17.1 Theory

17.1.1 Linear Programming Problems

Linear programming problems are a special type of optimization problem. The general linear programming problem can be formulated as follows:

Find \( x_1, x_2, \ldots, x_n \) to

Minimize the objective function

\[
    c = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n
\]

Subject to the constraints

\[
    a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n \leq b_1
\]

\[
    a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n \leq b_2
\]

\[
    \vdots
\]

\[
    a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n \leq b_m
\]

The linear in linear programming comes from the fact that the objective function and constraints are linear functions of the variables \( x_1, x_2, \ldots, x_n \).

Instead of the inequality \( \leq \) in the constraints, the inequality \( \geq \) or equality \( = \) may be used. This seemingly greater generality does not lead to a larger class of problems since a constraint of the form \( a \geq b \) can be converted to \( -a \leq -b \) and an equality constraint \( a = b \) is equivalent to the pair of constraints \( a \leq b \) and \( -a \leq -b \). Similarly maximizing an objective function \( c \) is equivalent to minimizing the function \( -c \).

The feasible region is the set of points \( (x_1, \ldots, x_n) \in \mathbb{R}^n \) satisfying the constraints of the problem.

17.1.2 An Example

A transport company needs to move sand between three sources X, Y and Z and three destinations A, B and C. Each source has a limited supply of sand:

<table>
<thead>
<tr>
<th>Source</th>
<th>Truckloads available</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>35</td>
</tr>
<tr>
<td>Y</td>
<td>40</td>
</tr>
<tr>
<td>Z</td>
<td>40</td>
</tr>
</tbody>
</table>

Each destination has a minimum requirement:

<table>
<thead>
<tr>
<th>Destination</th>
<th>Truckloads required</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>45</td>
</tr>
<tr>
<td>B</td>
<td>50</td>
</tr>
<tr>
<td>C</td>
<td>15</td>
</tr>
</tbody>
</table>
Transportation costs, in $ per truckload, are:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>5</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Y</td>
<td>20</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>Z</td>
<td>5</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

The problem is to find a transportation schedule to minimize the cost of transporting the sand.

The formulation of a linear programming problem requires three steps:

1. Identify the **variables**.

2. Write down the **objective function**.

3. Write down the **constraints**.

In this problem there are nine variables; the number of truckloads of sand moved between each of the three sources and each of the three destinations. Denote these variables by $N_{XA}$, $N_{XB}$, $N_{XC}$, $N_{YA}$, $N_{YB}$, $N_{YC}$, $N_{ZA}$, $N_{ZB}$ and $N_{ZC}$, where, for example, $N_{YA}$ denotes the number of truckloads of sand transported from Y to A.

The objective is to minimize the total transportation cost. Since we know the cost of transporting one truckload of sand from any source to any destination, it is easy to express the cost as a function of the variables:

$$c = 5N_{XA} + 10N_{XB} + 10N_{XC} + 20N_{YA} + 30N_{YB} + 20N_{YC} + 5N_{ZA} + 8N_{ZB} + 12N_{ZC}$$

There are two sets of constraints relating, respectively, to the supply and demand for sand.

Constraints on supply:

$$N_{XA} + N_{XB} + N_{XC} \leq 35$$
$$N_{YA} + N_{YB} + N_{YC} \leq 40$$
$$N_{ZA} + N_{ZB} + N_{ZC} \leq 40$$

Constraints on demand:

$$N_{XA} + N_{YA} + N_{ZA} \geq 45$$
$$N_{XB} + N_{YB} + N_{ZB} \geq 50$$
$$N_{XC} + N_{YC} + N_{ZC} \geq 15$$

It may seem more natural to have used equality constraints on the demands; however whether we use equality or $\geq$ constraints we will get the same
answer (why?). The inequality constraints are more convenient when solving the problem in Scilab.

Finally, there are the easily overlooked constraints that each of the variables is non-negative:

\[
N_{XA} \geq 0 \quad N_{XB} \geq 0 \quad N_{XC} \geq 0 \\
N_{YA} \geq 0 \quad N_{YB} \geq 0 \quad N_{YC} \geq 0 \\
N_{ZA} \geq 0 \quad N_{ZB} \geq 0 \quad N_{ZC} \geq 0
\]

Requiring all variables to be non-negative is very common in linear programming problems.

### 17.1.3 Geometry of 2D Problems

We will look at a two dimensional linear programming problem which illustrates the geometric side to the problem:

Minimize

\[
c = -2x + y
\]

Subject to

\[
2x + 3y \geq 6 \\
-x + y \leq 3 \\
x \leq 2 \\
x \geq 0 \quad y \geq 0
\]

The feasible region for this problem is shown Figure 1.

Figure 1: Feasible region
Recall that the level sets of a function \( f(x_1, x_2, \ldots, x_n) \) are the sets \( f(x_1, x_2, \ldots, x_n) = \text{constant} \). The level sets of the objective function \( c = -2x + y \) are the lines

\[-2x + y = \text{constant}\]

**Figure 2: Level Sets of the Objective Function**

From this diagram it is apparent that the minimum of the objective function occurs at the vertex \((2, 2/3)\) of the feasible region.

This example is typical of linear programming problems in two variables:

1. The feasible region is a convex polygonal region of the plane bounded by the constraints.

2. If the feasible region is non-empty and bounded, then the linear programming problem has a solution.

3. The solution, if it exists, occurs at a vertex of the feasible region.

If we remove the constraint \( x \leq 2 \) we obtain an unbounded feasible region in which our objective function has no minimum.

**17.1.4 The Simplex Algorithm**

Geometrically, the general linear programming problem is similar to the two-dimensional problem.

1. The feasible region is a convex polygonal region of \( \mathbb{R}^n \). It is bounded by hyperplanes, i.e. \((n - 1)\)-dimensional planes in \( \mathbb{R}^n \), determined by the constraints.

2. The level sets of the objective function are also hyperplanes in \( \mathbb{R}^n \).
3. If the feasible region is non-empty and bounded, then the linear programming problem has a solution.

4. The solution, if it exists, occurs at a vertex of the feasible region.

The standard method for solving linear programming problems is the **Simplex Algorithm**. The algorithm is easy to describe: it simply searches through the vertices of the feasible region in a systematic way until the minimum is found. Practical linear programming problems with thousands of variables and constraints are not unusual.

### 17.2 Scilab

Scilab’s linear programming solver (based on the simplex algorithm) is `linpro`. It is used in the form

\[
x = \text{linpro}(\text{obj}, \text{cnstr}, \text{rhs}, \text{lb}, \text{ub})
\]

We will illustrate its use on the two dimensional problem of §1.3:

1. The first argument, `obj`, is a column vector containing the coefficients of the objective function to be minimized.

```plaintext
--->obj = [-2 1]
    obj =

      -2.
      1.
```

Figure 3: An Unbounded Feasible Region
2. The second argument, \texttt{cnstr}, is a matrix containing the coefficients of the constraints. Each row of the matrix contains the coefficients of one constraint. The constraints are assumed to be \( \leq \) constraints. Note that the first constraint of our example is a \( \geq \) constraint, so we need to take its negative.

\[\text{cnstr} = \begin{bmatrix} -2 & -3 \\ -1 & 1 \\ 1 & 0 \end{bmatrix}\]

3. The argument \texttt{rhs} is the column vector containing the coefficients on the right hand side of the constraints. As above the first is negative.

\[\text{rhs} = \begin{bmatrix} -6 \\ 3 \\ 2 \end{bmatrix}\]

4. The final two arguments, \texttt{lb} and \texttt{ub} are column vectors of lower bounds and upper bounds on the variables. In our example both variables have lower bounds of zero and no upper bounds (other than those determined by the constraints). This a common situation in linear programming. If there are no bounds of some type an empty matrix, [] can be used.

\[\text{lb} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}\]

\[\text{ub} = []\]

Now we can compute the solution:

\[\text{x} = \text{linpro(obj, cnstr, rhs, lb, ub)}\]
which agrees with the solution we found earlier using geometric arguments.

Example

We will now use Scilab solve the problem of §1.2.

In our problem there are nine variables: $N_{XA}, N_{XB}, N_{XC}, N_{YA}, N_{YB}, N_{YC}, N_{ZA}, N_{ZB}$ and $N_{ZC}$. We will take the variables in this order so the objective is:

```scilab
-->obj = [5 10 10 20 30 20 5 8 12]'
obj =

! 5. !
! 10. !
! 10. !
! 20. !
! 30. !
! 20. !
! 5. !
! 8. !
! 12. !
```

There are six constraints, the last three of which are $\geq$ constraints which we need to multiply by -1. The variables appearing in each constraint follow a pattern:

```scilab
-->cnstr = [1 1 1 0 0 0 0 0 0
-->0 0 0 1 1 1 0 0 0
-->0 0 0 0 0 1 1 1
-->-1 0 0 -1 0 0 -1 0 0
-->0 -1 0 0 -1 0 0 -1 0
-->0 0 -1 0 0 -1 0 0 -1]
```

```scilab
cnstr =

! 1. 1. 1. 0. 0. 0. 0. 0. 0. !
! 0. 0. 0. 1. 1. 1. 0. 0. 0. !
! 0. 0. 0. 0. 0. 1. 1. 1. 1. !
! -1. 0. 0. -1. 0. 0. -1. 0. 0. !
! 0. -1. 0. 0. -1. 0. 0. -1. 0. !
! 0. 0. -1. 0. 0. -1. 0. 0. -1. !
```
We need to take account of the sign changes noted above in entering the right-hand-side vector:

```
-->rhs = [35 40 40 -45 -50 -15]
``` 

```
| 35. |
| 40. |
| 40. |
| -45. |
| -50. |
| -15. |
```

Each variable has a lower bound of zero, and there are no explicit upper bounds:

```
-->lb = [0 0 0 0 0 0 0 0 0]
``` 

```
| 0. |
| 0. |
| 0. |
| 0. |
| 0. |
| 0. |
| 0. |
| 0. |
| 0. |
```

```
-->ub = []
``` 

```
[]
```

Now we can solve the problem in Scilab:

```
-->x = linpro(obj, cnstr, rhs, lb, ub)
``` 

```
| 25. |
| 10. |
| 0. |
| 20. |
| 6.328E-15 |
| 15. |
| -9.992E-16 |
```
The tiny numbers are just noise due rounding error.
This result can be summarized in the table:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>25</td>
<td>10</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>Y</td>
<td>20</td>
<td>0</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>Z</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>Total</td>
<td>45</td>
<td>50</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

To find the total cost we evaluate the objective function at the optimal solution found above:

--->c = obj’*x

\[ c = 1245. \]