7 Linear Programming

So far we have looked at modeling problems that involve quantities that change with time. Time, however, is not always part of the picture. In a modeling scenario that arises very often in economics, as well as in other sciences, one has to describe relationships between different quantities that are additionally subject to constraints. For instance, these quantities might be prices of raw materials and of finished products: how do the latter depend on the former? More to the point, modeling problems of this type often are set up in order to address optimization questions. For example, how can one set the prices of an array of finished products so as to maximize profit, given the costs of the raw materials and possibly constraints on their availability? The following simple example motivates the technique of *linear programming*, arguably one of the most widely used optimization algorithms. The example is from the Wikipedia entry for linear programming, `http://en.wikipedia.org/wiki/Linear_programming`.

Suppose that a farmer has a piece of farm land, say A square kilometers large, to be planted with either wheat or barley or some combination of the two. The farmer has a limited permissible amount F of fertilizer and P of insecticide which can be used, each of which is required in different amounts per unit area for wheat (F_1, P_1) and barley (F_2, P_2). Assume that the crop from one square kilometer of wheat can be sold at a price S_1, and that from one square kilometer of barley can be sold at a price S_2.The problem is to find the areas x_1 and x_2 to be planted with wheat and barley respectively.

In this problem, the *target function* is profit, which depends on the two variables x_1 and x_2 that are under the farmer's control. If we assume for simplicity that fertilizer and insecticide were purchased with earlier income, profit is simply the overall selling price:

$$f(x_1, x_2) = S_1 x_1 + S_2 x_2 \ ,$$

so the question is what combination of x_1 and x_2 yields the greatest value of $f(x_1, x_2)$. In the absence of any constraints, this function has obviously no maximum. However, the circumstances of the problem impose quite a few constraints on x_1 and x_2. First, these two areas cannot add up to more than the total available area A:

$$x_1 + x_2 \leq A \ . \tag{40}$$

This is called an *inequality constraint*. In addition, since the amount of fertilizer is limited to F, we have

$$F_1 x_1 + F_2 x_2 \leq F \ , \tag{41}$$

another inequality constraint. Similarly for insecticide:

$$P_1 x_1 + P_2 x_2 \leq P \ . \tag{42}$$

Two other constraints on x_1 and x_2 prevent meaningless solutions:

$$x_1 \geq 0 \quad \text{and} \quad x_2 \geq 0 \ , \tag{43}$$

since areas cannot be negative.

Note that profit, that is, the target function $f(x_1, x_2)$, is a linear function of the unknowns x_1 and x_2, and so are all the left-hand sides of the constraints. A constrained optimization problem when both the target function and the constraints are linear is called a *linear programming* problem.

Linearity is natural for the specific problem in the example above. In other situations, the functions involved may be nonlinear. In that case, much less can be said about the existence of a solution, let alone about methods for finding one. Sometimes, a nonlinear problem can be approximated with a collection of linear subproblems. In any event, the power of methods for solving linear optimization problems is so great that one often tries to restate the quantities involved so as to at least approximate linearity. Other, more involved techniques are available for some restricted classes of nonlinear problems. These techniques, however, are beyond the scope of this course.

It is useful to look at the problem above from a geometric point of view illustrated in Figure 19.

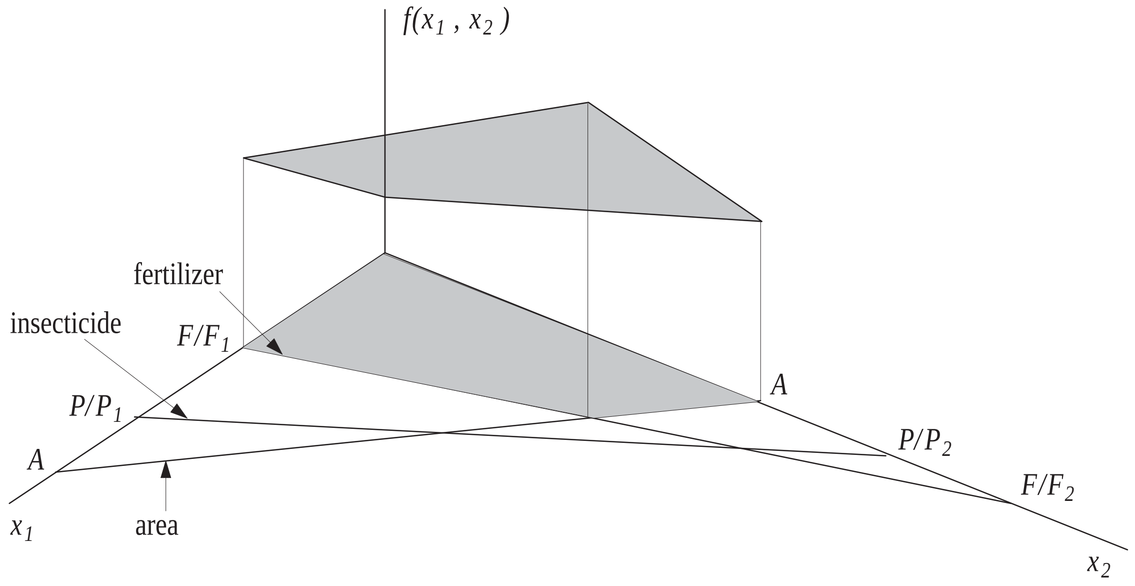

Figure 19: The shaded, convex polytope in the (x_1, x_2) plane is the feasible region of the constrained optimization problem in the text. The shaded area above is the graph of the target function $f(x_1, x_2)$ on the feasible region.

The three lines in the (x_1, x_2) plane are the boundaries of the inequalities (40) through (42). Together with the x_2 and x_1 axes (inequalities (43)), these lines define a *convex polytope*, that is, the intersection of five half-planes. Take a moment to convince yourself that the intersection of half-planes is always convex (although not always bounded). This polytope is called the *feasible region* of the problem, and is defined as the set of unknowns (x_1, x_2) that satisfy all the constraints of the problem. Each point in the feasible region (which includes the region's boundaries) is called a *feasible point* for the problem. Note that for the particular values chosen to draw Figure 19 the line corresponding to the constraint on the insecticide plays no role in defining the feasible region. If one or both intercepts of this line were decreased enough (for instance, by decreasing the value of P), a segment of this line would touch the feasible region, transforming it from a quadrangle to a pentagon.

Think of the target function $f(x_1, x_2)$ as a planar lid placed on top of the feasible region. Because this function is linear, it has no peaks or valleys, so it should be immediately obvious that the maximum of $f(x_1, x_2)$ over the feasible region is achieved at one of the corners of the region itself. To find the maximum (or, for that matter, the minimum) of the target function one merely has to compute the value of f at all vertices of the feasible region, and pick the vertex with the greatest (or smallest) value. With two variables and m constraints, the feasible polygon can have at most m vertices: with $m = 3$ the only polytope is a triangle, which has 3 vertices. Each additional side can only cut at most two existing sides of the polygon, thereby replacing a vertex with a new side (two vertices), that is, adding one net vertex to the count.

This seems to make this type of problems trivial to solve. However, this is not so, and the simplicity of the picture resulting from the example is misleading. In a more complex example, more constraints would result in a possibly more complex feasible region, but always a convex one. More unknowns (in addition to x_1, x_2)) would result into a multi-dimensional polytope rather than a polygon. For instance, with three unknowns the feasible polytope would be a convex polyhedron.

While it is not straightforward to visualize what happens as the number of unknowns increases, it is known that the maximum number of vertices of a polytope in n dimensions is an exponential function of n. For instance, a cube in n dimensions is defined as the polytope whose vertices have coordinates $(b_1, \ldots, b_n)$ where b is either 0 or 1. A cube in 1 dimension is the segment $[0, 1]$; in 2 dimensions it is the square with vertices $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$; in 3 dimensions, it is the regular cube with unit-length sides, and so forth. We see easily that the number of vertices of an n-dimensional cube is 2^n (the number of ways to select combinations of n zeros and ones), and this is exponential in the number of dimensions.

This is why the constrained optimization problems of the type illustrated by the example above become very difficult to solve: with hundreds or even thousands of variables in nontrivial problems, checking a number of vertices that is exponential in the number of variables is just not an option.

In 1947, George B. Dantzig published his now famous *simplex method* for solving complex linear programming problems, just around the time when John Von Neumann built Eniac, the first electronic computer. This convergence of software and hardware was a revolutionary development, and started the field of numerical optimization.

The Standard Linear Programming Problem

In addition to the inequality constraints illustrated in the previous example, a linear programming problem may also have *equality constraints*, which of course we again assume to be linear. With two variables, an equality constraint would be of the form

$$ax_1 + bx_2 = c$$

rather than

$$ax_1 + bx_2 \leq c\ .$$

With two variables, such a constraint would make the problem rather trivial. In Figure 19, an equality constraint would take the form of an additional line, and the solution (x_1^*, x_2^*) of the linear programming problem would be required to be on that line (which , mind you, is still a (trivial) polytope). With more variables, equality constraints become much more interesting, but remain convex polytopes nonetheless.

Thus, there is quite a variety of linear programming problems: minimization, maximization, with equality constraints, with "greater than" inequalities, with "less than" inequalities, and any combinations of these. To stem this variety, it is useful to define a canonical form of linear program, into which any particular problem can be translated.

The *Standard Linear Program (SLP)* is defined as follows:

$$\min c_1x_1 + \ldots + c_nx_n$$

subject to equality constraints

$$\begin{aligned} a_{11}x_1 + \ldots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{m1}x_1 + \ldots + a_{mn}x_n &= b_m \end{aligned}$$

and inequality constraints

$$x_1 \geq 0\,,\ \ldots\,,\ x_n \geq 0\ .$$

Note the particularly simple form of the inequality constraints: all we want is for the solution to be in the positive octant.

Any linear programming problem can be transformed into an equivalent SLP. For instance, a maximization problem,

$$\max c_1x_1 + \ldots + c_nx_n$$

can be transformed to a minimization problem by replacing $c_1, \ldots, c_n$ by $-c_1, \ldots, -c_n$. A problem in which the inequalities are of the form

$$\begin{aligned} \alpha_{11}x_1 + \ldots + \alpha_{1n}x_n &\leq \beta_1 \\ &\vdots \\ \alpha_{k1}x_1 + \ldots + \alpha_{kn}x_n &\leq \beta_k \end{aligned}$$

can be transformed into standard form by introducing k new, so-called *slack variables* $s_1, \ldots, s_k$

with equality constraints

$$\begin{aligned}\alpha_{11}x_1 + \ldots + \alpha_{1n}x_n + s_1 &= \beta_1\\ &\vdots\\ \alpha_{k1}x_1 + \ldots + \alpha_{kn}x_n + s_k &= \beta_k\end{aligned}$$

and inequality constraints

$$s_1 \geq 0\,,\ \ldots\,,\ s_k \geq 0\,.$$

The new problem is not quite in standard form yet, since SLP requires *all* unknowns (including $x_1, \ldots, x_n$) to be nonnegative. This is achieved by "splitting" each variable x_i into the difference of its positive and negative part:

$$x_i = x_i^+ - x_i^-$$

where

$$x_i^+ = \max(x, 0) \quad \text{and} \quad x_i^- = \max(-x, 0)\,.$$

In this way, we have

$$x_i^+ \geq 0 \quad \text{and} \quad x_i^- \geq 0$$

as required. Finally, "greater than" inequalities of the form

$$\begin{aligned}\alpha_{11}x_1 + \ldots + \alpha_{1n}x_n &\geq \beta_1\\ &\vdots\\ \alpha_{k1}x_1 + \ldots + \alpha_{kn}x_n &\geq \beta_k\end{aligned}$$

can be converted to "less than" inequalities by changing the signs of all coefficients:

$$\begin{aligned}-\alpha_{11}x_1 - \ldots - \alpha_{1n}x_n &\leq -\beta_1\\ &\vdots\\ -\alpha_{k1}x_1 - \ldots - \alpha_{kn}x_n &\leq -\beta_k\end{aligned}$$

and these in turn can be standardized by introducing slack variables.

These transformations make the number of variables proliferate, so it is reassuring to know that the simplex algorithm works efficiently even with many variables.

Importantly, the SLP still requires to optimize (minimize) a linear function subject to linear equality and inequality constraints. Thus, the feasible region is still a convex polytope, and the minimum is still achieved at one of its corners.

The Idea of the Simplex Algorithm

In principle, Dantzig's simplex algorithm for solving SLP is very simple: start at a vertex of the polytope, and move to a neighbor of that vertex that leads to a reduction in the value of the target function. Since the polytope has a finite (albeit large) number of vertices, the algorithm is guaranteed to eventually converge.

In practice, however, there are three key difficulties. The first one is to find a vertex to start from. It turns out that this problem is equivalent to solving yet another SLP. This would seem to be a show-stopper. Fortunately, it is possible to set up an SLP problem that can be initialized in a trivial fashion, and whose solution (computed with the simplex algorithm) provides an initial vertex for the original SLP. So two runs of the simplex algorithm do the trick. How to do this exactly is technical. The gist of the idea is conveyed by the following method for finding some feasible point, rather than a vertex on the feasible polytope.[24]

Given the original standard linear program (see page 88)

$$\min c_1x_1 + \ldots + c_nx_n$$

subject to equality constraints

$$\begin{aligned} a_{11}x_1 + \ldots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{m1}x_1 + \ldots + a_{mn}x_n &= b_m \end{aligned}$$

and inequality constraints

$$x_1 \geq 0\,,\ \ldots\,,\ x_n \geq 0\,,$$

consider the following auxiliary standard linear program:

$$\min z_1 + \ldots + z_m$$

subject to equality constraints

$$\begin{aligned} a_{11}x_1 + \ldots + a_{1n}x_n + e_1z_1 &= b_1 \\ &\vdots \\ a_{m1}x_1 + \ldots + a_{mn}x_n + e_mz_m &= b_m \end{aligned}$$

and inequality constraints

$$x_1 \geq 0\,,\ \ldots\,,\ x_n \geq 0\,,\ z_1 \geq 0\,,\ \ldots\,,\ z_n \geq 0\,.$$

Here, the terms e_i are defined as follows:

$$e_i = \begin{cases} 1 & \text{if } b_i \geq 0 \\ -1 & \text{if } b_i < 0 \end{cases}\,.$$

This auxiliary problem minimizes the sum of the violations z_i of the equality constraints in the original SLP. Since the target function $z_1 + \ldots + z_m$ cannot be negative (because of the inequality

[24]A feasible-polytope vertex is called a *basic* feasible point in the literature.

constraints $z_1 \geq 0, \ldots, z_n \geq 0$), the smallest possible target function value for a solution to the auxiliary problem is zero. If the *original* SLP admits a feasible point, there must be values $\tilde{x}_1, \ldots, \tilde{x}_n$ that satisfies all the constraints of the original problem, so the point

$$x_1 = \tilde{x}_1 \,,\; \ldots \,,\; x_n = \tilde{x}_n \,,\; z_1 = 0 \,,\; \ldots \,,\; z_n = 0$$

satisfies all the constraints of the auxiliary problem *and* yields an optimal point (target function value of zero) for the auxiliary problem.

This means that if the original problem admits a feasible point and if we can find a solution for the auxiliary problem, then the latter has a target function value of zero, achieved at a point where the values $x_i = \tilde{x}_i$ yield a feasible value for the original problem.

So, are we better off? We have just found that a solution to the auxiliary problem *would* give us a feasible point for the original problem. Haven't we just changed the problem to that of finding a feasible point for a different problem, the auxiliary one? Yes, but it is very easy to find a feasible point for the latter. Just pick

$$x_1 = 0 \,,\; \ldots \,,\; x_n = 0 \,,\; z_1 = |b_1| \,,\; \ldots \,,\; z_n = |b_m| \,.$$

Verify that these values do satisfy all the constraints of the auxiliary problems, and constitute therefore a feasible point for it.

The second difficulty in implementing Dantzig's idea is that finding the best polytope neighbor vertex to move to at each iteration is an expensive problem in itself. Trying all neighbors would be prohibitive. Dantzig's idea was based on an important observation, which here we give without proof:[25] in the standard formulation, the vertices of the polytope are points $(x_1, \ldots, x_n)$ where at most m of the variables are nonzero. Recall that m is the number of equality constraints. One can then move from a vertex to a neighbor by (i) zeroing one of the current nonzero unknowns (the so-called *leaving* variable x_l) and changing a currently zero unknown (the *entering* variable x_e) to a nonzero value that ensures to land us at a neighboring vertex of the polytope. Since there are at most m nonzero variables at any iteration, there are at most m ways to choose the leaving variable, and at most $n - 1$ ways to choose the entering variable. Dantzig proposed to choose any combination that reduces the target value. More modern versions of the simplex algorithm choose different compromises between a greater computational cost and a greater decrease of the target function value at every iteration.

The third difficulty of the simplex method is conceptually fundamental: in perverse situations, it is theoretically possible that the simplex algorithm traverses *all* the vertices of the polytope on its way to a minimum. In 1972, Klee and Minty showed a pathological SLP in which this is actually the case, thereby proving that the computational complexity of the simplex algorithm is worst-case exponential in the number of variables. Somewhat surprisingly, most practical SLPs stay well away from this type of situations, and the simplex algorithm is very efficient even with thousands of variables. Still, the theoretical difficulty remained until 1984, when Narendra Karmarkar developed an algorithm that has provably polynomial complexity, and is practically rather simple. The main idea of Karmarkar's so-called *interior-point method* is to approach the minimum of the target

[25] We also gloss over a few technical requirements, which are easy to meet in practice.

function from inside the feasibility polytope, rather than by walking along its boundary one vertex at a time. Still, years of refinements have kept the simplex algorithm practically very fast, and often faster than the interior-point methods. Many numerical packages today still use variants of the simplex algorithm as their main engine for linear programming.

Matlab provides a function `linprog` for linear programming. This uses an interior-point method for large problems, and a variant of the simplex method (in two different versions) for smaller problems. This function accepts problem formats that are somewhat more flexible than SLP, so some of the standardization work is done for you. Specifically, the help page for `linprog` describes the following format (adapted to our notation, but in more compact vector form):

$$\min_{\mathbf{x}} \mathbf{c}^T\mathbf{x}$$

subject to equality constraints

$$A_{eq}\mathbf{x} = \mathbf{b}_{eq}$$

and to inequality constraints

$$\begin{aligned} A\mathbf{x} &\leq \mathbf{b} \\ \mathbf{l} &\leq \mathbf{x} \leq \mathbf{u}\,. \end{aligned}$$

As an example, here is code that sets up and solves the LP problem illustrated earlier, for some specific numerical values of the various parameters. Note that all the work is in the single line that calls `linprog`.

```
ft = 10;      % Total amount of fertilizer available
pt = 24;      % Total amount of pesticide available
fu = [3 2];   % Amounts of fertilizer needed per unit area of wheat or barley
pu = [8 3];   % Amounts of pesticide needed per unit area of wheat or barley
at = 4;       % Total available area
f = [3 2.01];% Profits from a unit area planted with wheat or barley

% The four rows of A and b correspond to:
% + fertilizer limit
% + pesticide limit
% + area limit
% + nonnegative solution
A = [fu; pu; ones(1, 2); -eye(2)];
b = [ft; pt; at; zeros(2, 1)];

x = linprog(-f, A, b);  % Linprog minimizes, hence the minus sign

figure(1)
clf
hold on

% x1 and x2 intercepts for the first three constraints
intercept = (b(1:3) * ones(1, 2)) ./ A(1:3, :);

% List both coordinates for each intercept
x1 = [intercept(:, 1)'; zeros(1, 3)];
```

```
x2 = [zeros(1, 3); intercept(:, 2)'];

plot(x1, x2)
plot(x(1), x(2), '.k', 'MarkerSize', 18)
ax = [0 max(intercept(:, 1)) 0 max(intercept(:, 2))];
axis(ax)
title(sprintf('Pricing: wheat %.02f, barley %0.2f', f))
xlabel('Area for wheat')
ylabel('Area for barley')

legend('Fertilizer', 'Pesticide', 'Area')
```

A More Complex Problem

This Section sets up a more complex scenario that leads to a larger LP problem. This scenario belongs to a special class of LP problems called *network flow* problems. Because of the special structure of these problems, more specialized, and therefore more efficient algorithms than the simplex method could and are often used for their solution, especially when the number of variables is very large. However, for problems of moderate size, standard LP algorithms are adequate even for network flow problems.

BioPower is a company that runs a number of facilities that convert biomass (such as wood chips, switchgrass, and so forth) to diesel fuel. In order to operate, these conversion plants must purchase raw materials from various sources such as city wood waste repositories, farms that sell cereal straw or switchgrass, lumber mills that produce wood chips as waste, and so forth. Each source produces a limited amount of biomass, and transporting the materials from source to destination costs an amount per unit of material that depends on both distance traveled and the density of the material (a low density material such as straw may require more truck trips per ton than a higher density material such as wood chips). Each of BioPower's facilities processes materials in different ways, and BioPower's problem is to decide how much material to acquire from what source in order to minimize the overall cost.

A somewhat simplistic model of this situation can be defined as follows. Let the S sources be at locations $\mathbf{s}_1, \ldots, \mathbf{s}_S$, and the P plants be at locations $\mathbf{p}_1, \ldots, \mathbf{p}_P$ (the $\mathbf{s}_s$ and $\mathbf{p}_p$ are points on the plane, with coordinates measured in kilometers).

Assume that there are $T = 3$ types of biomass: (1) wood waste, (2) cereal straw, and (3) switchgrass. The cost of a ton of biomass of type k is d_k dollars, and it costs δ_k dollars to ship a ton of biomass of type k over a distance of one kilometer. An ideal plant with 100 percent efficiency would produce u_k barrels of diesel fuel from a ton of biomass of type k.

Source s can produce at most σ_s tons of type t_s biomass per year (t_s is either 1, 2, or 3). Plant p can convert any type of biomass. It does so at efficiency η_{pk} (a real number between 0 and 1) for biomass of type k, and it can produce an overall amount of fuel equal to β_p barrels per year.

Here is a numerical example in Matlab:

```
% Costs of one ton of each type of biomass, in dollars
```

```
d = [60, 20, 10]; % Wood chips, straw, switchgrass

% Costs of transporting one ton of each type of biomass, in dollars per km
delta = [0.3, 0.4, 0.2];

% Source locations
s = [0 0; 0 200; 150 30; 80 140; 200 0];

% Type of biomass for each source
t = [1 1 3 2 1];

% Biomass production capacities in tons per year for each source
sigma = 1000 * [20 40 30 50 60];

% Plant locations
p = [25 50; 100 0; 150 130; 20 180];

% Diesel production capacities in barrels per year for each plant
beta = 1000 * [10 8 20 5];

T = length(d);   % Types of biomass (wood chips, straw, switchgrass)
S = size(s, 1);  % Number of biomass sources
P = size(p, 1);  % Number of processing plants

% Diesel production efficiencies for each plant and type of biomass
eta = [1 0.5 0.8 0.3]' * [0.4 0.2 0.3];     % Separable, for simplicity

% Barrels per ton of biomass for an ideal plant (100 percent efficiency)
u = 1.2 * ones(1, T);       % Same for every biomass type, for simplicity
```

In a first attempt, we are tempted to ask the following question:

How much biomass should each of BioPower's plants purchase from each source to minimize costs?

However, a moment's reflection should reveal that this question has a trivial answer: if BioPower does not buy anything, it does not have any costs. A more meaningful question is as follows:

Assuming that BioPower wants to produce diesel fuel at maximum capacity, how much biomass should each of its plants purchase from each source to minimize costs?

This version of the question avoids the trivial solution, and introduces an equality constraint ("production equals total capacity") instead of an inequality constraint ("production does not exceed total capacity"). Of course, rather than a trivial answer, we should now be prepared to obtain possibly no answer: if all the sources combined do not make enough biomass to satisfy BioPower's needs at full production capacity, no feasible solution exists. We can let the simplex algorithm discover this.

For simplicity in formulating the problem, we assign two subscripts to each unknown: ξ_{sp} is the amount that plant p purchases from source s, for $s = 1, \ldots, S$ and $p = 1, \ldots, P$. When needed, we can package these variables into a single vector $\mathbf{x} = (x_1, \ldots, x_n)$ where $n = SP$ and, say,

$$x_{P(s-1)+p} = \xi_{sp}$$

The target function for this problem is the overall cost of the biomass:

$$C(\mathbf{x}) = \sum_{s=1}^{S}\sum_{p=1}^{P} \gamma_{sp}\xi_{sp}$$

where γ_{sp} is the overall cost of a ton of biomass, including transportation from source s to plant p. Again, these costs can be arranged into a single vector $\mathbf{c} = (c_1, \ldots, c_n)$ where

$$c_{P(s-1)+p} = \gamma_{sp}$$

so we have a total cost

$$C(\mathbf{x}) = \mathbf{c}^T\mathbf{x} = c_1x_1 + \ldots + c_nx_n$$

which BioPower wants to minimize. The unit costs γ_{sp} can be calculated as follows:

$$\gamma_{sp} = d_{t_s} + \delta_{t_s}\|\mathbf{s}_s - \mathbf{p}_p\|$$

where the first term is the cost of a ton of material and the second term is the cost of transportation for a ton of material over the distance $\|\mathbf{s}_s - \mathbf{p}_p\|$ between source s and plant p.

Several constraints must hold. First, no source can sell more than it can produce:

$$\sum_{p=1}^{P} \xi_{sp} \leq \sigma_s \; . \tag{44}$$

Second, each plant should purchase what it can process at full capacity:

$$\sum_{s=1}^{S} \pi_{sp}\xi_{sp} = \beta_p \tag{45}$$

where

$$\pi_{sp} = \eta_{pt_s}u_{t_s}$$

is the number of barrels of diesel fuel plant p can produce from a ton of biomass of type t_s.

Make sure you understand where the term $\pi_{sp}\xi_{sp} = \eta_{pt_s}u_{t_s}\xi_{sp}$ comes from: ξ_{sp} tons of type t_s biomass from source s produce ideally u_{t_s} barrels of diesel per ton, but plant p has only efficiency η_{pt_s} in converting type t_s biomass to diesel.

Third, all quantities are nonnegative:

$$\xi_{sp} \geq 0 \; .$$

Thus, this problem involves both inequality (equation (44)) and equality (equation (45)) constraints.

The only tricky part in Matlab is to package all the constraints (except for the nonnegativity constraints, which can be entered separately) into matrix equations

$$A\mathbf{x} \leq \mathbf{b}$$

for the inequality constraints and

$$A_{eq}\mathbf{x} = \mathbf{b}_{eq}$$

for the equality constraints.

The packaging

$$x_{P(s-1)+p} = \xi_{sp}$$

lists all unknowns for source 1, followed by all unknowns for source 2, ..., followed by all unknowns for source S. A moment's reflection will show that the desired matrix A for the inequality constraints (44) has S rows (one per source) and $n = PS$ columns (one per unknown):

$$A = \begin{bmatrix} 1 & \ldots & 1 & 0 & \ldots & 0 & \ldots & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 1 & \ldots & 1 & \ldots & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 0 & 0 & \ldots & 0 & \ldots & 1 & \ldots & 1 \end{bmatrix} .$$

Similarly, the matrix A_{eq} for the equality constraints (45) has P rows (one per plant) and n columns:

$$A_{eq} = \begin{bmatrix} \pi_{11} & 0 & \ldots & 0 & \pi_{21} & 0 & \ldots & 0 & \ldots & \pi_{S1} & 0 & \ldots & 0 \\ 0 & \pi_{12} & \ldots & 0 & 0 & \pi_{22} & \ldots & 0 & \ldots & 0 & \pi_{S2} & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & \pi_{1P} & 0 & 0 & \ldots & \pi_{2P} & \ldots & 0 & 0 & \ldots & \pi_{SP} \end{bmatrix} .$$

The right-hand side vector $\mathbf{b}$ of the inequality constraints has S rows (one per source), and $\mathbf{b}_{eq}$ has P rows:

$$\mathbf{b} = \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_S \end{bmatrix} \quad \text{and} \quad \mathbf{b}_{eq} = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_P \end{bmatrix} .$$

In Matlab, this can be achieved with many nested `for` loops (the safest alternative if you are not comfortable with matrix processing), or with a little array wizardry:

```
n = S*P;     % Number of unknowns

% Diesel production efficiencies for each source/plant pair
pi = eta(:, t)' .* (u(t)' * ones(1, P) );

% Source/plant distances
dist = zeros(S, P);
for k = 1:2
    dist = dist + (s(:, k) * ones(1, P) - ones(S, 1) * p(:, k)') .^ 2;
end
dist = sqrt(dist);

% Diesel costs per ton for each source/plant pair
```

```
gamma = d(t)' * ones(1, P) + (delta(t)' * ones(1, P)) .* dist;

% Same, packaged into a vector
c = gamma';
c = c(:);

% Upper and lower bounds on the unknowns
lb = zeros(n, 1);
ub = Inf * ones(n, 1);

% Constraint matrices
A = kron(eye(S), ones(1, P));
Aeq = zeros(P, n);
for k = 1:S
    Aeq(:, (((k-1)*P)+1):(k*P)) = diag(pi(k, :));
end

% Constraint vectors
b = sigma(:);
beq = beta(:);
```

If you are interested in how this all works, look up the definition of the various operations used, including the *Kronecker product* `kron`, in the Matlab Help. For the purposes of this course, using nested `for` loops instead is perfectly fine.

All we need to do to solve the problem is a single call to `linprog`, plus an instruction to repackage the output vector $\mathbf{x}$ into a matrix ξ:

```
x = linprog(c, A, b, Aeq, beq, lb, ub);
xi = reshape(x, P, S)';
```

The solution to the numerical example discussed above is shown in Figure 20. Try to understand the rationale of this solution by relating it back to the various costs and capacities.

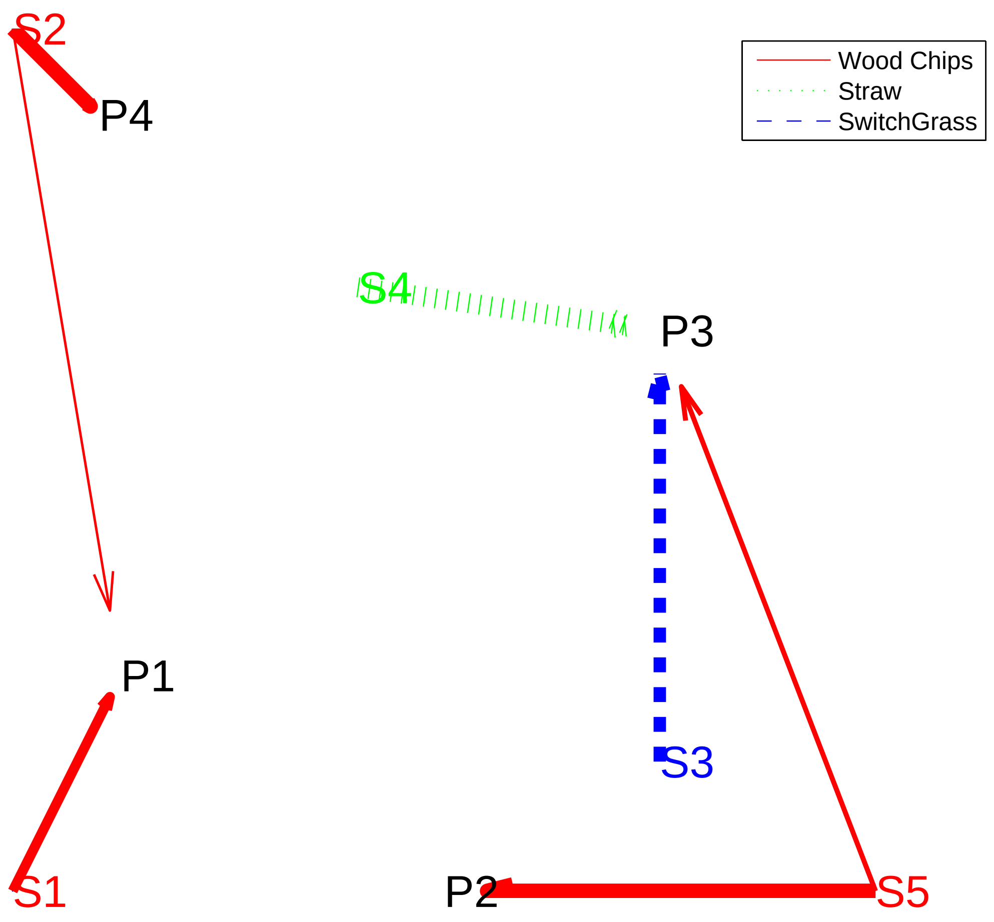

Figure 20: Supply paths from sources (S) to plants (P) that minimize the costs of biomass to achieve full diesel production capacity. The thickness of each arrow is proportional to the amount supplied.