Modelling Using LINGO

Use of LINGO software package for optimisation, sometimes known as mathematical programming, is discussed here.
LINGO is a trademark of LINDO Systems, Inc.
LINGO has the ability to solve linear and nonlinear models.
This package contains set or vector notation for compactly representing large models.

An Optimisation Example Problem: The Reddy Mikks Company

Reddy Mikks produces both interior and exterior paints from two raw materials, M\textsubscript{1} and M\textsubscript{2}. The following table provides the basic data of the problem:

<table>
<thead>
<tr>
<th></th>
<th>Tons of raw material per ton of</th>
<th>Maximum daily availability (tons)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw material, M\textsubscript{1}</td>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>Raw material, M\textsubscript{1}</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Profit per ton (Rs 1000)</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

A market survey indicates that the daily demand for interior paint cannot exceed that of exterior paint by more than 1 tone. Also, the maximum daily demand of interior paint is 2 tons.

Reddy Mikks wants to determine the optimum (best) product mix of interior and exterior paints that maximizes the total daily profit.

Formulation

\( x_1 = \text{Tons produced daily of exterior paint} \)
\( x_2 = \text{Tons produced daily of interior paint} \)

The model is

Maximise

\[ z = 5x_1 + 4x_2 \]

Subjected to

\[ 6x_1 + 4x_2 \leq 24 \]
\[ x_1 + 2x_2 \leq 6 \]
\[ -x_1 + x_2 \leq 1 \]
\[ x_2 \leq 2 \]
\[ x_1, x_2 \geq 0 \]

Alternate way of writing the model

Let \( x_j \) be the quantity of product \( j \) to be produced
\( C_j \) be the profit contribution due to a product (paint) \( j \)
Let $a_{ij}$ be the quantity of material $i$ required for product $j$.

Let $m_{kj}$ be the market restriction specified for restriction $k$ and product $j$.

Let $mat_i$ be the material $i$ available daily.

Let $market_k$ be the market restriction constrained for $k$.

Let $n$ be the number product.

Let $n_1$ be the number of resources (material) available.

Let $n_2$ be the number of market restrictions.

Maximise

$$z = \sum_{j=1}^{n} c_j x_j$$

Subjected to

$$\sum_{j=1}^{n} a_{ij} x_j \leq mat_i \quad \forall i$$

$$\sum_{k=1}^{n_2} m_{kj} x_j \leq market_k \quad \forall k$$

**LINGO Modelling Language**

**LINGO Model**

**Model:**

**Title** Reddy Mikks Problem;

**SETS:**

product/1..2/:C,x;

resource/1..2/:mat;

restriction/1..2/:market;

material(resource,product):a;

mar(restriction,product):m;

**endsets**

[objective] max=@sum(product(j):C(j)*x(j));

@for(resource(i): [Material_constraints]
   @sum(product(j):a(i,j)*x(j))<=mat(i));

@for(restriction(i): [Marketing_constraints]
   @sum(product(j):m(i,j)*x(j))<=market(i));

**Data:**
LINGO model contains optional sections such as SETS and DATA

Sets are simply groups of related objects.

A set might be a list of products, trucks, or employees.

Each member in the set may have one or more characteristics associated with it called attributes

Attribute values can be known in advance or unknowns that LINGO solves for.

For example, each product in a set of products might have a price attribute; each truck in a set of trucks might have a hauling capacity attribute; and each employee in a set of employees might have a salary attribute, as well as a birth date attribute.

LINGO recognizes two kinds of sets: primitive and derived.

**Primitive Sets**

A primitive set is a set composed only of objects that can’t be further reduced.

In the Reddy Mikks example, the set PRODUCT, which is composed of two products, is a primitive set.

To define a primitive set in a sets section, you specify:

```
setname [/ member_list /] [: attribute_list];
```

The setname is a standard LINGO name you choose to name the set.

**Explicit Representation**

Enter a unique name for each member, optionally separated by commas.

Example: WAREHOUSES / WH1 WH2 WH3 WH4 WH5 WH6/: CAPACITY;
Implicit Representation

Do not have to list a name for each set member.

Syntax: setname/member1..memberN/[:attribute_list];

Example: WAREHOUSES/WH1..WH6/:CAPACITY;

<table>
<thead>
<tr>
<th>Implicit Member List Format</th>
<th>Example</th>
<th>Set Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..n</td>
<td>1..5</td>
<td>1, 2, 3, 4, 5</td>
</tr>
<tr>
<td>stringM..stringN</td>
<td>TRUCKS3..TRUCKS204</td>
<td>TRUCK3, TRUCK4, ..., TRUCK204</td>
</tr>
<tr>
<td>dayM..dayN</td>
<td>MON..FRI</td>
<td>MON, TUE, WED, THU, FRI</td>
</tr>
<tr>
<td>monthM..monthN</td>
<td>OCT..JAN</td>
<td>OCT, NOV, DEC, JAN</td>
</tr>
</tbody>
</table>

Derived Sets

A derived set is defined using one or more other sets. In other words, a derived set derives its members from other preexisting sets.

The set MATERIAL is a derived set. It derives its members from the unique pairs of members of the RESOURCE and PRODUCT sets.

To define a derived set, you specify:

```
¨ the name of the set,
¨ its parent sets,
¨ optionally, its members, and
¨ optionally, any attributes the set members have.
```

Syntax: setname(parent_set_list) [:member_list] [:attribute_list];

If the member_list is omitted, the derived set will consist of all combinations of the members from the parent sets called dense set.

When a set includes a member_list that limits it to being a subset of its dense form, we say the set is sparse.

A derived set's member_list may be constructed using either:

```
¨ an explicit member list, or,
¨ a membership filter.
```

Explicit Member List

SETS:

PRODUCT / A B/;
MACHINE / M N/;
WEEK / 1..2/;
ALLOWED (PRODUCT, MACHINE, WEEK);

ENDSETS

ALLOWED Set Membership:

<table>
<thead>
<tr>
<th>Index</th>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(A,M,1)</td>
</tr>
<tr>
<td>2</td>
<td>(A,M,2)</td>
</tr>
<tr>
<td>3</td>
<td>(A,N,1)</td>
</tr>
<tr>
<td>4</td>
<td>(A,N,2)</td>
</tr>
<tr>
<td>5</td>
<td>(B,M,1)</td>
</tr>
<tr>
<td>6</td>
<td>(B,M,2)</td>
</tr>
<tr>
<td>7</td>
<td>(B,N,1)</td>
</tr>
<tr>
<td>8</td>
<td>(B,N,2)</td>
</tr>
</tbody>
</table>

An example of explicit list:

```
ALLOWED (PRODUCT, MACHINE, WEEK) / A M 1, A N 2, B N 1/;
```

**Membership Filter**

Suppose you have already defined a set called TRUCKS, and each truck has an attribute called CAPACITY. You would like to derive a subset from TRUCKS that contains only those trucks capable of hauling big loads. You could use an explicit member list, and explicitly enter each of the trucks that can carry heavy loads.

However, why do all that work when you can use a membership filter as follows:

```
HEAVY_DUTY( TRUCKS) | CAPACITY( &1) #GT# 50000;:
```

We have named the set HEAVY_DUTY and have derived it from the parent set, TRUCKS.

The vertical bar character (|) is used to mark the beginning of a membership filter.

The membership filter allows only those trucks that have a hauling capacity (CAPACITY( &1)) greater than (#GT#) 50,000 into the HEAVY_DUTY set. The &1 symbol in the filter is known as a set index placeholder.

When building a derived set that uses a membership filter, LINGO generates all the combinations of parent set members.

Each combination is then "plugged" into the membership condition to see if it passes the test. The first primitive parent set's member is plugged into &1, the second into &2, and so on.

In this example, we have only one parent set (TRUCKS), so &2 would not have made sense.

The symbol #GT# is a logical operator and means "greater than."

**The Sets Section of a Model**

Sets are defined in an optional section of a LINGO model, called the sets section.
Before you use sets in a LINGO model, you have to define them in the sets section of the model.

The sets section begins with the keyword SETS: (including the colon), and ends with the keyword ENDSETS.

A model may have no sets section; a single sets section, or multiple sets sections.

A sets section may appear anywhere in a model.

The only restriction is you must define a set and its attributes before they are referenced in the model's constraints.

The Data Section

This section is used to assign values to some set attributes.

The data section allows you to isolate data from the rest of your model.

This is a useful practice in that it leads to easier model maintenance and facilitates rescaling a model’s size.

Similar to the sets section, the data section begins with the keyword DATA: (including the colon) and ends with the keyword ENDDATA.

In the data section, you can have statements to initialize the attributes of the sets you defined in a sets section.

syntax: attribute_list = value_list;

The attribute_list contains the names of the attributes you want to initialize, optionally separated by commas.

An example:

```
SETS:
    SET1 /A, B, C/: X, Y;
ENDSETS

DATA:
    X = 1 2 3;
    Y = 4 5 6;
ENDDATA
```

Set Looping Functions

The power of set based modelling comes from the ability to apply an operation to all members of a set using a single statement.

<table>
<thead>
<tr>
<th>Function</th>
<th>Function's Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>@FOR</td>
<td>The most powerful of the set looping functions, @FOR is used to generate constraints over members of a set.</td>
</tr>
<tr>
<td>@SUM</td>
<td>Probably the most frequently used set looping function, @SUM computes the sum of an expression over all members of a set.</td>
</tr>
</tbody>
</table>

The syntax for a set looping function is:
@function( setname [ ( set_index_list) ] : expression_list);

Example form the Rikky Mikks Problem

\[ \sum_{j=1}^{n} a_{ij} x_j \leq mat_i \quad \forall i \]

@for(resource(i): [Material_constraints] 
    @sum(product(j):a(i,j)*x(j))<=mat(i));

### Using Variable Domain Functions

Unless specified otherwise, variables in a LINGO model default to being non-negative and continuous.

LINGO provides four variable domain functions, which allow you to override the default domain of a variable.

- **@GIN** restricts a variable to being an integer value,
- **@BIN** makes a variable binary (i.e., 0 or 1),
- **@FREE** allows a variable to assume any real value, positive or negative, and
- **@BND** limits a variable to fall within a finite range.

The syntax for all variable domain function is:

```
Variable domain function(variable_name);
```

**Example 1:** @GIN( X); makes the scalar variable X general integer,

**Example 2:** @GIN( PRODUCE( 5)); makes the variable PRODUCE( 5) general integer,

**Example 3:** @FOR( DAYS( I): @GIN( START( I))); makes all the variables of the START attribute general integer.

### Fixed Charge problem

Consider a production planning problem with \( N \) products such that the \( j \)th product requires a fixed production or setup cost \( K_j \), independent of the amount produced, and a variable cost \( C_j \) per unit, proportional to the quantity produced. Assume that every unit of product \( j \) requires \( a_{ij} \) units of resource \( i \) and there are \( M \) resources. Given that product \( j \), whose sales potential is \( D_j \), sells for Rs \( p_j \) per unit and no more than \( b_i \) units of resource \( i \) are available (\( i = 1, 2, \ldots, M \)), the problem is to determine the optimal product mix that maximises the net profit.

### Solution

The total cost of production (fixed plus variable) is a nonlinear function of the quantity produced.

But, with the help of binary (0,1) integer variables, the problem can be formulated as an integer linear program.
Let the binary integer variable $y_j$ denotes the decision to produce or not to produce product $j$.

$$y_j = \begin{cases} 
1 & \text{If product } j \text{ is produced} \\
0 & \text{Otherwise}
\end{cases}$$

Let $x_j$ denotes the quantity of product $j$ produced

**The model**

Maximise

$$z = \sum_{j=1}^{N} p_j x_j - \sum_{j=1}^{N} (K_j y_j + C_j x_j)$$

The supply constraints for the $i$th resource is given by

$$\sum_{j=1}^{N} a_{ij} x_j \leq b_i \quad \forall \ i$$

The demand constraint for the $j$th product is given by

$$x_j \leq D_j y_j \quad \forall \ i$$

$$x_j \geq 0 \quad \text{and} \quad y_j = 0 \text{ or } 1 \quad \forall \ j$$

Note:- $x_j$ can be positive only when $y_j = 1$

**LINGO Model**

Model:

Title Fixed charge problem;

SETS:

product/1..3/:k,c,d,x,y,p;

resource/1..2/:b;
aij(resource,product):a;

ENDSETS

DATA:

K=1000,1500,1200;
c=400,520,500;
d=1000,2000,1000;
p=500,650,580;
b=3500,4700;
a=3 4 2
4 5 3;

ENDDATA

[objective]max=@sum(product(j):p(j)*x(j))
-@sum(product(j):(k(j)*y(j)+c(j)*x(j))));

@for(resource(i):[Resorce_Constraint])
@sum(product(j):(a(i,j)*x(j)))<=b(i));
Diet problem
Consider a diet problem in which a college student is interested in finding a minimum cost diet that provides at least 21 units of Vitamin A and 12 units of vitamin B from five foods of the following properties:

<table>
<thead>
<tr>
<th>Food</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vitamin A content</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Vitamin B content</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cost per unit</td>
<td>20</td>
<td>20</td>
<td>31</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Formulate the problem and solve using LINGO mathematical modelling language.

Another research suggests that the level of vitamin required is at least 20 units Vitamin A, 14 units Vitamin B and 25 units of Vitamin C. The above products also contain Vitamin C and it is given below. Modify the data section of the program to evaluate this.

<table>
<thead>
<tr>
<th>Food</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vitamin C content</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Lingo model (for the initial part of the problem)

Model:
Title Diet Problem;
! Diet problem given in the note;
Data:
No_of_vitamin = 2;
enddata
sets:
food/1..5/:x,cost;
vitamin/1..No_of_vitamin/:q;
Food_vitamin_content(vitamin,food):c;
endsets
data:
cost=20 20 31 11 12;
q=21 12;
c=1 0 1 1 2
  0 1 2 1 1;
enddata
min=@sum(food(i):cost(i)*x(i));
@for(vitamin(j):@sum(food(i):c(j,i)*x(i))>=q(j));
end
Lingo model (Considering the latest part of the problem)

Model:
Title Diet Problem;
! Diet problem given in the note;

Data:
No_of_vitamin = 3;

sets:
food/1..5/:x,cost;
vitamin/1..No_of_vitamin/:q;
Food_vitamin_content(vitamin,food):c;

endsets

data:
cost=20 20 31 11 12;
q=20 14 25;
c=1 0 1 1 2
   0 1 2 1 1
   2 2 1 2 1;

enddata

min=@sum(food(i):cost(i)*x(i));
@for(vitamin(j):@sum(food(i):c(j,i)*x(i))>=q(j));

end