]
x[d] == 6 - ------ + ---- - ------
               5       5       5

             s[l]   3 s[m]   x[c]
x[b] == 7 + ---- - ------ - ----
              5       5       5
```

We can easily equip **Simplex** to recognize alternative representations of the linear programming problem. For example, to match the calling sequence of **ConstrainedMax**, we provide

```
Simplex[objective_,constraints_] :=
Simplex[Prepend[StandardForm[constraints],
z == objective]]
```

The output of the function **Simplex** is the *final tableau*, comprising the revised objective function and constraints at the optimal solution. The final tableau is a very compact summary of useful information regarding the optimal solution of a linear programming problem. It tells us the optimal values of the decision variables, the amounts of unused resources and the optimal value of the objective function. It indicates whether the optimal solution is unique. It also includes the solution of a related linear programme, *the dual*, and yields a wealth of information about the sensitivity of the optimal solution to changes in the specification of the problem. Mining this information is the taken up in the following section on Sensitivity Analysis. First, we must confront some potential problems with our implementation and extend the algorithm to deal with minimization and other problems.

6 Potential problems

Our description of the simplex algorithm glossed over some potential problems which we now confront. Our algorithm began at a feasible solution, and then made a sequence of potential improvements until no further improvements could be made. Disregarding questions of efficiency, there are only two ways in which our procedure can fail to produce an optimal solution eventually. It will only fail if (1) there is no optimal solution or (2) the algorithm gets stuck in an infinite loop and fails to terminate. There are two reasons why an optimal solution may not exist - either the feasible set in empty or the feasible set is unbounded. We consider each of these possibilites in turn.

Before proceding, we need to augment the function **StandardForm** to provide default constraint labels when none are specified.

```
StandardForm[constraints_] :=
StandardForm[constraints, Range[Length[constraints]]]
```

6.1 Unboundedness

At each pivoting step, we presumed that the amount of potential improvement would be limited by some resource. If this is not the case, we could repeat that improvement and obtain an infinite return. For example consider the problem

$$\max x_1 + x_2$$

subject to

$$x[1] - x[2] \leq 0$$

Any nonnegative pair (x_1, x_2) is feasible provided $x_1 \leq x_2$. There is no limit to how big we can make the return. The feasible set is unbounded. Clearly there is no optimal solution.

To handle this possibility, we need to extend the function **LimitingResource** to recognise cases in which there is no critical resource and the feasible set is unbounded.

```
LimitingResource[tableau_,var_] := Module[{ExpansionRatios},
    ExpansionRatios = (Rest[tableau] /.
{resource_ == (b_ ? NumberQ) + a_. var + x_:>
{resource,b/Abs[a]}/; a < 0,
 resource_ == a_. var + x_ :> {resource,0} /; a < 0,
 resource_ == x_  :> {resource,Infinity}});
    If[#[[2]] < Infinity,#[[1]],None] & @
MinPair[ExpansionRatios]
]
```

```
problem = Prepend[StandardForm[{x[1] - x[2] <= 10}],
z == x[1] + x[2]]
```

```
z == x[1] + x[2]

s[1] == 10 - x[1] + x[2]
```

```
LimitingResource[problem,x[2]]
```

```
None
```

We must also amend the function **Pivot** to recognise when the problem is unbounded.

```
Pivot[tableau_?TableauQ,x_[j_],None] := (
Print["The problem is unbounded"];
Return[tableau])
```

```
Simplex[problem]
```

```
The problem is unbounded
```

```
z == 10 - s[1] + 2 x[2]

x[1] == 10 - s[1] + x[2]
```

6.2 Empty feasible set

The other reason why an optimal solution may fail to exist is that there may be no feasible solutions. In other words, the constraints are inconsistent. At first sight, this poses an insurmountable problem for applying the simplex algorithm, which requires an initial solution from which to make sequential improvements. In the preceding example, we used the trivial solution as a starting point. However, if the feasible set is empty, this is not feasible and hence not a valid starting point.

Fortunately, we can apply the simplex method to any problem to ascertain whether or not the feasible set is empty. Furthermore, this test also provides an initial feasible solution from which to initiate the search for an optimal solution.

Consider the following problem.

```
objective = x[1] - x[2] + x[3];
```

```
constraints =
{ 2x[1] -  x[2] + 2x[3] <=  4,
  2x[1] - 3x[2] +  x[3] <= -5,
  -x[1] +  x[2] - 2x[3] <= -1};
```

and $x_i >= 0$

```
problem=Prepend[StandardForm[constraints],
z == objective]
```

```
z == x[1] - x[2] + x[3]

s[1] == 4 - 2 x[1] + x[2] - 2 x[3]

s[2] == -5 - 2 x[1] + 3 x[2] - x[3]

s[3] == -1 + x[1] - x[2] + 2 x[3]
```

Note that the origin is not a feasible solution.

```
constraints /. {x[1] -> 0, x[2] -> 0, x[3] -> 0}
```

```
{True, False, False}
```

If there is no feasible solution, this is because the constraints are too restrictive. We investigate the consistency of the constraints by relaxing them until we find a feasible solution. We can ensure that the feasible set is non-empty by adding a sufficiently large positive quantity a_0 to the right hand side of the constraints.

```
Relax[ineq_,epsilon_] := ineq /.
{lhs_ <= rhs_ :> lhs<= rhs + epsilon,
 lhs_ == rhs_ :> lhs == rhs + epsilon}
```

```
Relax[constraints,a[0]]
```

```
{2 x[1] - x[2] + 2 x[3] <= 4 + a[0],

 2 x[1] - 3 x[2] + x[3] <= -5 + a[0],

 -x[1] + x[2] - 2 x[3] <= -1 + a[0]}
```

We can find the minimum value of a_0 which allows a feasible solution by solving the linear programming problem $\min_x a_0$ subject to the relaxed constraints. The original problem is feasible if and only if the minimum perturbation a_0 is zero. Moreover, the optimal solution to the relaxed problem is a feasible solution to the original problem, and provides a suitable starting point for the simplex algorithm.

The relaxed problem is

```
relaxed = Prepend[StandardForm[Relax[constraints,a[0]]],
z == -a[0]]
```

```
z == -a[0]

s[1] == 4 + a[0] - 2 x[1] + x[2] - 2 x[3]

s[2] == -5 + a[0] - 2 x[1] + 3 x[2] - x[3]

s[3] == -1 + a[0] + x[1] - x[2] + 2 x[3]
```

We can readily obtain a feasible solution to this problem by pivoting on the perturbation a_0. The most stringent constraint is the one with the most negative intercept. The intercepts of the constraints are

```
Intercept[form_] := If[NumberQ[form],form,0]

Intercept[form_Plus] := Intercept[First @ form]

Rest[relaxed] /. lhs_ == rhs_ :> {lhs,Intercept[rhs]}
```

```
{{s[1], 4}, {s[2], -5}, {s[3], -1}}
```

The smallest (most negative) intercept is -5 associated with the second constraint.

```
MinPair[%]
```

```
{s[2], -5}
```

A feasible solution to the relaxed constraints can be generated by pivoting on a_0 using the second constraint.

```
Pivot[relaxed,a[0],s[2]]
```

```
z == -5 - s[2] - 2 x[1] + 3 x[2] - x[3]

s[1] == 9 + s[2] - 2 x[2] - x[3]

a[0] == 5 + s[2] + 2 x[1] - 3 x[2] + x[3]

s[3] == 4 + s[2] + 3 x[1] - 4 x[2] + 3 x[3]
```

This indicates that the relaxed problem has a feasible solution with $a_0 = 5$ and $x_1 = x_2 = x_3 = 0$. Now we can apply the simplex algorithm to find the minimum feasible value of a_0.

```
phaseI = Simplex[%]
```

```
z == -a[0]

s[1] == 3 + 2 a[0] - s[3] - x[1]

        8   4 a[0]   s[2]   3 s[3]   x[1]
x[3] == - - ------ + ---- + ------ - ----
        5     5       5       5       5

        11   3 a[0]   2 s[2]   s[3]   3 x[1]
x[2] == -- - ------ + ------ + ---- + ------
        5      5        5       5       5
```

The minimum value of a_0 is zero, and is achieved where $x_2 = 11/5$ and $x_3 = 8/5$. This is a feasible solution to the original problem, which indicates that the feasible set is not empty and which can serve as the starting point for applying the simplex algorithm to solve the original problem.

The revised constraints (in standard form) are obtained by setting $a_0 = 0$ in the phase I tableau.

```
RevisedConstraints = Rest[phaseI] /. a[0]->0
```

```
s[1] == 3 - s[3] - x[1]

         8   s[2]   3 s[3]   x[1]
x[3] ==  - + ---- + ------ - ----
         5    5       5       5

         11   2 s[2]   s[3]   3 x[1]
x[2] ==  -- + ------ + ---- + ------
         5      5       5       5
```

which indicates the initial feasible solution. The revised objective function, updated to recognise the tradeoffs at the initial feasible solution is

```
RevisedObjective = objective /.
Flatten @ (ToRules /@ RevisedConstraints)
```

```
 3     s[2]   2 s[3]   x[1]
-(-) - ---- + ------ + ----
 5      5       5       5
```

Together, these give the initial tableau for the original problem, to which the simplex algorithm can be applied. This is called phase II.

```
InitialTableau = Prepend[RevisedConstraints,
z == RevisedObjective]
```

```
        3     s[2]   2 s[3]   x[1]
z == -(-) - ---- + ------ + ----
        5      5       5       5

s[1] == 3 - s[3] - x[1]

         8   s[2]   3 s[3]   x[1]
x[3] ==  - + ---- + ------ - ----
         5    5       5       5

         11   2 s[2]   s[3]   3 x[1]
x[2] ==  -- + ------ + ---- + ------
         5      5       5       5
```

```
Simplex[InitialTableau]

     3   2 s[1]   s[2]   x[1]
z == - - ------ - ---- - ----
     5     5       5      5

s[3] == 3 - s[1] - x[1]

        17   3 s[1]   s[2]   4 x[1]
x[3] == -- - ------ + ---- - ------
        5      5       5       5

        14   s[1]   2 s[2]   2 x[1]
x[2] == -- - ---- + ------ + ------
        5     5       5        5
```

The optimal value of 3/5 is obtained at $(0, 14/5, 17/5)$. We implement this two stage procedure in the function **LP**.

```
LP[tableau_?TableauQ] := Module[{},
    smallest = MinPair[Rest[tableau] /.
rhs_ == lhs_ :> {rhs,Intercept[lhs]}];
    If[smallest[[2]] >= 0,      (* origin feasible ? *)
Simplex[tableau],       (* single phase *)

                               (* 2 phase required *)
        relaxed = Prepend[Relax[Rest[tableau],a[0]],
  z == -a[0]];
phaseI = Simplex @
    Pivot[relaxed,a[0],smallest[[1]]];
        If[Intercept[First[phaseI][[2]]]==0, (* feasible? *)
                               (* phaseII *)
    RevisedConstraints = Rest[phaseI] /. a[0]->0;
    RevisedObjective = First[tableau][[2]] /.
        Flatten @ (ToRules /@ RevisedConstraints) //
ExpandAll;
    Simplex @ Prepend[RevisedConstraints,
z == RevisedObjective],
                               (* infeasible *)
Print["The problem is infeasible"]
     ]
    ]
]
```

```
LP[problem]

      3   2 s[1]   s[2]   x[1]
z == - - ------ - ---- - ----
      5     5        5      5

s[3] == 3 - s[1] - x[1]

        17   3 s[1]   s[2]   4 x[1]
x[3] == -- - ------ + ---- - ------
        5      5       5       5

        14   s[1]   2 s[2]   2 x[1]
x[2] == -- - ---- + ------ + ------
        5     5       5        5
```

As with Simplex, we can easily provide an alternative invocation for LP.

```
LP[objective_,constraints_] :=
LP[Prepend[StandardForm[constraints],
z == objective]]
```

In the following problem, the constraints are inconsistent and there is no feasible solution.

```
constraints = {x[1] - x[2] <= -1,
-x[1] - x[2] <= -33,
2x[1] + x[2] <= 2};
objective = 3x[1] + x[2];
```

```
LP[objective,constraints]
```

```
The problem is infeasible
```

Actually, we have brushed over another potential problem above. We have implicitly assumed that a_0 will be nonbasic at the solution of phase I. However, if phase I is degenerate, it is possible that a_0 remains basic with the value zero. This implies that, in the penultimate iteration, a_0 was equally eligible as some other variable as the "critical resource". We need to ensure that a_0 is always driven out of the basis first. We can do this by amending the function **LimitingResource** to select a_0 whenever it is one of the critical resources. This can be achieved elegantly by sorting the expansion ratios prior to selecting the minimum. This will ensure that the artificial variable a_0 always comes before any slack s_i or decision x_j variable.

```
LimitingResource[tableau_?TableauQ,var_] := Module[{ExpansionRatios},
    ExpansionRatios = (Rest[tableau] /.
{resource_ == (b_ ? NumberQ) + a_. var + x_:>
{resource,b/Abs[a]}/; a < 0,
 resource_ == a_. var + x_ :> {resource,0} /; a < 0,
 resource_ == x_  :> {resource,Infinity}});
    If[#[[2]] < Infinity,#[[1]],None] & @
MinPair[Sort @ ExpansionRatios]]
```

PROGRAMMING NOTE: The user should be aware that the natural order of variable names is assumed at this point and avoid any variables which are alphabetically prior to a_0.

6.3 Degeneracy and Cycling

Provided an optimal solution exists, is our algorithm guaranteed to find it? Not necessarily. In rare examples, it is possible for the algorithm we have described to become stuck in an infinite loop. Cycling can be prevented by changing the way in which the pivoting variable is selected (amending the function **NextPivot**), at the cost of slower convergence in normal problems (See for example Chvátal (1980). Cycling is such a rare phenomenon that most computer implementations ignore the possibility.[2]

There we have it. Barring rare examples, the simplex algorithm has implemented here will either find the optimal solution of a maximization problem or demonstrate that no solution exists. Furthermore, the procedure can easily be extended to deal with minimization problems and equality constraints.

6.4 Minimization problems

Since $\min_x f(x) = \max_x -f(x)$, any minimization can be converted into an equivalent maximization problem by reversing the sign of the objective function. Similarly, inequalities of the form $x \geq c$ can be converted into equivalent equations by subtracting a nonnegative surplus variable. For example, consider the problem of minimizing

```
objective =   30 x[1] + 25 x[2] + 20 x[3];
```

subject to

```
constraints = {2 x[1] +    x[2] +   2 x[3] >= 3,
        2 x[1] +  2 x[2] +    x[3] >= 1,
          x[1] +  3 x[2] +    x[3] >= 3};
```

This problem is the *dual* of the production planning example. It will be discussed further below.

We can easily extend the function **StandardForm** to transform $\geq$ inequalities

```
StandardForm[constraints_, labels_List] :=
Transpose @
   {labels,constraints} /.
{{i_, lhs_ <= rhs_}:> s[i] == rhs - lhs,
 {i_, lhs_ >= rhs_}:> s[i] == lhs - rhs}
```

```
StandardForm[constraints]
```

[2] With rational data, Mathematica uses exact or infinite precision arithmetic. This *may* increase the incidence of cycling, since rounding error is thought to mitigate the incidence of cycling when using real arithmetic.

```
s[1] == -3 + 2 x[1] + x[2] + 2 x[3]

s[2] == -1 + 2 x[1] + 2 x[2] + x[3]

s[3] == -3 + x[1] + 3 x[2] + x[3]
```

Applying the two phase procedure to the negative of the objective function, the solution of the dual problem is

```
LP[-objective,constraints]
```

```
z == -39 - 7 s[1] - 6 s[3] - 10 x[1]

         3   s[1]   2 s[3]
x[2] ==  - - ---- + ------
         5    5       5

         7   s[1]   3 s[3]
s[2] ==  - + ---- + ------ + x[1]
         5    5       5

         6   3 s[1]   s[3]
x[3] ==  - + ------ - ---- - x[1]
         5     5       5
```

Note that the optimal solution to the dual problem gives the final shadow prices of the production planning problem. The optimal value of the dual objective function, \$39, is equal to the maximum profit of the production planning problem. This is an illustration of the duality theorem of linear programming.

6.5 Equalities and Artificial Variables

Often, linear programming problems involves equations as well as inequalties. For example, consider the problem of maximizing

```
objective = x[1] + 3 x[2];
```

subject to two inequalities and two equations.

```
constraints =
{ x[1] +  x[2] +  x[3] <= 10,
 2x[1] +  x[2] - 2x[3] >= 2,
  x[1] + 2x[2]         == 4,
  x[1]         - 2x[3] == -2};
```

One way to handle such problems is to replace every equation with a pair of inequalities as follows

```
constraints /. lhs_ == rhs_ :> {lhs >= rhs,
lhs <= rhs} //
Flatten // TableForm
```

```
x[1] + x[2] + x[3] <= 10

2 x[1] + x[2] - 2 x[3] >= 2

x[1] + 2 x[2] >= 4

x[1] + 2 x[2] <= 4

x[1] - 2 x[3] >= -2

x[1] - 2 x[3] <= -2
```

and solve as before

```
LP[objective,%]
```

```
z == 4 - s[2] - 2 s[4] - s[6]

                      s[4]   5 s[6]
s[1] == 3 - 2 s[2] - ---- - ------
                       2       2

                   s[4]   3 s[6]
x[3] == 3 + s[2] + ---- + ------
                    2       2

x[1] == 4 + 2 s[2] + s[4] + 2 s[6]

s[3] == -s[4]

s[5] == -s[6]

x[2] == -s[2] - s[4] - s[6]
```

The optimal solution is $x_1 = 4, x_2 = 0, x_3 = 3$.

The drawback of this procedure is that it increases the dimensionality of the problem by introducing two slack variables for every equation. A more common approach to handling equations involves replacing these two slack variables with a single variable, called an *artificial variable*, which is allowed to adopt both positive and negative values. The artificial variable is driven to zero during the solution procedure. The variable a_0 introduced previously was an artificial variable. It would be relatively straightforward to amend **LP** to handle additional artificial variables. However, the effort is probably not warranted as we will later show how to take advantage of *Mathematica*'s inbuilt linear programming facility to obtain the final tableau of a linear programming problem.

Instead, we extend the function **StandardForm** to treat equations as above.

```
StandardForm[constraints_, labels_List] :=
    Transpose @ {labels,constraints} /.
    {{i_, lhs_ <= rhs_}:> s[i] == rhs - lhs,
    {i_, lhs_ >= rhs_}:> s[i] == lhs - rhs,
    {i_, lhs_ == rhs_}:> {sl[i] == rhs - lhs,
  sg[i] == lhs - rhs}} // Flatten
```

```
StandardForm[constraints]
```

```
s[1] == 10 - x[1] - x[2] - x[3]

s[2] == -2 + 2 x[1] + x[2] - 2 x[3]

sl[3] == 4 - x[1] - 2 x[2]

sg[3] == -4 + x[1] + 2 x[2]

sl[4] == -2 - x[1] + 2 x[3]

sg[4] == 2 + x[1] - 2 x[3]
```

```
LP[objective,constraints]
```

```
z == 4 - s[2] - 2 sl[3] - sl[4]

                         sl[3]   5 sl[4]
s[1] == 3 - 2 s[2] - ----- - -------
                           2         2

                     sl[3]   3 sl[4]
x[3] == 3 + s[2] + ----- + -------
                       2         2

sg[3] == -sl[3]

x[1] == 4 + 2 s[2] + sl[3] + 2 sl[4]

x[2] == -s[2] - sl[3] - sl[4]

sg[4] == -sl[4]
```

7 Sensitivity Analysis

The data in any real optimization problem are seldom known with absolute precision. Consequently, it is important to be able to estimate the sensitivity of the optimal solution to changes in the specification of the problem. Fortunately, it is possible to deduce a great deal about the sensitivity of the optimal solution to a linear programming problem from the final tableau. This is known as *sensitivity analysis.*

In the next subsection, we explore the anatomy of the final tableau, outlining the range of information which it embodies. We also show how this information is related to the solution of a related linear programming problem, which is called the *dual*. In following subsections, we analyze more systematically the sensitivity of the optimal solution to changes in the parameters of the problem. First, we elaborate the role of shadow prices in the analysis of changes in resource availability and derive bounds on their validity. Then, we explore the sensitivity of the optimal solution to changes in the objective function.

Although the previous instalment outlined a complete implementation of the simplex algorithm, it was intended for pedagogical rather than computational purposes. *Mathematica*'s built-in linear programming facility is significantly more efficient at solving substantive problems, but it's output is too concise to allow for sensitivity analysis. Therefore, in Section 4, we show how the final tableau can be reconstructed from the output of Mathematica's built-in linear programming facility. The paper concludes with a sensitivity analysis of a classic problem of nutrition, which was first posed by George Stigler before the simplex algorithm was available.

7.1 Interpreting the final tableau

The production planning example involved maximizing the function

```
profit = 3x[b] + x[c] + 3x[d];
```

subject to the constraints

```
constraints =
{ 2x[b] + 2x[c] +  x[d] <= 30,
   x[b] + 2x[c] + 3x[d] <= 25,
  2x[b] +  x[c] +  x[d] <= 20
  };
```

The optimal solution can be computed using the function **LP**, which yields the final tableau.

```
finalTableau = LP[profit,
StandardForm[constraints,{f,l,m}]]
```

```
            3 s[l]   6 s[m]   7 x[c]
z == 39 - ------ - ------ - ------
             5        5        5
```

```
s[f] == 10 + s[m] - x[c]

               2 s[l]    s[m]    3 x[c]
x[d] == 6 - ------ + ---- - ------
                 5        5        5

             s[l]    3 s[m]    x[c]
x[b] == 7 + ---- - ------ - ----
               5        5        5
```

The final tableau not only indicates the optimal solution to a particular problem. It contains a wealth of information regarding the sensitivity of the optimal solution to changes in the parameters of the problem. It indicates whether the optimal solution is unique, and implicitly contains the solution of a host of related linear programming problems. In this section, we explore the range of information contained in the final tableau.

Those variables which appear on the left hand of the final tableau are called *basic* variables. The right hand side variables are the *non-basic* variables.

```
BasicVariables[tableau_?TableauQ] := First /@ Rest[tableau]

basic = BasicVariables[finalTableau]

{s[f], x[d], x[b]}

NonBasicVariables[tableau_] := Union @@
(Variables[#[[2]]]& /@ tableau)

nonbasic=NonBasicVariables[finalTableau]

{s[l], s[m], x[c]}
```

The nonbasic variables are all zero at the optimal solution. Each belongs to a binding constraint - either a resource constraint ($s_i = 0$) or a nonnegativity constraint ($x_j = 0$). The basic variables are typically positive at the optimal solution. If not, that is if one or more of the basic variables are zero, the solution is called *degenerate*. A degenerate solution indicates that one or more of the constraints are redundant in the sense that they could be relaxed without changing the solution. The values of the objective function and the basic variable can be highlighted by evaluating the final tableau with the nonbasic variables set to zero.

```
SetAttributes[Zero,Listable];
Zero[x_] := x -> 0

solution = finalTableau /. Zero[nonbasic]


z == 39

s[f] == 10

x[d] == 6

x[b] == 7
```

The optimal plan produces 7 bookcases and 6 desks for a total profit of \$39. The optimal values of the slack variable s_i measure the utilization of resources. In this case, the optimal plan leaves spare finishing capacity of 10 units, but fully utilizes all the labor and machining capacity (since $s_l = s_m = 0$).

The first row of the final tableau is the revised objective function, which embodies the economic tradeoffs which pertain at the optimal solution. The coefficients of the nonbasic variables indicate the shadow prices of the binding constraints.

```
RevisedObjective = First[finalTableau]
```

```
            3 s[l]   6 s[m]   7 x[c]
z == 39 - ------ - ------ - ------
             5        5        5
```

We augment the definition of **ShadowPrices** to extract the shadow prices of the nonbasic variables from the final tableau.

```
ShadowPrices[tableau_?TableauQ] :=
ShadowPrices[First[tableau][[2]]]
```

```
ShadowPrices[finalTableau]
```

```
         3          6          7
{{s[l], -}, {s[m], -}, {x[c], -}}
         5          5          5
```

Shadow prices measures the economic consequences of marginal changes in resources. The shadow prices of basic variables are always zero.

```
ShadowPrice[tableau_?TableauQ,var_] :=
ShadowPrice[First[tableau][[2]],var]
```

```
ShadowPrice[finalTableau,s[f]]
```

```
0
```

To determine the shadow prices of a set of variables, we define

```
ShadowPrices[tableau_?TableauQ,vars_List] :=
{#,ShadowPrice[First[tableau][[2]],#]}& /@ vars
```

For example, the shadow prices of the resource constraints are

```
ShadowPrices[finalTableau,{s[f],s[l],s[m]}]
```

```
                    3           6
{{s[f], 0}, {s[l], -}, {s[m], -}}
                    5           5
```

The significance of the shadow prices is that they enable us to evaluate the impact hypothetical changes in the parameters of the problem without solving the problem afresh. The reason was elucidated in the discussion of the simplex algorithm. The revised objective function fully embodies the economic ramifications of all the technological constraints viewed from the perspective of the optimal solution. In this sense, the final tableau summarizes the solution of a whole family of related optimization problems. We elaborate this point in the following sections.

The changes in resource constraints may not be hypothetical. The marginal value of an additional hour of labor supply is \$3/5. Consequently, if additional labor can be obtained for less than \$3/5 an hour, it would be profitable to purchase additional labor. Similarly, if machining capacity can be let for more than \$6/5, the furniture maker would increase total profit by reducing her own production and leasing some capacity. Shadow prices are the imputed values of the resources in terms of the objective function, and can be directly compared with market prices. The producer should buy those resources whose shadow price (value to her) is greater than the market price and sell those resources which the market values more highly than her own production opportunities.

The shadow prices can also be used to evaluate the profitability of new activities. Suppose that the furniture maker is contemplating adding a new product, an executive desk, to her range. Each executive desk would require 2 hours each of machining and finishing and 3 hours of labor. The economic cost of producing an executive desk (with fixed resources) is the value of the displaced production of bookcases and ordinary desks, which can be measured by the shadow prices of these resources. This cost is

$$\text{Cost of executive desk} = 2(0) + (\frac{3}{5}) + 2(\frac{6}{5}) = \frac{21}{5}$$

The executive desk will only be profitable if it can be sold for at least $21/5 =$ \$4.20.

The shadow prices attached to the resource constraints are in fact the solutions of a related linear programming problem called the *dual*. The dual of the production planning example poses the problem of finding the minimum price at which the furniture manufacturer would be willing to sell the her resources (finishing, machining and labor) given that their opportunity cost is determined their alternative use in the production of bookcases, chairs and desks for sale.

Formally specified, the dual of the production planning example is

$$\begin{aligned} &\min 30x[1] + 25x[2] + 20x[3] \\ \text{subject to } &2x[1] + x[2] + 2x[3] \geq 3 \\ &2x[1] + 2x[2] + x[3] \geq 1 \\ &x[1] + 3x[2] + x[3] \geq 3 \end{aligned}$$

The solution represent the imputed or shadow prices of the resource. We solved this problem using **LP** in the section of minimization problems above, obtaining the solution $x_1 = 0, x_2 = 3/5, x_3 = 6/5$. The shadow prices of finishing, machining and labor are 0, 3/5 and 6/5 respectively, which exactly correspond to the shadow prices derived from the original production planning example (the *primal*). Furthermore, we note that the minimal value obtained for dual problem was \$39, which is exactly equal to the maximal profit obtained for the primal problem.

These observations illustrate the fundamental *duality theorem* of linear programming. If the primal linear programming problem has an optimal solution, then so does its dual and the optimal value of the objective function is the same for both problems. Furthermore, the optimal values of the decision variables in the dual problem are equal to the shadow prices of the resource constraints at the optimal solution of the primal problem. Therefore, the final tableau for the primal problem includes the optimal solution of the related dual problem.

At the optimal solution, shadow prices are always nonnegative. (Recall that the simplex algorithm terminates when the shadow prices are nonnegative. A negative shadow prices indicates the possibility of a profitable substitution, by tightening a binding constraint.) The shadow prices of basic variables are always zero. However, it is not necessary that the shadow prices of nonbasic variables are strictly positive. A zero shadow price on a nonbasic variable indicates that the optimal solution is not unique, since that variable can be "pivoted" into the solution without changing the value of the objective function.

To illustrate, suppose the price of chairs is increased to $2\frac{2}{5}$. The original production plan of 7 bookcases and 6 desks remains optimal.

```
Simplex[3 x[b] + (12/5) x[c] + 3 x[d],
StandardForm[constraints,{f,l,m}]]
```

```
             3 s[l]   6 s[m]
z == 39 - ------ - ------
               5        5

s[f] == 10 + s[m] - x[c]

              2 s[l]    s[m]    3 x[c]
x[d] == 6 - ------ + ---- - ------
                5        5        5

              s[l]    3 s[m]    x[c]
x[b] == 7 + ---- - ------ - ----
               5        5        5
```

However, the absence of the x_c from the revised objective function indicates that there are multiple optimal solutions. At this price, desks and chairs are equally profitable. Substituting chairs for desks reveals another optimal solution, producing 5 bookcases and 10 chairs. This substitution can be made by pivoting.

```
Pivot[%,x[c]]
```

```
           3 s[l]   6 s[m]
z == 39 - ------ - ------
             5        5

x[c] == 10 - s[f] + s[m]

         3 s[f]   2 s[l]   2 s[m]
x[d] == ------ - ------ - ------
           5        5        5

              s[f]   s[l]   4 s[m]
x[b] == 5 + ---- + ---- - ------
              5      5        5
```

This is also an example of degenerate optimal solution, in which the basic variable x_d is zero.

```
% /. Zero[NonBasicVariables[%]]
```

```
z == 39

x[c] == 10

x[d] == 0

x[b] == 5
```

Any convex combination of this solution and the previous solution will also be optimal.

In summary, the intercepts of the equations in the final tableau indicate the optimal value of the objective function and the values of the basic variables at which the optimum is obtained. The coefficients of the nonbasic variables in the first equation (the revised objective function) are the shadow prices, which indicate the economic consequences of marginal changes in the quantity of resources. Nonbasic variables with zero shadow prices indicate multiple optimal. As we will show below, the coefficients of the nonbasic variables in the remaining equations (the revised constraints) indicate how these consequences are obtained.

The final tableau is a very compact representation of the optimal solution to a linear programming problem. It embodies all the economic and technological tradeoffs which are pertinent at the optimal solution. It lists the maximal value of the objective function, and the values of the decision variables necessary to achieve this maximum. It also reveals the degree of utilization of resources, highlighting any unused capacity. It indicates whether or not the optimal solution is unique. The shadow prices appraise the imputed values of limited resources, assessing the profitability of any sale or purchase of additional resources. The tableau also indicates the optimal response to any change in resources. In this sense, it summarizes the solution to a whole family of related optimization problems. It is truly pregnant with information.

For future use, it is convenient to represent the final tableau as a set of transformation rules.

```
rules = ToRules /@ Rest[finalTableau] // Flatten
```

```
{s[f] -> 10 + s[m] - x[c],

          2 s[l]    s[m]    3 x[c]
x[d] -> 6 - ------ + ---- - ------,
            5        5       5

            s[l]    3 s[m]    x[c]
x[b] -> 7 + ---- - ------ - ----}
            5        5       5
```

7.2 Sensitivity with respect to resources

In this subsection, we examine more closely the sensitivity of the optimal solution to changes in the constraints. The shadow prices measure the impact on profit on changes in resource availability. For example, the shadow price of 3/5 on the labor constraint means that an additional hour of labor would increase profit by $3/5. To verify this, let us re-solve the production planning problem with an additional 5 hours of labor.

```
Simplex[profit,StandardForm[
{ 2x[b] + 2x[c] +  x[d] <= 30,
    x[b] + 2x[c] + 3x[d] <= 30,
  2x[b] +  x[c] +  x[d] <= 20
  },{f,l,m}]] /. Zero[nonbasic]
```

```
z == 42

s[f] == 10

x[d] == 8

x[b] == 6
```

An additional 5 hours of labor would allow the furniture maker produce 6 bookcases and 8 desks for a total profit of $42. Compared to the previous plan of 7 bookcases and 6 desks, this involves substituting two desks for a bookcase, increasing total profit by $3. Each additional hour of labor is worth $3/5.

Similarly, an additional 5 units of machining time would allow her to substitute 3 bookcases for a desk, increasing total profit by $6. Additional units of machining are worth $6/5 each. Again, this can be confirmed by re-solving the problem.

```
Simplex[profit,StandardForm[
{ 2x[b] + 2x[c] +  x[d] <= 30,
     x[b] + 2x[c] + 3x[d] <= 25,
  2x[b] +  x[c] +  x[d] <= 25
  },{f,l,m}]] /. Zero[nonbasic]
```

```
z == 45

s[f] == 5

x[d] == 5

x[b] == 10
```

The significance of the shadow price is that it indicates the impact on profit of a change in resources without resolving the problem. We can also deduce how this is achieved, by evaluating the final tableau while allowing the appropriate slack variable to deviate from zero.

For example, increasing the labor resource is equivalent to allowing the slack variable s_l to assume negative values. Letting $\Delta_l = -s_l$ denote the increase in labor resources, we find that

```
finalTableau /. s[l] -> -Delta[l] /. Zero[nonbasic]
```

```
          3 Delta[l]
z == 39 + ----------
              5

s[f] == 10

            2 Delta[l]
x[d] == 6 + ----------
                5

            Delta[l]
x[b] == 7 - --------
               5
```

This indicates that each additional unit of labor will increase profit by \$3/5, increase the number of desks by 2/5 and decrease the number of bookcases by 1/5. Consequently, an additional 5 hours of labor would allow

```
% /. Delta[l] -> 5
```

```
z == 42
```

```
s[f] == 10

x[d] == 8

x[b] == 6
```

which is consistent with the result obtained above by re-solving the production planning problem with labor supply equal to 30 hours. In this way, we can estimate the impact of hypothetical changes in resources without re-solving the problem. For large problems, this can represent a considerable saving in computation.

Typically, shadow prices have a limited range of validity, which can also be deduced from the final tableau. Just above, we assessed the impact on the basic variables of small changes in labor supply, namely

```
Rest[finalTableau] /. s[l] -> -Delta[l] /. Zero[nonbasic]
```

```
s[f] == 10

                2 Delta[l]
x[d] == 6 + ----------
                    5

                Delta[l]
x[b] == 7 - --------
                  5
```

An *optimal* response to an increase in labor supply $\Delta_l > 0$ has no effect on the utilization of finishing capacity, and requires the substitution of desks for bookcases. We observe that this substitution can profitably continue until the quantity of bookcases is reduced to zero, that is while $\Delta_l \leq 35$. Similarly, an optimal response to a decrease in labor supply ($\Delta_l < 0$) requires substituting bookcases for desks, which substitution can profitably continue until it reduces the production of desks to zero, that is while $\Delta_l \geq -15$. This establishes the range over which the shadow price of labor supply is valid, namely over the range $-15 \leq \Delta_l \leq 35$, where Δ_l is the change in the supply of labor.

Consider a similar calculation for the machining constraint. The optimal response is for small changes

```
Rest[finalTableau] /. s[m] -> -Delta[m] /.
 Zero[nonbasic]
```

```
s[f] == 10 - Delta[m]

                Delta[m]
x[d] == 6 - --------
                  5
```

```
              3 Delta[m]
x[b] == 7 + ----------
                 5
```

This response will remain optimal provided it does not violate any of other (resource and nonnegativity) constraints, that is provided the basic variables remain non-negative.

```
bounds = Cases[%,
_ == b_  + a_. Delta[m]  :>
Sign[a] Delta[m] >= -b/Abs[a] ]
```

```
                                                                 35
{-Delta[m] >= -10, -Delta[m] >= -30, Delta[m] >= -(--)}
                                                                 3
```

Using the following function to simplify the inequalities

```
SimplifyInequalities[ineq_List,var_] :=
Max @ Join[ Cases[ineq, var >= a_ -> a],
Cases[ineq, -var <= a_ -> -a]] <=
var <=
Min @ Join[ Cases[ineq, var <= a_ -> a],
Cases[ineq, -var >= a_ -> -a]]
```

the valid range is

```
SimplifyInequalities[bounds,Delta[m]]
```

```
  35
-(--) <= Delta[m] <= 10
  3
```

We summarize this facility in the function ShadowPriceRange.

```
ShadowPriceRange[tableau_?TableauQ,s_[i_]] :=
    SimplifyInequalities[
        Select[Rest[tableau] /. s[i] -> -Delta[i] /.
Zero[NonBasicVariables[tableau]],
   !FreeQ[#,Delta[i]]&] /.
    _ == b_  + a_. Delta[i]  :>
     Sign[a] Delta[i] >= -b/Abs[a],
Delta[i]]
```

```
ShadowPriceRange[finalTableau,s[1]]
```

```
-15 <= Delta[1] <= 35
```

```
ShadowPriceRange[finalTableau,s[m]]
```

```
  35
-(--) <= Delta[m] <= 10
  3
```

The preceding applies to the shadow prices of binding constraints. For non-binding constraints (basic variables) , the determination of the range of validity of the shadow prices is more straightforward. A surplus resource has a shadow price of zero, which is the imputed value of additional resources. The shadow price will remain zero as long as the constraint remains nonbinding, that is provided the slack capacity is not reduced to zero.

In the production planning example, there is surplus finishing capacity of 10 units.

```
Cases[Rest[finalTableau] /. s[f] -> -Delta[f] /.
Zero[nonbasic],
-Delta[f] == rhs_ :>  Delta[f] >= -rhs]
```

```
{Delta[f] >= -10}
```

```
SimplifyInequalities[%,Delta[f]]
```

```
-10 <= Delta[f] <= Infinity
```

Consequently, the shadow price of finishing capacity will remain zero for all increases in finishing capacity and any decreases in capacity of less than 10 units. That is, range of validity of the shadow price of finishing capacity is $-10 \leq \Delta_f \leq \infty$. We extend the function **ShadowPriceRange** to deal with basic variables.

```
ShadowPriceRange[tableau_?TableauQ,s_[i_]] :=
    SimplifyInequalities[
        Select[Rest[tableau] /. s[i] -> -Delta[i] /.
Zero[NonBasicVariables[tableau]],
   !FreeQ[#,Delta[i]]&] /.
{-Delta[i] == b_ :> Delta[i] >= -b,  (* basic *)
          _ == b_  + a_. Delta[i]  :> (* nonbasic *)
     Sign[a] Delta[i] >= -b/Abs[a]},
Delta[i]]
```

```
ShadowPriceRange[finalTableau,s[f]]
```

```
-10 <= Delta[f] <= Infinity
```

The function **SensitivityAnalysis** computes the shadow price and range of a given variable.

```
SensitivityAnalysis[tableau_?TableauQ,s_[i_]] :=
{s[i],ShadowPrice[tableau,s[i]],
ShadowPriceRange[tableau,s[i]]}
```

```
SensitivityAnalysis[finalTableau,s[f]]
```

```
{s[f], 0, -10 <= Delta[f] <= Infinity}
```

Provided we adhere to the convention of using s_i to denote slack variables, we can extend the domain of **SensitivityAnalysis** to compute the shadow prices and ranges of all slack variables.

```
SensitivityAnalysis[tableau_?TableauQ] :=
SensitivityAnalysis[tableau,#]& /@
Cases[Join[BasicVariables[tableau],
   NonBasicVariables[tableau]],s[i_]]
```

```
SensitivityAnalysis[finalTableau] // TableForm
```

```
s[f]    0    -10 <= Delta[f] <= Infinity

        3
        -
s[l]    5    -15 <= Delta[l] <= 35

        6      35
        -    -(--) <= Delta[m] <= 10
s[m]    5      3
```

7.3 Sensitivity with respect to prices

In the production planning example, no chairs are produced because they are not sufficiently profitable. By how much would the selling price of chairs have to rise to make their production worthwhile? We can address this question by varying the price of chairs. For example, if the price of chairs becomes $1 + \Delta_c$, the profit function is

```
Deltaprofit = 3 x[b] + (1 + Delta[c]) x[c] + 3 x[d]
```

```
3 x[b] + (1 + Delta[c]) x[c] + 3 x[d]
```

At the current optimal production plan, the revised objective function is

```
RevisedObjective = Collect[Deltaprofit /. rules,nonbasic]
```

```
      3 s[l]    6 s[m]        7
39 - ------ - ------ + (-(-) + Delta[c]) x[c]
        5         5           5
```

This reveals that production of chairs will not be profitable provided the coefficient of x_c is negative. Alternatively, production of chairs is not profitable provided the shadow price of chairs is positive, that is $\Delta_c \leq 7/5$

```
ShadowPrice[RevisedObjective,x[c]]
```

```
7
- - Delta[c]
5
```

The price of chairs would have to rise to $1 + \frac{7}{5} = 2\frac{2}{5}$ before their production is profitable.

We can verify this conclusion by solving the production planning problem with varying prices for chairs. For example, doubling the price of chairs does not change the optimal solution.

```
Simplex[3 x[b] + 2 x[c] + 3 x[d],
    StandardForm[constraints,{f,l,m}]]
```

```
           3 s[l]   6 s[m]   2 x[c]
z == 39 - ------ - ------ - ------
             5        5        5

s[f] == 10 + s[m] - x[c]

              2 s[l]    s[m]   3 x[c]
x[d] == 6 - ------ + ---- - ------
                5        5       5

             s[l]   3 s[m]   x[c]
x[b] == 7 + ---- - ------ - ----
              5       5       5
```

However, at \$3, chairs become more profitable than desks. Again, the solution is degenerate.

```
Simplex[3 x[b] + 3 x[c] + 3 x[d],
StandardForm[constraints,{f,l,m}]]
```

$$z == 45 - \frac{3\ s[f]}{5} - \frac{3\ s[l]}{5} - \frac{3\ s[m]}{5}$$

$$x[c] == 10 - s[f] + s[m]$$

$$x[d] == \frac{3\ s[f]}{5} - \frac{2\ s[l]}{5} - \frac{2\ s[m]}{5}$$

$$x[b] == 5 + \frac{s[f]}{5} + \frac{s[l]}{5} - \frac{4\ s[m]}{5}$$

Similarly, we might ask by how much the price of desks would have to fall to render their production unprofitable. Letting Δ_d denote changes in the price of desks, the profit function is

```
Deltaprofit = 3 x[b] + x[c] + (3 + Delta[d]) x[d]
```

```
3 x[b] + x[c] + (3 + Delta[d]) x[d]
```

and the revised objective function is

```
RevisedObjective = Collect[Deltaprofit /. rules,nonbasic]
```

$$39 + 6\ Delta[d] + \left(-\left(\frac{3}{5}\right) - \frac{2\ Delta[d]}{5}\right) s[l] + \left(-\left(\frac{6}{5}\right) + \frac{Delta[d]}{5}\right) s[m] +$$

$$\left(-\left(\frac{7}{5}\right) - \frac{3\ Delta[d]}{5}\right) x[c]$$

```
ShadowPrices[RevisedObjective]
```

$$\{\{Delta[d], -6\}, \{s[l], \frac{3}{5} + \frac{2\ Delta[d]}{5}\}, \{s[m], \frac{6}{5} - \frac{Delta[d]}{5}\},$$

$$\{x[c], \frac{7}{5} + \frac{3\ Delta[d]}{5}\}\}$$

The first component $(\Delta_d, 6)$ indicates the direct effect on profit of a change in the price of desks. At the optimal production of 6 desks, every one dollar increase in the price of desks increases profit by \$6. The remaining terms can be interpreted as follows. Whatever the change Δ_d in the price of desks, the production plan $(7,0,6)$ remains optimal provided all these shadow prices are positive. Inspection reveals this will be true provided $-3/2 \leq \Delta_d \leq 6$. Let us programme that evaluation.

First, we select those shadow prices which depend upon Δ_d and convert them into inequalites, which can be simplified by the function **SimplifyInequalities**.

```
Cases[ShadowPrices[RevisedObjective],
{_,b_  + a_. Delta[d]}  :> Sign[a] Delta[d] >= -b/Abs[a] ]
```

```
                  3                                       7
{Delta[d] >= -(-), -Delta[d] >= -6, Delta[d] >= -(-)}
                  2                                       3
```

```
SimplifyInequalities[%,Delta[d]]
```

```
  3
-(-) <= Delta[d] <= 6
  2
```

To determine the sensitivity to prices changes, we have to know the original objective function, which was

```
objective = 3 x[b] + x[c] + 3 x[d]
```

```
3 x[b] + x[c] + 3 x[d]
```

We summarize this facility in the function **PriceRange**.

```
PriceRange[tableau_?TableauQ,objective_,x_[i_]] :=
Module[{RevisedObjective},
RevisedObjective =
Collect[objective /.
a_. x[i] -> (a + Delta[i]) x[i] /.
(ToRules /@ Rest[tableau] // Flatten),
NonBasicVariables[tableau]];
SimplifyInequalities[Cases[ShadowPrices[RevisedObjective],
{_,b_  + a_. Delta[i]}  :> Sign[a] Delta[i] >= -b/Abs[a] ],
Delta[i]]
]
```

```
PriceRange[finalTableau,objective,x[d]]
```

```
  3
-(-) <= Delta[d] <= 6
  2
```

To verify this conclusion, we observe that increasing the price of desks by \$5 to \$8 does not change the optimal production plan.

```
Simplex[3 x[b] +  x[c] + 8 x[d],
   StandardForm[constraints,{f,l,m}]]
```

```
          13 s[l]   s[m]   22 x[c]
z == 69 - ------- - ---- - -------
             5        5       5

s[f] == 10 + s[m] - x[c]

            2 s[l]   s[m]   3 x[c]
x[d] == 6 - ------ + ---- - ------
              5       5       5

            s[l]   3 s[m]   x[c]
x[b] == 7 + ---- - ------ - ----
             5       5       5
```

However, if the price of desks falls by 50%, it is just as profitable to concentrate on bookcases alone.

```
Simplex[3 x[b] +  x[c] +  (3/2) x[d],
    StandardForm[constraints,{f,l,m}]]
```

```
          3 s[m]   x[c]
z == 30 - ------ - ----
            2       2

s[f] == 10 + s[m] - x[c]

             s[m]   3 x[c]   5 x[d]
s[l] == 15 + ---- - ------ - ------
              2       2        2

             s[m]   x[c]   x[d]
x[b] == 10 - ---- - ---- - ----
              2      2      2
```

although the previous plan is equally profitable.

```
Pivot[%,x[d]]
```

```
          3 s[m]   x[c]
z == 30 - ------ - ----
            2       2

s[f] == 10 + s[m] - x[c]

            2 s[l]   s[m]   3 x[c]
x[d] == 6 - ------ + ---- - ------
              5       5       5
```

```
                s[l]    3 s[m]    x[c]
x[b] == 7 + ---- - ------ - ----
                 5        5        5
```

At a lower price, desks are not profitable.

In line with the facilities developed in the previous section, we extend the function **SensitivityAnalysis** to tabulate the price and applicable range of any decision variable x_i.

```
SensitivityAnalysis[tableau_?TableauQ,objective_Plus,x_[i_]] :=
{x[i], Coefficient[objective,x[i]],
PriceRange[tableau,objective,x[i]]}
```

```
SensitivityAnalysis[finalTableau,objective,x[d]]
```

```
               3
{x[d], 3, -(-) <= Delta[d] <= 6}
               2
```

Its domain can then be extended to cover all the variables in the problem.

```
SensitivityAnalysis[tableau_?TableauQ,objective_Plus] :=
Join[SensitivityAnalysis[tableau,objective,#]& /@
Variables[objective],
     SensitivityAnalysis[tableau,#]& /@
Cases[Join[BasicVariables[tableau],
   NonBasicVariables[tableau]],s[i_]]]
```

The function **SensitivityAnalysis** extracts from the final tableau the prices associated with each decision variable, the shadow price of each resource constraint and their respective ranges of validity.

```
SensitivityAnalysis[finalTableau,objective] // TableForm
```

```
x[b]    3    -2 <= Delta[b] <= 3

                                         7
             -Infinity <= Delta[c] <= -
x[c]    1                                5

               3
             -(-) <= Delta[d] <= 6
x[d]    3      2
```

```
s[f]    0    -10 <= Delta[f] <= Infinity

        3
        -
s[l]    5    -15 <= Delta[l] <= 35

        6      35
        -    -(--) <= Delta[m] <= 10
s[m]    5      3
```

This does not exhaust the information which can potentially be extracted from the final tableau. Before leaving this topic, we shall briefly outline some of the ways in which sensitivity analysis could he extended. We could compute the impact of changes in the technology, that is in the resource coefficients of the various activities. For example, what would be the effect on the optimal solution if the labour required for producing a desk was reduced from 3 to 2 hours? We could also consider the impact of multiple changes. What would be the impact of a simultaneous rise in the price of desks and chairs? What would be effect of adding an additional shift, increasing finishing, machining and labour capacity simultaneously? All these questions could be addressed utilizing information in the final tableau.

8 Computing the final tableau

Although the preceding discussion presents a complete implementation of the simplex algorithm for solving linear programming problems, it is not the most efficient way of doing so. Fortunately, *Mathematica* contains built-in functions for linear programming — **ConstrainedMax**, **ConstrainedMin** and **LinearProgramming**. However, the output from these built-in functions is brief and concise. In this section, we show how the output of the built-in functions can be mined to yield the final tableau and hence all the sensitivity information which can be deduced from this tableau.

The usage message for **ConstrainedMax** reads

```
?ConstrainedMax
```

```
ConstrainedMax[f, {inequalities}, {x, y, ...}] finds the global
   maximum of f in the domain specified by the inequalities. The
   variables x, y, ... are all assumed to be non-negative.
```

Let us apply this function to the production planning example which involved maximizing

```
profit = 3x[b] + x[c] + 3x[d];
```

subject to the

```
constraints =
{ 2x[b] + 2x[c] +  x[d] <= 30,
   x[b] + 2x[c] + 3x[d] <= 25,
  2x[b] +  x[c] +  x[d] <= 20
  };
```

and $x_i >= 0$

In reconstructing the final tableau, it is more convenient to apply **ConstrainedMax** to the standard form.

```
SFconstraints = StandardForm[constraints,{f,l,m}]
```

```
s[f] == 30 - 2 x[b] - 2 x[c] - x[d]

s[l] == 25 - x[b] - 2 x[c] - 3 x[d]

s[m] == 20 - 2 x[b] - x[c] - x[d]
```

To find the variables, we extend the definition of the built-in function **Variables** to handle equations.

```
Unprotect[Variables];
Variables[equation_Equal] := Variables[List @@ equation]
Protect[Variables];
```

The variables are

```
vars = Union @@ Variables /@ SFconstraints
```

```
{s[f], s[l], s[m], x[b], x[c], x[d]}
```

Applying **ConstrainedMax** to the production planning example yields

```
solution=ConstrainedMax[profit, SFconstraints,vars]
```

```
{39, {s[f] -> 10, s[l] -> 0, s[m] -> 0, x[b] -> 7, x[c] -> 0,

  x[d] -> 6}}
```

The output of **ConstrainedMax** is the optimal value of the objective function, and the values of all the variables.

The first step is to separate the basic and nonbasic variables. Provided the optimal solution is not degenerate[3], this is straightforward. The basic variables are nonzero, while all the zero variables are nonbasic. Since this example is not degenerate, the nonbasic variables are

[3] We treat the degenerate case below.

```
zerovars = Cases[solution[[2]], Rule[x_,0] -> x]
```

```
{s[l], s[m], x[c]}
```

and the rest are basic variables.

```
basic = Complement[vars,zerovars]
```

```
{s[f], x[b], x[d]}
```

The constraints of the final tableau are obtained by solving the original constraints for the basic variables.

```
FromRules[rules_List] := Apply[Equal,rules,1]
```

```
rules = Flatten @ Solve[SFconstraints,basic];
FromRules[rules] // ExpandAll
```

```
s[f] == 10 + s[m] - x[c]

                s[l]   3 s[m]   x[c]
x[b] == 7 + ---- - ------ - ----
                 5       5       5

              2 s[l]   s[m]   3 x[c]
x[d] == 6 - ------ + ---- - ------
                5       5       5
```

where **FromRules** converts a list of transformation rules back into equations.

PROGRAMMING NOTE: It is necessary to use **Solve** rather than **Roots** to solve the constraints for the basic variables, since **Roots** only applies to a single equation. This provides a list of transformation rules, to which we must apply **FromRules**, which is the inverse of the built-in function **ToRules**.

The revised objective function is

```
profit /.rules // ExpandAll
```

```
      3 s[l]   6 s[m]   7 x[c]
39 - ------ - ------ - ------
        5        5        5
```

Thus the final tableau can be reconstructed

```
finalTableau = Prepend[FromRules[rules],
    z == profit /. rules] // ExpandAll
```

```
          3 s[l]   6 s[m]   7 x[c]
z == 39 - ------ - ------ - ------
            5        5        5

s[f] == 10 + s[m] - x[c]

            s[l]   3 s[m]   x[c]
x[b] == 7 + ---- - ------ - ----
             5       5       5

            2 s[l]   s[m]   3 x[c]
x[d] == 6 - ------ + ---- - ------
              5       5       5
```

We collect this procedure in the function **FinalTableau**, first removing the variable **finalTableau** to avoid confusion.

```
FinalTableau[objective_,constraints_] :=
    Module[{SFconstraints,vars,solution,zerovars,basic,rules},
    SFconstraints = StandardForm[constraints];
    vars = Union @@ Variables /@ SFconstraints;
    solution = ConstrainedMax[objective,SFconstraints,vars];
    zerovars = Cases[solution[[2]], Rule[x_,0] -> x];
    basic = Complement[vars,zerovars];
    rules = Flatten @
Solve[SFconstraints,basic];
    Prepend[FromRules[rules],
z == objective /. rules] // ExpandAll
]
```

```
FinalTableau[profit,SFconstraints]
```

```
          3 s[l]   6 s[m]   7 x[c]
z == 39 - ------ - ------ - ------
            5        5        5

s[f] == 10 + s[m] - x[c]

            s[l]   3 s[m]   x[c]
x[b] == 7 + ---- - ------ - ----
             5       5       5

            2 s[l]   s[m]   3 x[c]
x[d] == 6 - ------ + ---- - ------
              5       5       5
```

8.0.1 Degeneracy

Linear programming problems are degenerate when some of the basic variables are zero at the optimal solution. To handle degeneracy, we have to modify **FinalTableau** to select a suitable basis for inverting the initial tableau.[4] All nonzero variables are basic. To these we add a sufficient number of zero variables to complete the basis. However, not all of the nonzero variables can be basic, and some trial and error is required.

To illustrate degeneracy, consider the following example

```
objective = x[1];
```

```
constraints = { x[1] <= 1,
x[1] + x[2]      <= 1,
x[1] + x[2] + x[3] <= 1};
```

```
SFconstraints = StandardForm[constraints]
```

```
s[1] == 1 - x[1]

s[2] == 1 - x[1] - x[2]

s[3] == 1 - x[1] - x[2] - x[3]
```

```
vars = Union @@ Variables /@ SFconstraints
```

```
{s[1], s[2], s[3], x[1], x[2], x[3]}
```

The optimal solution is

```
solution=ConstrainedMax[objective, SFconstraints, vars]
```

```
{1, {s[1] -> 0, s[2] -> 0, s[3] -> 0, x[1] -> 1, x[2] -> 0,

  x[3] -> 0}}
```

The only nonzero variable is $x_1 = 1$. The remaining five variables are all zero.

```
zerovars = Cases[solution[[2]], Rule[x_,0] -> x]
```

```
{s[1], s[2], s[3], x[2], x[3]}
```

[4] A basis comprises m distinct variables, the coefficients of which are linearly independent (where m is the number of rows in the standard form).

```
nonzerovars = Complement[vars,zerovars]
```

```
{x[1]}
```

Any basis must include x_1 and two of the five zero variables. However, not all selections of zero variables form a basis. For example, the set x_1, s_1, s_2 forms a basis, since the constraints are solvable for these basis variables

```
Solve[SFconstraints, {x[1], s[1], s[2]}]
```

```
{{s[1] -> s[3] + x[2] + x[3], s[2] -> s[3] + x[3],

  x[1] -> 1 - s[3] - x[2] - x[3]}}
```

However, the sets x_1, x_2, s_1 and x_1, x_3, s_3 do not form a basis, and cannot be used to solve for the final tableau. For example

```
Solve[SFconstraints, {x[1], x[2], s[1]}]
```

```
{}
```

where the empty set $\{\}$ indicates no solution. To find a basis, we simply iterate through subsets of zero variables until we find a set which, together with the nonzero variables, forms a basis. To extract subsets of the appropriate size, we adopt the function **KSubsets** from the package DiscreteMath‘Combinatorica‘, which we repeat here for convenience.

```
KSubsets[l_List,0] := { {} }
KSubsets[l_List,1] := Partition[l,1]
KSubsets[l_List,k_Integer?Positive] := {l} /; (k == Length[l])
KSubsets[l_List,k_Integer?Positive] := {}  /; (k > Length[l])

KSubsets[l_List,k_Integer?Positive] :=
        Join[
                Map[(Prepend[#,First[l]])&, KSubsets[Rest[l],k-1]],
                KSubsets[Rest[l],k]
        ]
```

In the example, the subsets of size 2 of zero variables are

```
KSzerovars = KSubsets[zerovars,2]
```

```
{{s[1], s[2]}, {s[1], s[3]}, {s[1], x[2]}, {s[1], x[3]},

 {s[2], s[3]}, {s[2], x[2]}, {s[2], x[3]}, {s[3], x[2]},

 {s[3], x[3]}, {x[2], x[3]}}
```

FindBasic successively tries the elements of this set until it constructs a suitable basis.

```
FindBasic[SFconstraints_,nonzerovars_,KSzerovars_]:=
   Module[{rules},
    rules = Solve[SFconstraints,
Join[nonzerovars,First[KSzerovars]]];
    If[rules != {},rules,
FindBasic[SFconstraints,nonzerovars,Rest[KSzerovars]]]
]
```

```
FindBasic[SFconstraints,nonzerovars,KSzerovars]
```

```
{{s[1] -> s[3] + x[2] + x[3], s[2] -> s[3] + x[3],

  x[1] -> 1 - s[3] - x[2] - x[3]}}
```

In this case, the first selection s_1, s_2 are independent and sufficient to complete the basis. However, if we reorder the subsets

```
RotateLeft[KSzerovars,2] // Short
```

```
{{s[1], x[2]}, {s[1], x[3]}, <<7>>, {s[1], s[3]}}
```

```
FindBasic[SFconstraints,nonzerovars,%]
```

```
{{s[1] -> s[2] + x[2], x[3] -> s[2] - s[3], x[1] -> 1 - s[2] - x[2]}}
```

FindBasic deduces that the first pair s_1, x_2 is not basic, and chooses the second pair s_1, x_3. Since the function **KSubsets** orders the subsets lexicographically, **FindBasic** will select slack variables s_j before decision variables x_j. The former are more likely to be basic, and **FindBasic** is likely to find a basis quickly.

A minor modification equips **FinalTableau** to handle degeneracy.

```
FinalTableau[objective_,constraints_] :=
    Module[{SFconstraints,vars,solution,zerovars,nonzerovars,rules},
    SFconstraints = StandardForm[constraints];
    vars = Union @@ Variables /@ SFconstraints;
    solution = ConstrainedMax[objective,SFconstraints,vars];
    zerovars = Cases[solution[[2]], Rule[x_,0] -> x];
    nonzerovars = Complement[vars,zerovars];
    rules = Flatten @ FindBasic[SFconstraints,nonzerovars,
KSubsets[zerovars,Length[SFconstraints]-Length[nonzerovars]]];
    Prepend[FromRules[rules],
z == objective /. rules] // ExpandAll
]
```

```
FinalTableau[objective,constraints]
```

```
z == 1 - s[3] - x[2] - x[3]

s[1] == s[3] + x[2] + x[3]

s[2] == s[3] + x[3]

x[1] == 1 - s[3] - x[2] - x[3]
```

8.0.2 Nonexistence

To make **FinalTableau** robust, we need to equip it with the possibility that no solution exists, because the feasible set is either empty or unbounded. This will become evident when **ConstrainedMax** attempts to solve the problem, and *Mathematica* will generate an error message. The easy way to provide for this possibility uses the in built function **Check** to monitor **ConstrainedMax**, and to abort if an error message is detected.

```
FinalTableau[objective_,constraints_] :=
    Module[{SFconstraints,vars,solution,zerovars,nonzerovars,rules},
    SFconstraints = StandardForm[constraints];
    vars = Union @@ Variables /@ SFconstraints;
    solution =
Check[ConstrainedMax[objective,SFconstraints,vars],
      nonexistent];
(* stop if no solution exists *)
    If[solution == nonexistent, Return[]];
(* else *)
      zerovars = Cases[solution[[2]], Rule[x_,0] -> x];
      nonzerovars = Complement[vars,zerovars];
      rules = Flatten @ FindBasic[SFconstraints,nonzerovars,
KSubsets[zerovars,Length[SFconstraints]-Length[nonzerovars]]];
       Prepend[FromRules[rules],
z == objective /. rules] // ExpandAll
]
```

To illustrate, we saw earlier that the following problem is unbounded

```
FinalTableau[x[1] + x[2],{x[1] - x[2] <= 10}]
```

```
ConstrainedMax::nbdd: Specified domain appears unbounded.
```

whereas the next problem is infeasible

```
FinalTableau[3x[1] + x[2],
  { x[1] - x[2] <= 1,
   -x[1] - x[2] <= -33,
      2x[1] + x[2] <= 2}]
```

```
ConstrainedMax::nsat: The specified constraints cannot be satisfied.
```

For simplicity, we rely on the native Mathematica error messages. These could easily be converted to something more informative if required.

9 A classic example

One of the first applications of the simplex algorithm was to the determination of an adequate diet of minimum cost. The nutrition problem had been posed by George Stigler (1945), who gave a heuristic analysis. As a first step in his analysis of the problem, Stigler eliminated all foods whose nutritional contribution was dominated by other commodities. This reduced the list of eligible foods from 77 to 15. A further 6 foods were eliminated because they were dominated by a linear combination of the remaining foods. He then derived an approximate solution by inspection.[5] In 1947, J. Laderman of the National Bureau of Standards, utilized the problem to test the newly developed simplex algorithm. Using desk calculators, the solution of the system involving 9 equations and 77 unknowns required 120 person-days to calculate (Dantzig, 1963:551). In 1953, the problem was solved by Dantzig on an IBM 701 computer in about 4 minutes.

The reduced problem, comprising 9 foods and 9 constraints, poses a worthwhile test for our linear programming functions. The data for the problem, the nutritive value per dollar of expenditure of the 9 foods and the minimum daily allowance of nutrients, are listed in the following table.

	Calories (1000)	Protein (grams)	Calcium (grams)	Iron (mg)
Daily Allowance	3	70	0.8	12
Wheat Flour	44.7	1411	2.0	365
Evaporated Milk	8.4	422	15.1	9
Cheese (Cheddar)	7.4	448	16.4	19
Liver (Beef)	2.2	333	0.2	139
Cabbage	2.6	125	4.0	36
Spinach	1.1	106		138
Sweet Potatoes	9.6	138	2.7	54
Lima Beans (Dried)	17.4	1055	3.7	459
Navy Beans (Dried)	26.9	1691	11.4	792

	Vitamin A (1000 IU)	Thiamine (B_1) (mg)	Riboflavin (mg)	Niacin (mg)	Ascorbic Acid (mg)
Daily Allowance	5	1.8	2.7	18	75
Wheat Flour		55.4	33.3	441	
Evaporated Milk	26	3.0	23.5	11	60
Cheese (Cheddar)	28.1	0.8	10.3	4	
Liver (Beef)	169.2	6.4	50.8	316	525
Cabbage	7.2	9.0	4.5	26	5369
Spinach	918.4	5.7	13.8	33	2755
Sweet Potatoes	290.7	8.4	5.4	83	1912
Lima Beans (Dried)	5.1	26.9	38.2	93	
Navy Beans (Dried)		38.4	24.6	217	

[5] Stigler did not anticipate the development of the simplex algorithm two years later, stating "... there does not appear to be any direct method of finding the minimum of linear function subject to linear conditions." (p. 310) However, his approximate solution turned out to be only 27 cent per year more costly than the optimum!

Since the nutritive values of the various foods are given per dollar of expenditure, the cost function is total expenditure

```
Off[General::spell1];
cost =  x[Flour] + x[Milk] + x[Cheese] + x[Liver] +
x[Cabbage] + x[Spinach] + x[Potatoes] +
 x[Lima] + x[Navy];
```

where x_{Flour} is the expenditure on Flour, and so on. The nutritional constraints are

```
constraints =
{44.7x[Flour] + 8.4x[Milk] + 7.4x[Cheese] + 2.2x[Liver]
    + 2.6x[Cabbage] + 1.1x[Spinach] + 9.6x[Potatoes] + 17.4 x[Lima]
 + 26.9x[Navy] >= 3,
1411x[Flour] + 422x[Milk] + 448x[Cheese] + 333x[Liver]
    + 125x[Cabbage] + 106x[Spinach] + 138x[Potatoes] + 1055 x[Lima]
 + 1691x[Navy] >= 70,
 2x[Flour] + 15.1x[Milk] + 16.4x[Cheese] + 0.2x[Liver]
    + 4x[Cabbage]  + 2.7x[Potatoes] + 3.7 x[Lima]
 + 11.4x[Navy] >= .8,
 365x[Flour] + 9x[Milk] + 19x[Cheese] + 139x[Liver]
    + 36x[Cabbage] + 138x[Spinach] + 54x[Potatoes] + 459 x[Lima]
 + 792x[Navy] >= 12,
 26x[Milk] + 28.1x[Cheese] + 169.2x[Liver]
    + 7.2x[Cabbage] + 918.4x[Spinach] + 290.7x[Potatoes] + 5.1 x[Lima]
     >= 5,
55.4x[Flour] + 3x[Milk] + 0.8x[Cheese] + 6.4x[Liver]
    + 9x[Cabbage] + 5.7x[Spinach] + 8.4x[Potatoes] + 26.9 x[Lima]
 + 38.4x[Navy] >= 1.8,
33.3x[Flour] + 23.5x[Milk] + 10.3x[Cheese] + 50.8x[Liver]
    + 4.5x[Cabbage] + 13.8x[Spinach] + 5.4x[Potatoes] + 38.2 x[Lima]
 + 24.6x[Navy] >= 2.7,
441x[Flour] + 11x[Milk] + 4x[Cheese] + 316x[Liver]
    + 26x[Cabbage] + 33x[Spinach] + 83x[Potatoes] + 93 x[Lima]
 + 217x[Navy] >= 18,
60x[Milk]     + 525x[Liver]
    + 5369x[Cabbage] + 2755x[Spinach] + 1912x[Potatoes] >= 75};
```

Computing the final tableau we get

```
finalTableau=FinalTableau[-cost,StandardForm[constraints,
{Cals,Prot,Ca,Fe,A,B1,B2,Nia,C}]]

z == -0.108662 - 0.000400233 s[A] - 0.016358 s[B2] -

  0.000144118 s[C] - 0.0317377 s[Ca] - 0.00876515 s[Cals] -

  0.234905 x[Cheese] - 0.103139 x[Lima] - 0.0436664 x[Milk] -

  0.349929 x[Potatoes]
```

```
s[B1] == 2.32044 + 0.00178173 s[A] + 0.0600855 s[B2] +
  0.00070545 s[C] + 0.469462 s[Ca] + 1.17361 s[Cals] -
  16.2528 x[Cheese] + 2.43788 x[Lima] - 15.4478 x[Milk] -
  6.3254 x[Potatoes]
s[Fe] == 48.4669 + 0.231077 s[A] + 1.96333 s[B2] - 0.0384643 s[C] +
  55.254 s[Ca] + 4.23072 s[Cals] - 945.189 x[Cheese] +
  104.768 x[Lima] - 910.712 x[Milk] - 140.033 x[Potatoes]
s[Nia] == 9.31598 - 0.0721403 s[A] + 6.22203 s[B2] +
  0.00263721 s[C] - 7.52855 s[Ca] + 5.56742 s[Cals] +
  24.2095 x[Cheese] - 213.331 x[Lima] - 66.5855 x[Milk] +
  32.2098 x[Potatoes]
s[Prot] == 77.4135 + 0.166875 s[A] + 5.18424 s[B2] - 0.0527498 s[C] +
  80.247 s[Ca] + 24.1134 s[Cals] - 1104.58 x[Cheese] +
  139.623 x[Lima] - 1115.29 x[Milk] - 285.804 x[Potatoes]
x[Flour] == 0.0295191 - 0.000114638 s[A] - 0.000828639 s[B2] +
  0.0000321396 s[C] - 0.0586491 s[Ca] + 0.0256128 s[Cals] +
  0.784067 x[Cheese] - 0.196422 x[Lima] + 0.690979 x[Milk] -
  0.111181 x[Potatoes]
x[Navy] == 0.0610286 + 0.000221774 s[A] - 0.000205663 s[B2] -
  0.0000712074 s[C] + 0.0981656 s[Ca] - 0.00423899 s[Cals] -
  1.58266 x[Cheese] - 0.282729 x[Lima] - 1.44335 x[Milk] -
  0.151564 x[Potatoes]
x[Spinach] == 0.00500766 + 0.00114747 s[A] - 0.00394299 s[B2] -
             -7
  9.3211 10   s[C] + 0.00176352 s[Ca] + 0.00285849 s[Cals] -
  0.0417057 x[Cheese] + 0.0885074 x[Lima] + 0.0122415 x[Milk] -
```

```
  0.342698 x[Potatoes]

x[Cabbage] == 0.0112144 - 0.00056002 s[A] - 0.000069798 s[B2] +

  0.000187016 s[C] + 0.0000312175 s[Ca] + 0.0000506004 s[Cals] +

  0.0155691 x[Cheese] + 0.00452643 x[Lima] + 0.00408336 x[Milk] -

  0.19497 x[Potatoes]

x[Liver] == 0.00189256 - 0.000294355 s[A] + 0.0214051 s[B2] -

              -6
  2.89874 10    s[C] - 0.00957355 s[Ca] - 0.0155178 s[Cals] +

  0.0596362 x[Cheese] - 0.510744 x[Lima] - 0.220284 x[Milk] +

  0.150343 x[Potatoes]
```

On[General::spell1];

This is too much information to be readily absorbed. We can focus attention on the optimal solution values by setting the nonbasic variables to zero.

finalTableau /. Zero[NonBasicVariables @ finalTableau]

```
z == -0.108662

s[B1] == 2.32044

s[Fe] == 48.4669

s[Nia] == 9.31598

s[Prot] == 77.4135

x[Flour] == 0.0295191

x[Navy] == 0.0610286

x[Spinach] == 0.00500766

x[Cabbage] == 0.0112144

x[Liver] == 0.00189256
```

The minimum cost diet in not exciting fare, comprising just five foods: wheat flour, navy beans, spinnach, cabbage and liver. Nutritional requirements could be met at a cost (in 1939 prices) of 10.9 cents per day or $39.66 dollars per year. This is only 27 cents (per year) less than the diet computed heuristically by Stigler. Only five of the nine nutritional constraints are binding.

More useful information is provided by a sensitivity analysis.

```
SensitivityAnalysis[finalTableau,-cost]

{{x[Cabbage], -1, -0.714676 <= Delta[Cabbage] <= 0.770615},
 {x[Cheese], -1, -Infinity <= Delta[Cheese] <= 0.234905},
 {x[Flour], -1, -0.525087 <= Delta[Flour] <= 0.063195},
 {x[Lima], -1, -Infinity <= Delta[Lima] <= 0.103139},
 {x[Liver], -1, -0.198228 <= Delta[Liver] <= 0.764211},
 {x[Milk], -1, -Infinity <= Delta[Milk] <= 0.0436664},
 {x[Navy], -1, -0.0302534 <= Delta[Navy] <= 0.323308},
 {x[Potatoes], -1, -Infinity <= Delta[Potatoes] <= 0.349929},
 {x[Spinach], -1, -1.0211 <= Delta[Spinach] <= 0.348796},
 {s[B1], 0, -2.32044 <= Delta[B1] <= Infinity},
 {s[Fe], 0, -48.4669 <= Delta[Fe] <= Infinity},
 {s[Nia], 0, -9.31598 <= Delta[Nia] <= Infinity},
 {s[Prot], 0, -77.4135 <= Delta[Prot] <= Infinity},
 {s[A], 0.000400233, -6.42952 <= Delta[A] <= 4.36409},
 {s[B2], 0.016358, -1.27002 <= Delta[B2] <= 0.0884161},
 {s[C], 0.000144118, -652.89 <= Delta[C] <= 59.9651},
 {s[Ca], 0.0317377, -0.197686 <= Delta[Ca] <= 0.62169},
 {s[Cals], 0.00876515, -0.121961 <= Delta[Cals] <= 1.15251}}
```

For example, sensitivity analysis reveals that the price of cheese would have to fall nearly 25% before it would enter the optimal diet. A 3% rise in the price of navy beans would eliminate them from the optimal diet, while spinach would be retained even if the price rose 100%.

Since the nutritional constraints are measured in different units, it is worthwhile converting the shadow prices of the nutritional constraints to elasticities. Economists use elasticities to relate the percentage change in one variable to the percentage change in another, giving a measure of responsiveness which is independent of the units of measurement.

The nutritional requirements are

```
requirements = StandardForm[constraints,
    {Cals,Prot,Ca,Fe,A,B1,B2,Nia,C}] /.
lhs_ == rhs_ :> {lhs, Abs @ Intercept[rhs]}
```

```
{{s[Cals], 3}, {s[Prot], 70}, {s[Ca], 0.8}, {s[Fe], 12}, {s[A], 5},

 {s[B1], 1.8}, {s[B2], 2.7}, {s[Nia], 18}, {s[C], 75}}
```

and their shadow prices are

```
shadowP = ShadowPrices[finalTableau,#[[1]]& /@
requirements]
```

```
{{s[Cals], 0.00876515}, {s[Prot], 0}, {s[Ca], 0.0317377}, {s[Fe], 0},

 {s[A], 0.000400233}, {s[B1], 0}, {s[B2], 0.016358}, {s[Nia], 0},

 {s[C], 0.000144118}}
```

The cost of the optimal diet is

```
minimumCost = Intercept[First[finalTableau][[2]]] //
 Abs
```

```
0.108662
```

The *elasticity* of the cost of the optimal diet with respect to changes in the *calorie* requirement is

$$\begin{aligned}\text{Elasticity} &= 100\frac{\text{Shadow price/Minimum cost}}{\text{1/Nutritional requirement}}\\ &= 100\frac{0.00876515/0.108662}{1/3}\\ &= 24.1993\%\end{aligned}$$

This calculation is made by the function **Elasticity**.

```
Elasticity[{var_,requirement_},{var_,shadowPrice_}] :=
{var,100(shadowPrice/minimumCost)requirement}
```

The elasticities of cost with respect to changes in the nutritional requirements can be computed by threading this function over the lists of requirements and shadow prices.

```
Thread[Elasticity[requirements,shadowP]] //
TableForm
```

```
s[Cals]    24.1992

s[Prot]    0

s[Ca]      23.3661
```

```
s[Fe]      0

s[A]       1.84164

s[B1]      0

s[B2]      40.6458

s[Nia]     0

s[C]       9.94716
```

The elasticities are zero for the nonbinding constraints. The cost of the optimal diet is very sensitive to the vitamin B2 requirement. A one percent increase in this requirement would increase the cost of the minimum diet by 40 %. At the other end of the scale, the cost of the optimal diet is relative insensitive to the vitamin A requirement, and quite insensitive to the vitamin B1 requirement (which is not binding).

Finally, some interesting information was obtained by timing the computation of the final tableau. On my 50 MHz DX, calculating the final tableau took 51 seconds. Of this, only 0.4 seconds was required to solve the problem using **ConstrainedMax**. Solving the system in terms of the basic variables took considerably longer, 2.9 seconds. Most of the time (48 seconds) was required simply to transform the systems of equations in the final tableau using **ExpandAll**! Since **ExpandAll** is used repeatedly in Simplex, we would expect algorithm implemented here (**LP**) to very slow. I am not sure how slow, because the following calculation was aborted after 46 hours.

```
LP[Prepend[StandardForm[requirements,
{Cals,Prot,Ca,Fe,A,B1,B2,Nia,C}],
- cost]] // Timing
```

```
$Aborted
```

10 Conclusion

This notebook describes a package designed to supplement *Mathematica's* in built linear programming facilities. We exploit the symbolic capabilities of *Mathematica* to provide a self-contained exposition of the simplex algorithm and a set of tools for conducting sensitivity analysis of the optimal solution. Since *Mathematica's* native linear optimization function **ConstrainedMax** is faster than the implementation provided here, we then develop a function which can deduce the final tableau from the Spartan output of **ConstrainedMax**. This final tableau contains all the information usually provided by a good linear programming package, This information can be explored using the tools for sensitivity analysis developed in the package. This enables the package to be applied to a substantive problem, relying on *Mathematica*'s native code for intensive computation.

11 References

Carter M. (1993) Cooperative games. In: Varian H. (ed) *Economic and Financial Modeling with Mathematica.* Springer-Verlag, New York

Chvatal V. (1983). *Linear Programming.* Freeman, San Francisco

Dantzig G. (1963). *Linear Programming and Extensions.* Princeton University Press, Princeton

Dorfman R., Samuelson P. A. and Solow R. M. (1958). *Linear Programming and Economic Analysis.* McGraw-Hill, New York

Franklin J. (1980). *Methods of Mathematical Economics - Linear and Nonlinear Programming, Fixed Point Theorems.* Springer-Verlag

Lemke C. E. and Howson J. T. (1964). "Equilibrium points of bimatrix games." *SIAM Journal of Applied Mathematics*, vol. 12, pp. 413-23.

Press W. H., Teukolsky S. A., Vetterling W. T., et al (1992). *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edn. Cambridge University Press, Cambridge

Stigler G. (1945). "The cost of subsistence" *Journal of Farm Economics*, pp. 303-314

Strum J. E. (1972). *Introduction to linear programming.* Holden-Day, San Francisco

Wilson R. (1992). "Computing simple stable equilibria." *Econometrica*, vol. 60, pp. 1039-70.

LIST OF DISCUSSION PAPERS*

No. 9101 Bounds on the Effect of Heteroscedasticity on the Chow Test for Structural Change, by David Giles and Offer Lieberman.
No. 9102 The Optimal Size of a Preliminary Test for Linear Restrictions when Estimating the Regression Scale Parameter, by Judith A. Giles and Offer Lieberman.
No. 9103 Some Properties of the Durbin-Watson Test After a Preliminary t-Test, by David Giles and Offer Lieberman.
No. 9104 Preliminary-Test Estimation of the Regression Scale Parameter when the Loss Function is Asymmetric, by Judith A. Giles and David E. A. Giles.
No. 9105 On an Index of Poverty, by Manimay Sengupta and Prasanta K. Pattanaik.
No. 9106 Cartels May Be Good For You, by Michael Carter and Julian Wright.
No. 9107 Lp-Norm Consistencies of Nonparametric Estimates of Regression, Heteroskedasticity and Variance of Regression Estimate when Distribution of Regression is Known, by Radhey S. Singh.
No. 9108 Optimal Telecommunications Tariffs and the CCITT, by Michael Carter and Julian Wright.
No. 9109 Price Indices : Systems Estimation and Tests, by David Giles and Ewen McCann.
No. 9110 The Limiting Power of Point Optimal Autocorrelation Tests, by John P. Small.
No. 9111 The Exact Power of Some Autocorrelation Tests When the Disturbances are Heteroscedastic, by John P. Small.
No. 9112 Some Consequences of Using the Chow Test in the Context of Autocorrelated Disturbances, by David Giles and Murray Scott.
No. 9113 The Exact Distribution of R when the Disturbances are Autocorrelated, by Mark L. Carrodus and David E. A. Giles.
No. 9114 Optimal Critical Values of a Preliminary Test for Linear Restrictions in a Regression Model with Multivariate Student-t Disturbances, by Jason K. Wong and Judith A. Giles.
No. 9115 Pre-Test Estimation in a Regression Model with a Misspecified Error Covariance Matrix, by K. V. Albertson.
No. 9116 Estimation of the Scale Parameter After a Pre-test for Homogeneity in a Mis-specified Regression Model, by Judith A. Giles.
No. 9201 Testing for Arch-Garch Errors in a Mis-specified Regression, by David E. A. Giles, Judith A. Giles, and Jason K. Wong.
No. 9202 Quasi Rational Consumer Demand Some Positive and Normative Surprises, by John Fountain.
No. 9203 Pre-test Estimation and Testing in Econometrics: Recent Developments, by Judith A. Giles and David E. A. Giles.
No. 9204 Optimal Immigration in a Model of Education and Growth, by K-L. Shea and A. E. Woodfield.
No. 9205 Optimal Capital Requirements for Admission of Business Immigrants in the Long Run, by K-L. Shea and A. E. Woodfield.
No. 9206 Causality, Unit Roots and Export-Led Growth: The New Zealand Experience, by David E. A. Giles, Judith A. Giles and Ewen McCann.
No. 9207 The Sampling Performance of Inequality Restricted and Pre-Test Estimators in a Mis-specified Linear Model, by Alan T. K. Wan.
No. 9208 Testing and Estimation with Seasonal Autoregressive Mis-specification, by John P. Small.
No. 9209 A Bargaining Experiment, by Michael Carter and Mark Sunderland.
No. 9210 Pre-Test Estimation in Regression Under Absolute Error Loss, by David E. A. Giles.
No. 9211 Estimation of the Regression Scale After a Pre-Test for Homoscedasticity Under Linex Loss, by Judith A. Giles and David E. A. Giles.
No. 9301 Assessing Starmer's Evidence for New Theories of Choice: A Subjectivist's Comment, by John Fountain.
No. 9302 Preliminary-Test Estimation in a Dynamnic Linear Model, by David E. A. Giles and Matthew C. Cunneen.

(Continued on next page)

No. 9303 Fans, Frames and Risk Aversion: How Robust is the Common Consequence Effect? by John Fountain and Michael McCosker.

No. 9304 Pre-test Estimation of the Regression Scale Parameter with Multivariate Student-t Errors and Independent Sub-Samples, by Juston Z. Anderson and Judith A. Giles

No. 9305 The Exact Powers of Some Autocorrelation Tests When Relevant Regressors are Omitted, by J. P. Small, D. E. Giles and K. J. White.

No. 9306 The Exact Risks of Some Pre-Test and Stein-Type Regression Estimators Under Balanced Loss*, by J. A. Giles, D. E. A. Giles, and K. Ohtani.

No. 9307 The Risk Behavior of a Pre-Test Estimator in a Linear Regression Model with Possible Heteroscedasticity under the Linex Loss Function, by K. Ohtani, D. E. A. Giles and J. A. Giles.

No. 9308 Comparing Standard and Robust Serial Correlation Tests in the Presence of Garch Errors, by John P. Small.

No. 9309 Testing for Serial Independence in Error Components Models: Finite Sample Results, by John P. Small.

No. 9310 Optimal Balanced-Growth Immigration Policy for Investors and Entrepeneurs, by A. E. Woodfield and K-L. Shea.

No. 9311 Optimal Long-Run Business Immigration Under Differential Savings Functions, by A. E. Woodfield and K-L. Shea.

No. 9312 The Welfare Cost of Taxation in New Zealand Following Major Tax Reforms, by P. McKeown and A. Woodfield.

No. 9313 The Power of the Goldfeld-Quandt Test when the errors are autocorrelated, by J.P. Small and R.J. Dennis.

No. 9314 The Nucleolus Strikes Back, by M. Carter and P. Walker.

No. 9315 The Output-Inflation Tradeoff in the United States: New evidence on the New Classical vs. New Keynesian Debate, by Alfred V. Guender

No. 9401 Insurance Market Equilibrium and the Welfare Costs of Gender-Neutral Insurance Pricing under Alternative Regulatory Regimes by Alan E. Woodfield

No. 9402 Labour Market Signalling and the Welfare Costs of Regulated Insurance Market Equilibria under Gender-neutral Pricing, by Alan E. Woodfield.

No. 9403 The New Classical Vs The New Keynesian debate On The Output – Inflation tradeoff: Evidence From Four industrialized Countries, by Alfred V. Guender

No. 9404 Yield Spreads & Real Economic Activity: The Case of New Zealand & Australia, by Alfred V. Guender and Mathias Moersch.

No. 9405 Periodic Integration & cointegration with applications to the New Zealand Aggregate Consumption Function, by Robin Harrison and Aaron Smith.

9406 Linear Programming with Mathematica, by Michael Carter

* Copies of these Discussion Papers may be obtained for $4 (including postage, price changes occasionally) each by writing to the Secretary, Department of Economics, University of Canterbury, Christchurch, New Zealand. A list of the Discussion Papers prior to 1989 is available on request.