2

Diet and Other Input Models: Minimizing Costs

To complement the profit-maximizing models of Chapter 1, we now consider linear programming models in which the objective is to minimize costs. Where the constraints of maximization models tend to be upper limits on the availability of resources, the constraints in minimization models are more likely to be lower limits on the amounts of certain ‘‘qualities’’ in the solution.

As an intuitive example of a cost-minimizing model, this chapter uses the well-known ‘‘diet problem’’, which finds a mix of foods that satisfies requirements on the amounts of various vitamins. We will again construct a small, explicit linear program, and then show how a general model can be formulated for all linear programs of that kind. Since you are now more familiar with AMPL, however, we will spend more time on AMPL and less with algebraic notation.

After formulating the diet model, we will discuss a few changes that might make it more realistic. The full power of this model, however, derives from its applicability to many situations that have nothing to do with diets. Thus we conclude this chapter by rewriting the model in a more general way, and discussing its application to blending, economics, and scheduling.

2.1 A linear program for the diet problem

Consider the problem of choosing prepared foods to meet certain nutritional requirements. Suppose that precooked dinners of the following kinds are available for the following prices per package:

BEEF	beef	$3.19
CHK	chicken	2.59
FISH	fish	2.29
HAM	ham	2.89
MCH	macaroni & cheese	1.89
MTL	meat loaf	1.99
SPG	spaghetti	1.99
TUR	turkey	2.49

These dinners provide the following percentages, per package, of the minimum daily requirements for vitamins A, C, B1 and B2:

	A	C	B1	B2
BEEF	60%	20%	10%	15%
CHK	8	0	20	20
FISH	8	10	15	10
HAM	40	40	35	10
MCH	15	35	15	15
MTL	70	30	15	15
SPG	25	50	25	15
TUR	60	20	15	10

The problem is to find the cheapest combination of packages that will meet a week's requirements — that is, at least 700% of the daily requirement for each nutrient.

Let us write X_{BEEF} for the number of packages of beef dinner to be purchased, X_{CHK} for the number of packages of chicken dinner, and so forth. Then the total cost of the diet will be:

$$\text{total cost} = 3.19\,X_{BEEF} + 2.59\,X_{CHK} + 2.29\,X_{FISH} + 2.89\,X_{HAM} + 1.89\,X_{MCH} + 1.99\,X_{MTL} + 1.99\,X_{SPG} + 2.49\,X_{TUR}$$

The total percentage of the vitamin A requirement is given by a similar formula, except that X_{BEEF}, X_{CHK}, and so forth are multiplied by the percentage per package instead of the cost per package:

$$\text{total percentage of vitamin A daily requirement met} = 60\,X_{BEEF} + 8\,X_{CHK} + 8\,X_{FISH} + 40\,X_{HAM} + 15\,X_{MCH} + 70\,X_{MTL} + 25\,X_{SPG} + 60\,X_{TUR}$$

This amount needs to be greater than or equal to 700 percent. There is a similar formula for each of the other vitamins, and each of these also needs to be ≥ 700.

Putting these all together, we have the following linear program:

Minimize

$$3.19\,X_{BEEF} + 2.59\,X_{CHK} + 2.29\,X_{FISH} + 2.89\,X_{HAM} + 1.89\,X_{MCH} + 1.99\,X_{MTL} + 1.99\,X_{SPG} + 2.49\,X_{TUR}$$

Subject to

$$60\,X_{BEEF} + 8\,X_{CHK} + 8\,X_{FISH} + 40\,X_{HAM} + 15\,X_{MCH} + 70\,X_{MTL} + 25\,X_{SPG} + 60\,X_{TUR} \geq 700$$

$$20\,X_{BEEF} + 0\,X_{CHK} + 10\,X_{FISH} + 40\,X_{HAM} + 35\,X_{MCH} + 30\,X_{MTL} + 50\,X_{SPG} + 20\,X_{TUR} \geq 700$$

$$10\,X_{BEEF} + 20\,X_{CHK} + 15\,X_{FISH} + 35\,X_{HAM} + 15\,X_{MCH} + 15\,X_{MTL} + 25\,X_{SPG} + 15\,X_{TUR} \geq 700$$

$$15\,X_{BEEF} + 20\,X_{CHK} + 10\,X_{FISH} + 10\,X_{HAM} + 15\,X_{MCH} + 15\,X_{MTL} + 15\,X_{SPG} + 10\,X_{TUR} \geq 700$$

$$X_{BEEF} \geq 0, X_{CHK} \geq 0, X_{FISH} \geq 0, X_{HAM} \geq 0, X_{MCH} \geq 0, X_{MTL} \geq 0, X_{SPG} \geq 0, X_{TUR} \geq 0$$

At the end we have added the common-sense requirement that no fewer than zero packages of a food can be purchased.

As we first did with the production LP of Chapter 1, we can transcribe to a file, say `diet0.mod`, an AMPL statement of the explicit diet LP:

```
var Xbeef >= 0; var Xchk >= 0; var Xfish >= 0;
var Xham >= 0;  var Xmch >= 0; var Xmtl >= 0;
var Xspg >= 0;  var Xtur >= 0;

minimize cost:
  3.19*Xbeef + 2.59*Xchk + 2.29*Xfish + 2.89*Xham +
  1.89*Xmch  + 1.99*Xmtl + 1.99*Xspg  + 2.49*Xtur;

subject to A:
  60*Xbeef +  8*Xchk +  8*Xfish + 40*Xham +
  15*Xmch  + 70*Xmtl + 25*Xspg  + 60*Xtur >= 700;

subject to C:
  20*Xbeef +  0*Xchk + 10*Xfish + 40*Xham +
  35*Xmch  + 30*Xmtl + 50*Xspg  + 20*Xtur >= 700;

subject to B1:
  10*Xbeef + 20*Xchk + 15*Xfish + 35*Xham +
  15*Xmch  + 15*Xmtl + 25*Xspg  + 15*Xtur >= 700;

subject to B2:
  15*Xbeef + 20*Xchk + 10*Xfish + 10*Xham +
  15*Xmch  + 15*Xmtl + 15*Xspg  + 10*Xtur >= 700;
```

Again a few AMPL commands then suffice to read the file, send the LP to a solver, and retrieve the results:

```
ampl: model diet0.mod;
ampl: solve;
MINOS 5.5: optimal solution found.
6 iterations, objective 88.2
ampl: display Xbeef,Xchk,Xfish,Xham,Xmch,Xmtl,Xspg,Xtur;
Xbeef = 0
Xchk = 0
Xfish = 0
Xham = 0
Xmch = 46.6667
Xmtl = -3.69159e-18
Xspg = -4.05347e-16
Xtur = 0
```

The optimal solution is found quickly, but it is hardly what we might have hoped for. The cost is minimized by a monotonous diet of 46⅔ packages of macaroni and cheese! You can check that this neatly provides 15% × 46⅔ = 700% of the requirement for vitamins A, B1 and B2, and a lot more vitamin C than necessary; the cost is only \$1.89 × 46⅔ = \$88.20. (The tiny negative values for meat loaf and spaghetti can be regarded as zeros, like the tiny positive values we saw in Section 1.6.)

You might guess that a better solution would be generated by requiring the amount of each vitamin to equal 700% exactly. Such a requirement can easily be imposed by changing each >= to = in the AMPL constraints. If you go ahead and solve the changed LP, you will find that the diet does indeed become more varied: approximately 19.5 packages of chicken, 16.3 of macaroni and cheese, and 4.3 of meat loaf. But since equalities are more restrictive than inequalities, the cost goes up to \$89.99.

2.2 An AMPL model for the diet problem

Clearly we will have to consider more extensive modifications to our linear program in order to produce a diet that is even remotely acceptable. We will probably want to change the sets of food and nutrients, as well as the nature of the constraints and bounds. As in the production example of the previous chapter, this will be much easier to do if we rely on a general model that can be coupled with a variety of specific data files.

This model deals with two things: nutrients and foods. Thus we begin an AMPL model by declaring sets of each:

```
set NUTR;
set FOOD;
```

Next we need to specify the numbers required by the model. Certainly a positive cost should be given for each food:

```
param cost {FOOD} > 0;
```

We also specify that for each food there are lower and upper limits on the number of packages in the diet:

```
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];
```

Notice that we need a dummy index `j` to run over `FOOD` in the declaration of `f_max`, in order to say that the maximum for each food must be greater than or equal to the corresponding minimum.

To make this model somewhat more general than our examples so far, we also specify similar lower and upper limits on the amount of each nutrient in the diet:

```
param n_min {NUTR} >= 0;
param n_max {i in NUTR} >= n_min[i];
```

Finally, for each combination of a nutrient and a food, we need a number that represents the amount of the nutrient in one package of the food. You may recall from Chapter 1 that such a ''product'' of two sets is written by listing them both:

```
param amt {NUTR,FOOD} >= 0;
```

References to this parameter require two indices. For example, `amt[i,j]` is the amount of nutrient `i` in a package of food `j`.

The decision variables for this model are the numbers of packages to buy of the different foods:

```
var Buy {j in FOOD} >= f_min[j], <= f_max[j];
```

The number of packages of some food `j` to be bought will be called `Buy[j]`; in any acceptable solution it will have to lie between `f_min[j]` and `f_max[j]`.

The total cost of buying a food `j` is the cost per package, `cost[j]`, times the number of packages, `Buy[j]`. The objective to be minimized is the sum of this product over all foods `j`:

```
minimize Total_Cost:  sum {j in FOOD} cost[j] * Buy[j];
```

This `minimize` declaration works the same as `maximize` did in Chapter 1.

Similarly, the amount of a nutrient `i` supplied by a food `j` is the nutrient per package, `amt[i,j]`, times the number of packages `Buy[j]`. The total amount of nutrient `i` supplied is the sum of this product over all foods `j`:

```
sum {j in FOOD} amt[i,j] * Buy[j]
```

To complete the model, we need only specify that each such sum must lie between the appropriate bounds. Our constraint declaration begins

```
subject to Diet {i in NUTR}:
```

to say that a constraint named `Diet[i]` must be imposed for each member `i` of `NUTR`. The rest of the declaration gives the algebraic statement of the constraint for nutrient `i`: the variables must satisfy

```
n_min[i] <= sum {j in FOOD} amt[i,j] * Buy[j] <= n_max[i]
```

```
set NUTR;
set FOOD;

param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];

param n_min {NUTR} >= 0;
param n_max {i in NUTR} >= n_min[i];

param amt {NUTR,FOOD} >= 0;

var Buy {j in FOOD} >= f_min[j], <= f_max[j];

minimize Total_Cost:  sum {j in FOOD} cost[j] * Buy[j];

subject to Diet {i in NUTR}:
   n_min[i] <= sum {j in FOOD} amt[i,j] * Buy[j] <= n_max[i];
```

Figure 2-1: Diet model in AMPL (`diet.mod`).

A ‘‘double inequality’’ like this is interpreted in the obvious way: the value of the sum in the middle must lie between `n_min[i]` and `n_max[i]`. The complete model is shown in Figure 2-1.

2.3 Using the AMPL diet model

By specifying appropriate data, we can solve any of the linear programs that correspond to the above model. Let’s begin by using the data from the beginning of this chapter, which is shown in AMPL format in Figure 2-2.

The values of `f_min` and `n_min` are as given originally, while `f_max` and `n_max` are set, for the time being, to large values that won’t affect the optimal solution. In the table for `amt`, the notation `(tr)` indicates that we have ‘‘transposed’’ the table so the columns correspond to the first index (nutrients), and the rows to the second (foods). Alternatively, we could have changed the model to say

```
param amt {FOOD,NUTR}
```

in which case we would have had to write `amt[j,i]` in the constraint.

Suppose that model and data are stored in the files `diet.mod` and `diet.dat`, respectively. Then AMPL is used as follows to read these files and to solve the resulting linear program:

```
ampl: model diet.mod;
ampl: data diet.dat;

ampl: solve;
MINOS 5.5: optimal solution found.
6 iterations, objective 88.2
```

```
set NUTR := A B1 B2 C ;
set FOOD := BEEF CHK FISH HAM MCH MTL SPG TUR ;

param:   cost  f_min  f_max :=
  BEEF   3.19    0     100
  CHK    2.59    0     100
  FISH   2.29    0     100
  HAM    2.89    0     100
  MCH    1.89    0     100
  MTL    1.99    0     100
  SPG    1.99    0     100
  TUR    2.49    0     100 ;

param:   n_min  n_max :=
   A      700   10000
   C      700   10000
   B1     700   10000
   B2     700   10000 ;

param amt (tr):
           A    C   B1   B2 :=
   BEEF   60   20   10   15
   CHK     8    0   20   20
   FISH    8   10   15   10
   HAM    40   40   35   10
   MCH    15   35   15   15
   MTL    70   30   15   15
   SPG    25   50   25   15
   TUR    60   20   15   10 ;
```

Figure 2-2: Data for diet model (`diet.dat`).

```
ampl: display Buy;
Buy [*] :=
BEEF   0
 CHK   0
FISH   0
 HAM   0
 MCH  46.6667
 MTL  -1.07823e-16
 SPG  -1.32893e-16
 TUR   0
;
```

Naturally, the result is the same as before.

Now suppose that we want to make the following enhancements. To promote variety, the weekly diet must contain between 2 and 10 packages of each food. The amount of sodium and calories in each package is also given; total sodium must not exceed 40,000 mg, and total calories must be between 16,000 and 24,000. All of these changes can be made through a few modifications to the data, as shown in Figure 2-3. Putting this new data in file `diet2.dat`, we can run AMPL again:

```
set NUTR := A B1 B2 C NA CAL ;
set FOOD := BEEF CHK FISH HAM MCH MTL SPG TUR ;

param:   cost  f_min  f_max :=
  BEEF   3.19    2     10
  CHK    2.59    2     10
  FISH   2.29    2     10
  HAM    2.89    2     10
  MCH    1.89    2     10
  MTL    1.99    2     10
  SPG    1.99    2     10
  TUR    2.49    2     10  ;

param:   n_min  n_max :=
   A      700   20000
   C      700   20000
   B1     700   20000
   B2     700   20000
   NA       0   40000
   CAL  16000   24000 ;

param amt (tr):
           A    C   B1   B2    NA   CAL :=
   BEEF   60   20   10   15   938   295
   CHK     8    0   20   20  2180   770
   FISH    8   10   15   10   945   440
   HAM    40   40   35   10   278   430
   MCH    15   35   15   15  1182   315
   MTL    70   30   15   15   896   400
   SPG    25   50   25   15  1329   370
   TUR    60   20   15   10  1397   450 ;
```

Figure 2-3: Data for enhanced diet model (`diet2.dat`).

```
ampl: model diet.mod;
ampl: data diet2.dat;

ampl: solve;
MINOS 5.5: infeasible problem.
9 iterations
```

The message `infeasible problem` tells us that we have constrained the diet too tightly; there is no way that all of the restrictions can be satisfied.

AMPL lets us examine a variety of values produced by a solver as it attempts to find a solution. In Chapter 1, we used marginal (or dual) values to investigate the sensitivity of an optimum solution to changes in the constraints. Here there is no optimum, but the solver does return the last solution that it found while attempting to satisfy the constraints. We can look for the source of the infeasibility by displaying some values associated with this solution:

```
ampl: display Diet.lb, Diet.body, Diet.ub;
:   Diet.lb   Diet.body Diet.ub    :=
A       700    1993.09    20000
B1      700     841.091   20000
B2      700     601.091   20000
C       700    1272.55    20000
CAL   16000   17222.9     24000
NA        0   40000       40000
;
```

For each nutrient, `Diet.body` is the sum of the terms `amt[i,j] * Buy[j]` in the constraint `Diet[i]`. The `Diet.lb` and `Diet.ub` values are the ''lower bounds'' and ''upper bounds'' on the sum in `Diet[i]` — in this case, just the values `n_min[i]` and `n_max[i]`. We can see that the diet returned by the solver does not supply enough vitamin B2, while the amount of sodium (`NA`) has reached its upper bound.

At this point, there are two obvious choices: we could require less B2 or we could allow more sodium. If we try the latter, and relax the sodium limit to 50,000 mg, a feasible solution becomes possible:

```
ampl: let n_max["NA"] := 50000;

ampl: solve;
MINOS 5.5: optimal solution found.
5 iterations, objective 118.0594032

ampl: display Buy;
Buy [*] :=
BEEF   5.36061
 CHK   2
FISH   2
 HAM  10
 MCH  10
 MTL  10
 SPG   9.30605
 TUR   2
;
```

This is at least a start toward a palatable diet, although we have to spend $118.06, compared to $88.20 for the original, less restricted case. Clearly it would be easy, now that the model is set up, to try many other possibilities. (Section 11.3 describes ways to quickly change the data and re-solve.)

One still disappointing aspect of the solution is the need to buy 5.36061 packages of beef, and 9.30605 of spaghetti. How can we find the best possible solution in terms of whole packages? You might think that we could simply round the optimal values to whole numbers — or integers, as they're often called in the context of optimization — but it is not so easy to do so in a feasible way. Using AMPL to modify the reported solution, we can observe that rounding up to 6 packages of beef and 10 of spaghetti, for example, will violate the sodium limit:

```
ampl: let Buy["BEEF"] := 6;
ampl: let Buy["SPG"] := 10;

ampl: display Diet.lb, Diet.body, Diet.ub;
:   Diet.lb Diet.body Diet.ub    :=
A       700     2012    20000
B1      700     1060    20000
B2      700      720    20000
C       700     1730    20000
CAL   16000    20240    24000
NA        0    51522    50000
;
```

(The `let` statement, which permits modifications of data, is described in Section 11.3.) You can similarly check that rounding the solution down to 5 of beef and 9 of spaghetti will provide insufficient vitamin B2. Rounding one up and the other down doesn't work either. With enough experimenting you can find a nearby all-integer solution that does satisfy the constraints, but still you will have no guarantee that it is the least-cost all-integer solution.

AMPL does provide for putting the integrality restriction directly into the declaration of the variables:

```
var Buy {j in FOOD} integer >= f_min[j], <= f_max[j];
```

This will only help, however, if you use a solver that can deal with problems whose variables must be integers. For this, we turn to CPLEX, a solver that can handle these so-called integer programs. If we add `integer` to the declaration of variable `Buy` as above, save the resulting model in the file `dieti.mod`, and add the higher sodium limit to `diet2a.dat`, then we can re-solve as follows:

```
ampl: reset;
ampl: model dieti.mod;
ampl: data diet2a.dat;
ampl: option solver cplex;
ampl: solve;
CPLEX 8.0.0: optimal integer solution; objective 119.3
11 MIP simplex iterations
1 branch-and-bound nodes

ampl: display Buy;
Buy [*] :=
BEEF   9
 CHK   2
FISH   2
 HAM   8
 MCH  10
 MTL  10
 SPG   7
 TUR   2
;
```

Since integrality is an added constraint, it is no surprise that the best integer solution costs about $1.24 more than the best ‘‘continuous’’ one. But the difference between the diets is unexpected; the amounts of 3 foods change, each by two or more packages. In general, integrality and other ‘‘discrete’’ restrictions make solutions for a model much harder to find. We discuss this at length in Chapter 20.

2.4 Generalizations to blending, economics and scheduling

Your personal experience probably suggests that diet models are not widely used by people to choose their dinners. These models would be much better suited to situations in which packaging and personal preferences don’t play such a prominent role — for example, the blending of animal feed or perhaps food for college dining halls.

The diet model is a convenient, intuitive example of a linear programming formulation that appears in many contexts. Suppose that we rewrite the model in a more general way, as shown in Figure 2-4. The objects that were called foods and nutrients in the diet model are now referred to more generically as ‘‘inputs’’ and ‘‘outputs’’. For each input `j`, we must decide to use a quantity `X[j]` that lies between `in_min[j]` and `in_max[j]`; as a result we incur a cost equal to `cost[j] * X[j]`, and we create `io[i,j] * X[j]` units of each output `i`. Our goal is to find the least-cost combination of inputs that yields, for each output `i`, an amount between `out_min[i]` and `out_max[i]`.

In one common class of applications for this model, the inputs are raw materials to be mixed together. The outputs are qualities of the resulting blend. The raw materials could be the components of an animal feed, but they could equally well be the crude oil derivatives that are blended to make gasoline, or the different kinds of coal that are mixed as input to a coke oven. The qualities can be amounts of something (sodium or calories for animal feed), or more complex measures (vapor pressure or octane rating for gasoline), or even physical properties such as weight and volume.

In another well-known application, the inputs are production activities of some sector of an economy, and the outputs are various products. The `in_min` and `in_max` parameters are limits on the levels of the activities, while `out_min` and `out_max` are regulated by demands. Thus the goal is to find levels of the activities that meet demand at the lowest cost. This interpretation is related to the concept of an economic equilibrium, as we will explain in Chapter 19.

In still another, quite different application, the inputs are work schedules, and the outputs correspond to hours worked on certain days of a month. For a particular work schedule `j`, `io[i,j]` is the number of hours that a person following schedule `j` will work on day `i` (zero if none), `cost[j]` is the monthly salary for a person following schedule `j`, and `X[j]` is the number of workers assigned that schedule. Under this interpretation, the objective becomes the total cost of the monthly payroll, while the constraints say that for each day `i`, the total number of workers assigned to work that day must lie between the limits `out_min[i]` and `out_max[i]`. The same approach can

```
set INPUT;    # inputs
set OUTPUT;   # outputs

param cost {INPUT} > 0;
param in_min {INPUT} >= 0;
param in_max {j in INPUT} >= in_min[j];

param out_min {OUTPUT} >= 0;
param out_max {i in OUTPUT} >= out_min[i];

param io {OUTPUT,INPUT} >= 0;

var X {j in INPUT} >= in_min[j], <= in_max[j];

minimize Total_Cost:  sum {j in INPUT} cost[j] * X[j];

subject to Outputs {i in OUTPUT}:
   out_min[i] <= sum {j in INPUT} io[i,j] * X[j] <= out_max[i];
```

Figure 2-4: Least-cost input model (`blend.mod`).

be used in a variety of other scheduling contexts, where the hours, days or months are replaced by other periods of time.

Although linear programming can be very useful in applications like these, we need to keep in mind the assumptions that underlie the LP model. We have already mentioned the ''continuity'' assumption whereby `X[j]` is allowed to take on any value between `in_min[j]` and `in_max[j]`. This may be a lot more reasonable for blending than for scheduling.

As another example, in writing the objective as

```
sum {j in INPUT} cost[j] * X[j]
```

we are assuming ''linearity of costs'', that is, that the cost of an input is proportional to the amount of the input used, and that the total cost is the sum of the inputs' individual costs.

In writing the constraints as

```
out_min[i] <= sum {j in INPUT} io[i,j] * X[j] <= out_max[i]
```

we are also assuming that the yield of an output `i` from a particular input is proportional to the amount of the input used, and that the total yield of an output `i` is the sum of the yields from the individual inputs. This ''linearity of yield'' assumption poses no problem when the inputs are schedules, and the outputs are hours worked. But in the blending example, linearity is a physical assumption about the nature of the raw materials and the qualities, which may or may not hold. In early applications to refineries, for example, it was recognized that the addition of lead as an input had a nonlinear effect on the quality known as octane rating in the resulting blend.

AMPL makes it easy to express discrete or nonlinear models, but any departure from continuity or linearity is likely to make an optimal solution much harder to obtain. At the

least, it takes a more powerful solver to optimize the resulting mathematical programs. Chapters 17 through 20 discuss these issues in more detail.

Bibliography

George B. Dantzig, ''The Diet Problem.'' Interfaces **20**, 4 (1990) pp. 43–47. An entertaining account of the origins of the diet problem.

Susan Garner Garille and Saul I. Gass, ''Stigler's Diet Problem Revisited.'' Operations Research **49**, 1 (2001) pp. 1–13. A review of the diet problem's origins and its influence over the years on linear programming and on nutritionists.

Said S. Hilal and Warren Erikson, ''Matching Supplies to Save Lives: Linear Programming the Production of Heart Valves.'' Interfaces **11**, 6 (1981) pp. 48–56. A less appetizing equivalent of the diet problem, involving the choice of pig heart suppliers.

Exercises

2-1. Suppose the foods listed below have calories, protein, calcium, vitamin A, and costs per pound as shown. In what amounts should these food be purchased to meet at least the daily requirements listed while minimizing the total cost? (This problem comes from George B. Dantzig's classic book, *Linear Programming and Extensions*, page 118. We will take his word on nutritional values, and for nostalgic reasons have left the prices as they were when the book was published in 1963.)

	bread	meat	potatoes	cabbage	milk	gelatin	required
calories	1254	1457	318	46	309	1725	3000
protein	39	73	8	4	16	43	70 g.
calcium	418	41	42	141	536	0	800 mg.
vitamin A	0	0	70	860	720	0	500 I.U.
cost/pound	$0.30	$1.00	$0.05	$0.08	$0.23	$0.48	

2-2. (a) You have been advised by your doctor to get more exercise, specifically, to burn off at least 2000 extra calories per week by some combination of walking, jogging, swimming, exercise-machine, collaborative indoor recreation, and pushing yourself away from the table at mealtimes. You have a limited tolerance for each activity in hours/week; each expends a certain number of calories per hour, as shown below:

	walking	jogging	swimming	machine	indoor	pushback
Calories	100	200	300	150	300	500
Tolerance	5	2	3	3.5	3	0.5

How should you divide your exercising among these activities to minimize the amount of time you spend?

(b) Suppose that you should also have some variety in your exercise — you must do at least one hour of each of the first four exercises, but no more than four hours total of walking, jogging, and exercise-machine. Solve the problem in this form.

2-3. (a) A manufacturer of soft drinks wishes to blend three sugars in approximately equal quantities to ensure uniformity of taste in a product. Suppliers only provide combinations of the sugars, at varying costs/ton:

		SUPPLIER					
Sugar	A	B	C	D	E	F	G
Cane	10%	10	20	30	40	20	60
Corn	30%	40	40	20	60	70	10
Beet	60%	50	40	50	0	10	30
Cost/ton	$10	11	12	13	14	12	15

Formulate an AMPL model that minimizes the cost of supply while producing a blend that contains 52 tons of cane sugar, 56 tons of corn sugar, and 59 tons of beet sugar.

(b) The manufacturer feels that to ensure good relations with suppliers it is necessary to buy at least 10 tons from each. How does this change the model and the minimum-cost solution?

(c) Formulate an alternative to the model in (a) that finds the lowest-cost way to blend one ton of supplies so that the amount of each sugar is between 30 and 37 percent of the total.

2-4. At the end of Chapter 1, we indicated how to interpret the marginal (or dual) values of constraints and the reduced costs of variables in a production model. The same ideas can be applied to this chapter's diet model.

(a) Going back to the diet problem that was successfully solved in Section 2.3, we can display the marginal values as follows:

```
ampl: display Diet.lb,Diet.body,Diet.ub,Diet;
:   Diet.lb  Diet.body Diet.ub      Diet        :=
A       700    1956.29   20000    0
B1      700    1036.26   20000    0
B2      700     700      20000    0.404585
C       700    1682.51   20000    0
CAL   16000   19794.6    24000    0
NA        0   50000      50000   -0.00306905
;
```

How can you interpret the two that are nonzero?

(b) For the same problem, this listing gives the reduced costs:

```
ampl: display Buy.lb,Buy,Buy.ub,Buy.rc;
:    Buy.lb     Buy    Buy.ub      Buy.rc       :=
BEEF    2      5.36061    10     8.88178e-16
CHK     2      2          10     1.18884
FISH    2      2          10     1.14441
HAM     2     10          10    -0.302651
MCH     2     10          10    -0.551151
MTL     2     10          10    -1.3289
SPG     2      9.30605    10     0
TUR     2      2          10     2.73162
;
```

Based on this information, if you want to save money by eating more than 10 packages of some food, which one is likely to be your best choice?

2-5. A chain of fast-food restaurants operates 7 days a week, and requires the following minimum number of kitchen employees from Monday through Sunday: 45, 45, 40, 50, 65, 35, 35. Each employee is scheduled to work one weekend day (Saturday or Sunday) and four other days in a week. The management wants to know the minimum total number of employees needed to satisfy the requirements on every day.

(a) Set up and solve this problem as a linear program.

(b) In light of the discussion in Section 2.4, explain how this problem can be viewed as a special case of the blending model in Figure 2-4.

2-6. The output of a paper mill consists of standard rolls 110 inches (110") wide, which are cut into smaller rolls to meet orders. This week there are orders for rolls of the following widths:

Width	Orders
20"	48
45"	35
50"	24
55"	10
75"	8

The owner of the mill wants to know what cutting patterns to apply so as to fill the orders using the smallest number of 110" rolls.

(a) A cutting pattern consists of a certain number of rolls of each width, such as two of 45" and one of 20", or one of 50" and one of 55" (and 5" of waste). Suppose, to start with, that we consider only the following six patterns:

Width	1	2	3	4	5	6
20"	3	1	0	2	1	3
45"	0	2	0	0	0	1
50"	1	0	1	0	0	0
55"	0	0	1	1	0	0
75"	0	0	0	0	1	0

How many rolls should be cut according to each pattern, to minimize the number of 110" rolls used? Formulate and solve this problem as a linear program, assuming that the number of smaller rolls produced need only be greater than or equal to the number ordered.

(b) Re-solve the problem, with the restriction that the number of rolls produced in each size must be between 10% under and 40% over the number ordered.

(c) Find another pattern that, when added to those above, improves the optimal solution.

(d) All of the solutions above use fractional numbers of rolls. Can you find solutions that also satisfy the constraints, but that cut a whole number of rolls in each pattern? How much does your whole-number solution cause the objective function value to go up in each case? (See Chapter 20 for a discussion of how to find optimal whole-number, or integer, solutions.)

2-7. In the refinery model of Exercise 1-6, the amount of premium gasoline to be produced is a decision variable. Suppose instead that orders dictate a production of 42,000 barrels. The octane rating of the product is permitted to be in the range of 89 to 91, and the vapor pressure in a range of 11.7 to 12.7. The five feedstocks that are blended to make premium gasoline have the following production and/or purchase costs:

SRG	9.57
N	8.87
RF	11.69
CG	10.88
B	6.75

Other data are as in Exercise 1-6. Construct a blending model and data file to represent this problem. Run them through AMPL to determine the optimal composition of the blend.

2-8. Recall that Figure 2-4 generalizes the diet model as a minimum-cost input selection model, with constraints on the outputs.

(a) In the same way, generalize the production model of Figure 1-6a as a maximum-revenue output selection model, with constraints on the inputs.

(b) The concept of an ''input-output'' model was one of the first applications of linear programming in economic analysis. Such a model can be described in terms of a set A of activities and a set M of materials. The decision variables are the levels $X_j \geq 0$ at which the activities are run; they have lower limits u_j^- and upper limits u_j^+.

Each activity j has either a revenue per unit $c_j > 0$, or a cost per unit represented by $c_j < 0$. Thus total profit from all activities is $\sum_{j \in A} c_j X_j$, which is to be maximized.

Each unit of activity j produces an amount of material i given by $a_{ij} \geq 0$, or consumes an amount of material i represented by $a_{ij} < 0$. Thus if $\sum_{j \in A} a_{ij} X_j$ is > 0 it is the total production of material i by all activities; if < 0, it is the total consumption of material i by all activities.

For each material i, there is either an upper limit on the total production given by $b_i^+ > 0$, or a lower limit on the total consumption given by $b_i^+ < 0$. Similarly, there is either a lower limit on the total production given by $b_i^- > 0$, or an upper limit on the total consumption given by $b_i^- < 0$.

Write out a formulation of this model in AMPL.

(c) Explain how the minimum-cost input selection model and maximum-revenue output-selection model can be viewed as special cases of the input-output model.