LINGO 8.0 TUTORIAL

Created by:
Kris Thornburg
Anne Hummel

Table of Contents

Introduction to LINGO 8.0

LINGO is a software tool designed to efficiently build and solve linear, nonlinear, and integer optimization models.

LINGO 8.0 includes several new features, including:

- A new global solver to confirm that the solution found is the global optimum,
- Multistart capability to solve problems more quickly,
- Quadratic recognition and solver to identify quadratic programming (QP) problems,
- A faster and more robust Dual Simplex solver,
- An improved integer solver to enhance performance in solving many types of problems,
- Linearization capability to transform common nonsmooth functions to a series of linear functions,
- Infeasible and unbounded analytical tools to help identify model definition problems,
- A decomposition feature to identify if a model contains independent submodels,
- A threadsafe DLL for various classes of models, and
- More fun than ever before!

Creating a LINGO Model

An optimization model consists of three parts:

- Objective function – This is single formula that describes exactly what the model should optimize. A general manufacturing example of an objective function would be to minimize the cycle time for a given product.
- Variables – These are the quantities that can be changed to produce the optimal value of the objective function. For example, when driving a car, the duration of the trip (*t*) and the speed at which it is taken (*v*) determine the distance (*d*) that can be traveled.
- Constraints – These are formulas that define the limits on the values of the variables. If an ice cream store is determining how many flavors it should offer, only a positive number of flavors is feasible. This constraint could be expressed as

```
Flavors >= 0;
```

A sample model for cookie production by two bakers at a bakery is given by:

```
! A cookie store can produce drop cookies and decorated cookies, which sell
for $1 and $1.50 apiece, respectively.  The two bakers each work 8 hours per
day and can produce up to 400 drop cookies and 200 decorated cookies.  It
takes 1 minute to produce each drop cookie and 3 minutes to produce each
decorated cookie.  What combination of cookies produced will maximize the
baker's profit? ;

MAX = 1*Drop + 1.5*Deco;

Drop <= 400;
Deco <= 200;

1/60*Drop + 3/60*Deco <=16;
```

Several other items must be noted about this model:

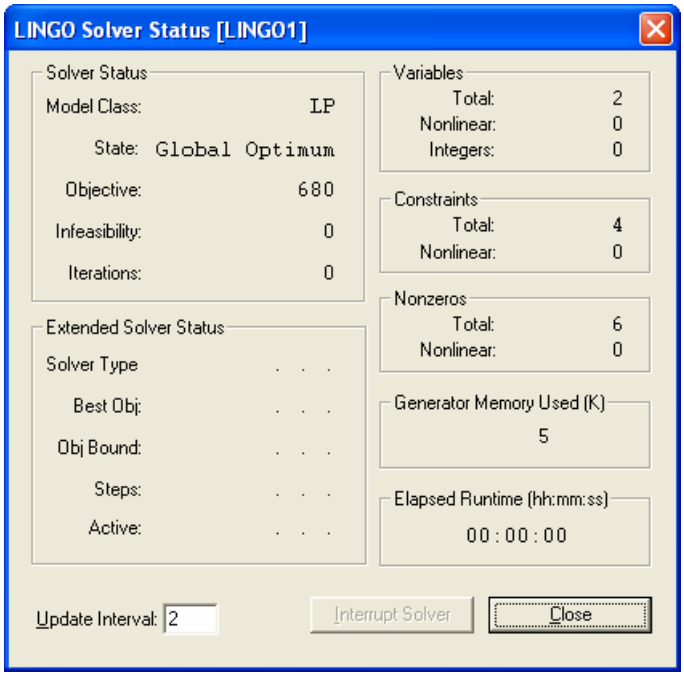
- Comments in the model are initiated with an exclamation point (!) and appear in green text.
- LINGO specified operators and functions appear in blue text.
- All other text is shown in black.
- Each LINGO statement must end in a semi-colon (;).
- Variable names are not case-sensitive and must begin with a letter (A-Z). Other characters in the variable name may be letters, numbers (0-9), or the underscore character (_). Variable names can be up to 32 characters in length.

Solving a LINGO Model

Once the LINGO model has been entered into the LINGO Model window, the model can be solved by clicking the *Solve* button on the toolbar, by selecting LINGO | Solve from the menus, or by using the ctrl + s keyboard shortcut.

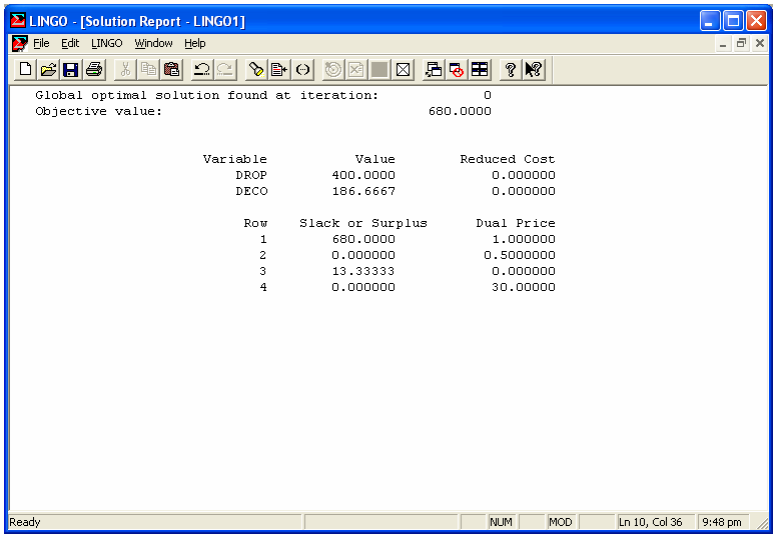
LINGO will notify you of any errors it has encountered. The best way to get information about these errors is to consult the *Error Messages* section in the software's proprietary tutorial.

If no errors are found, then the LINGO Solver Status window appears.



This window provides information on the number of nonlinear, integer, and total variables in the model; the nonlinear and total number of constraints used in the model; and the number of nonlinear and total nonzero variable coefficients used. The Solver Status box in this window details the model classification (LP, QP, ILP, IQP, NLP, etc.), state of the current solution (local or global optimum, feasible or infeasible, etc.), the value of the objective function, the infeasibility of the model (amount constraints are violated by), and the number of iterations required to solve the model. The Extended Solver Status box details similar information for the more advanced branch-and-bound, global, and multistart solvers.

By closing this window, you can then view the Solution Report window.



This window shows the values of each variable that will produce the optimal value of the objective function.

The reduced cost for any variable that is included in the optimal solution is always zero. For variables not included in the optimal solution, the reduced cost shows how much the value of the objective function would decrease (for a MAX problem) or increase (for a MIN problem) if one unit of that variable were to be included in the solution. For example, if the reduced cost of a certain variable was 5, then the optimal value of the MAX problem would decrease by 5 units if 1 unit of the variable were to be added.

The Slack or Surplus column in the Solution Report shows how tight the constraint is. If a constraint is completely satisfied as an equality, then slack/surplus is zero. If slack/surplus is positive, then this tells you how many more units of the variable could be added to the optimal solution before the constraint becomes an equality. If slack/surplus is negative, then the constraint has been violated.

The Dual Price column describes the amount to which the value of the objective function would improve if the constraining value is increased by one unit.

Using Sets in LINGO

LINGO allows you to group many instances of the same variable into sets. For example, if a model involved 27 delivery trucks, then these 27 trucks could be described more simply as a single set. Sets may also include attributes for each member, such as the hauling capacity for each delivery truck.

Sets may be either primitive or derived. A primitive set is one that contains distinct members. A derived set, however, contains other sets as its members.

To use sets in a model, the special section called the *SETS* section must be defined before any of the set members are used in the model's constraints. This section begins with the tag `SETS:` and ends with the tag `ENDSETS`.

A primitive set could be defined as follows:

```
SETS:
  Trucks/TR1..TR27/:Capacity;
ENDSETS
```

This set is given the setname "Trucks" and contains 27 members, identified by TR1 – TR27. The attributes for each member are called "Capacity."

The derived set is defined similarly, but must also include the parent set list. An example of a derived set could be:

```
SETS:
  Product/X Y/;
  Machine/L M/;
  Make(Product Machine)/X L, X M,
    Y M/;
ENDSETS
```

This set declaration defines two primitive sets, Product and Machine, and one derived set called Make. The Make set is derived from the parent sets Product and Machine. Members are specified as shown. Notice that a fourth Product-Machine combination, Y L, could be theoretically possible. This example does not allow for such a combination. If all combinations of the parent sets are possible, then no member set need be defined. An attribute list for the derived set can also be included in the same way as for a primitive set.

Several set looping functions are also available for use in LINGO. These functions are as follows:

- @FOR – generates constraints over members of a set.
- @SUM – sums an expression over all members of the set.
- @MIN – computes the minimum of an expression over all members of the set.
- @MAX – computes the maximum of an expression over all members of the set.

Each of the above looping functions has a similar form of syntax and the looping functions can even be nested. Examples of expressions using each type of looping function are as follows:

- This @FOR statement sets the hauling capacity for all 27 delivery trucks in the Trucks set to at most 3000 pounds:

```
@FOR(Trucks(T): Capacity(T)<=3000);
```

- This @SUM statement calculates the total hauling capacity from the individual trucks:

```
TOTAL_HAUL=@SUM(Trucks(J): Capacity(J));
```

- These @MIN and @MAX statements find the extreme hauling capacity levels from the individual delivery trucks:

```
MIN_HAUL = @MIN(Trucks(J): Capacity(J));
MAX_HAUL = @MAX(Trucks(J): Capacity(J));
```

The LINGO Data Section

LINGO provides a separate section called the *DATA* section in which values can be defined for different variables. Set members can be initialized in this section, attributes of the sets can be defined, or scalar variable parameters can be assigned values as well.

The *DATA* section is defined after the *SETS* section is defined in the model. The section begins with the tag `DATA:` and ends with the tag `ENDDATA`. Statements within the *DATA* section follow the syntax: `object_list = value_list;`

The object list contains the names of the attributes or of the set whose values are being initialized. The value list assigns the values to the specified members of the object list.

The following examples show two ways to use the *DATA* section in LINGO. In each example, the X and Y attributes of SET1 are being initialized to [1, 2, 3] and [4, 5, 6], respectively. The first example defines values for each attribute separately:

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

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

The next example shows how one statement can be used to assign values to the two attributes simultaneously. Each row assigns different values to the X, Y pair:

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

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

When parameters or attributes are defined in the *DATA* section of a model, a feature called *What-if Analysis* can be used to examine the effects of varying the value of the parameter or attribute. For example, if the inflation rate is most likely going to fall between 2% and 6%, the parameter can be defined as follows in the *DATA* section:

```
DATA:
   INFLATION_RATE = ?;
ENDDATA
```

When LINGO encounters the ? in the *DATA* section, it will prompt the user to enter a value for the parameter. The user can then enter values between 2% and 6%, and LINGO will solve the model using that "what-if" value.

All the elements of an attribute can be initialized to a single value using the *DATA* section as well. The following example shows how to assign the value of 20 to all seven members of the NEEDS attribute and 100 to all seven members of the COST attribute:

```
SETS:
   DAYS / MO, TU, WE, TH, FR, SA,
      SU/: NEEDS, COST;
ENDSETS

DATA:
   NEEDS, COST = 20, 100;
ENDDATA
```

Data values can also be omitted from the *DATA* section of a LINGO model to indicate that LINGO is free to determine the values of those attributes itself. The following example shows that the first two values of the attribute CAPACITY have been initialized to 34, but the last three variables have not been defined:

```
SETS:
   YEARS /1..5/: CAPACITY;
ENDSETS

DATA:
   CAPACITY = 34, 34, , , ;
ENDDATA
```

Variable Types in LINGO

All variables in a LINGO model are considered to be non-negative and continuous unless otherwise specified. LINGO's four variable domain functions can be used to override the default domain for given variables. These variable domain functions are:

- @GIN – any positive integer value
- @BIN – a binary value (ie, 0 or 1)
- @FREE – any positive or negative real value
- @BND – any value within the specified bounds

Similar syntax is used for the @GIN, @BIN, and @FREE variable domain functions. The general form for the declaration of a variable *x* using any of these functions is
`@FUNCTION(X);`

The @BND function has a slightly modified syntax, which includes the upper and lower bounds for the acceptable variable values. The general form for the declaration of a variable *x* between a lower bound and an upper bound is given by
`@BND(lowerBound, X, upperBound);`

Navigating the LINGO Interface

Operations in LINGO can be carried out using commands from the menus, toolbars, or shortcut keys.

There are five menus in the main LINGO window. They are the File, Edit, LINGO, Window, and Help menus.

The following list details the commands in the File menu. Shortcut keys are included in parentheses when available:

New (F2)	Open a new model window
Open (F3)	Open a saved file
Save (F4)	Save a current model
Save As (F5)	Save a current model to a new filename
Close (F6)	Close the current model
Print (F7)	Prints the current window's content
Print Setup (F8)	Configures printer preferences
Print Preview (Shift+F8)	Displays the window content as it would look if printed
Log Output (F9)	Opens a log file to log output to the command window
Take Commands (F11)	Runs a command script contained in a file
Import LINDO File (F12)	Converts a LINDO file into a LINGO model
Export File	Exports a model in MPS or MPI file format
License	Prompts user for new license password to upgrade system
Database User Info	Prompts for a user id and password for database access via the @ODBC() function
Exit (F10)	Closes LINGO

The Edit menu contains the following commands:

Undo (ctrl+z)	Undoes the last action
Redo (ctrl+y)	Redoes the last undo command
Cut (ctrl+x)	Copies and deletes highlighted text
Copy (ctrl+c)	Copies highlighted text to the clipboard
Paste (ctrl+v)	Pastes the clipboard's contents into the document
Paste Special	Pastes the clipboard's content into the document, in a user-specified manner
Select All (ctrl+a)	Selects all of the contents of the current window
Find (ctrl+f)	Searches the document for a specific text string
Find Next (ctrl+n)	Searches the document for the next occurrence of a specific text string
Replace (ctrl+h)	Replaces a specific text string with a new string
Go To Line (ctrl+t)	Moves the cursor to a certain line number
Match Parenthesis (ctrl+p)	Finds the selected parenthesis's mate

Paste Function	Pastes a syntax template for a specific LINGO @function
Select Font (ctrl+j)	Configures the font for a selected portion of text
Insert New Object	Puts an OLE object in the document
Links	Creates links to external objects
Object Properties	Defines the properties of an embedded object

The LINGO menu contains the following commands:

Solve (ctrl+s)	Solves the model
Solution (ctrl+o)	Makes a solution report window for the current model
Range (ctrl+r)	Creates a range analysis report for the current window
Options (ctrl+i)	Sets system options
Generate	Generates the algebraic form of the model
Picture (ctrl+k)	Displays a graphic of the model's matrix
Debug (ctrl+d)	Identifies errors in infeasible and unbounded models
Model Statistics (ctrl+e)	Reports the technical details the model
Look (ctrl+l)	Creates a formulation report for the current window

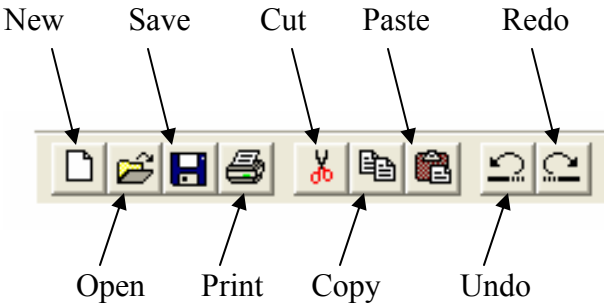
The Window menu contains the following commands:

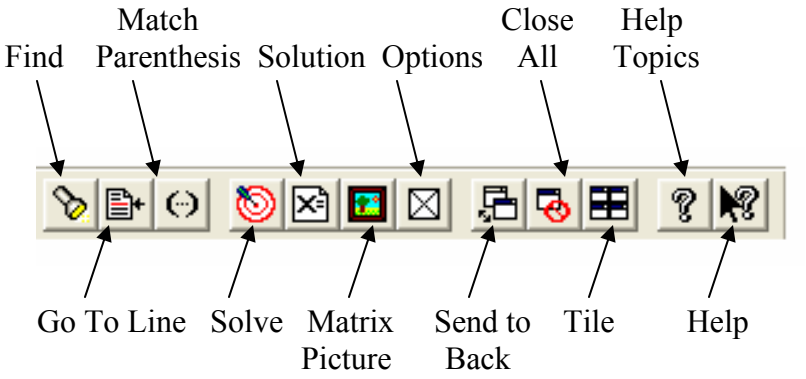
Command Window (ctrl+1)	Opens a window for command-line operation of LINGO
Status Window (ctrl+2)	Opens the solver's status window
Send to Back (ctrl+b)	Places the current window behind all other open windows
Close All (ctrl+3)	Closes all open windows
Tile (ctrl+4)	Places open windows in a tiled arrangement
Cascade (ctrl+5)	Places all open windows in a cascaded arrangement
Arrange Icons (ctrl+6)	Aligns icon windows at the bottom of the main LINGO window

The Help window contains the following commands:

Help Topics	Opens LINGO's manual
Register	Registers your version of LINGO online
AutoUpdate	Provides prompts to download software updates
About LINGO	Displays software information

A single toolbar located at the top of the main LINGO window contains many of the same commands as listed above. These commands can be accessed simply by using the mouse to click on the icon representing them. The following pictures detail which icons correspond to which commands.





LINGO Operators and Functions

LINGO provides a vast array of operators and functions, making it a useful problem-solving tool. A selection of the primary operators and functions is given below.

There are three types of operators that LINGO uses: arithmetic, logical, and relational operators. The arithmetic operators are as follows:

- Exponentiation: ^
- Multiplication: *
- Division: /
- Addition: +
- Subtraction: -

The logical operators are used in set looping functions to define true/false conditions:

- #LT#: TRUE if the left argument is strictly less than the right argument, else FALSE
- #LE#: TRUE if the left argument is less-than-or-equal-to the right argument, else FALSE
- #GT#: TRUE if the left argument is strictly greater than the right argument, else FALSE
- #GE#: TRUE if the left argument is greater-than-or-equal-to the right argument, else FALSE
- #EQ#: TRUE if both arguments are equal, else FALSE
- #NE#: TRUE if both arguments are not equal, else FALSE
- #AND#: TRUE only if both arguments are TRUE, else FALSE
- #OR#: FALSE only if both arguments are FALSE, else TRUE
- #NOT#: TRUE if the argument immediately to the right is FALSE, else FALSE

The relational operators are used when defining the constraints for a model. They are as follows:

- The expression is equal: =
- The left side of the expression is less than or equal to the right side: <=
- The left side of the expression is greater than or equal to the right side: >=

The following list contains a sampling of mathematical functions that can be used in LINGO:

- @ABS(X) – returns the absolute value of *X*
- @SIGN(X) – returns -1 if *X* is negative and +1 if *X* is positive
- @EXP(X) – calculates e^X
- @LOG(X) – calculates the natural log of *X*
- @SIN(X) – returns the sine of *X*, where *X* is the angle in radians
- @COS(X) – returns the cosine of *X*
- @TAN(X) – returns the tangent of *X*

LINGO also contains a plethora of financial, probability, and import/export functions. These are commonly used in more advanced models, which are beyond the intended scope of this tutorial.

Common LINGO Error Messages

LINGO provides a variety of error messages useful for debugging a developing model. The most common errors include the following:

- 7: Unable to open file: *filename*
  - Retype filename correctly
- 11: Invalid input: A syntax error has occurred
  - Check the line LINGO suggests for missing semi-colons, etc.
- 12: Unmatched parenthesis
  - Close the parenthesis set
- 15: No relational operator found
  - Make sure all constraints contain =, <=, >=
- 44: Unterminated condition
  - Put a colon at the end of each conditional statement in a set operator
- 50: Improper use of the @FOR() function
  - @FOR() functions cannot be nested inside other set operators
- 68: Multiple objective functions in model
  - Only one is allowed, please
- 71: Improper use of a variable domain function (eg, @GIN, @BIN, @FREE, @BND)
  - Check the syntax
- 81: No feasible solution found
  - Check model's consistency and constraints
- 82: Unbounded solution
  - Add constraints
- 102: Unrecognized variable name: *variable name*
  - Check spelling
- 108: The model's dimensions exceed the capacity of this version
  - Upgrade to full version or use Excel
- 164: Invalid LINGO name
  - Create a name to conform to LINGO's naming conventions

LINGO Programming Examples

A common programming model is the Knapsack problem, which deals with maximizing the utility of loading capacity. This example shows how to properly set up a knapsack problem.

```
SETS:
  ITEMS / ANT_REPEL, BEER, BLANKET,
   BRATWURST, BROWNIES, FRISBEE,
    SALAD, WATERMELON/:
     INCLUDE, WEIGHT, RATING;
ENDSETS

DATA:
  WEIGHT RATING =
    1      2
    3      9
    4      3
    3      8
    3     10
    1      6
    5      4
   10     10;
  KNAPSACK_CAPACITY = 15;
ENDDATA

MAX = @SUM( ITEMS: RATING * INCLUDE);

@SUM( ITEMS: WEIGHT * INCLUDE) <=
 KNAPSACK_CAPACITY;

@FOR( ITEMS: @BIN( INCLUDE));
```

Another common programming model is the transportation problem. Transportation problems deal with transporting goods from one location to another at minimal cost. This example shows how to model a simple transportation problem.

```
MODEL:
! A 6 Warehouse 8 Vendor Transportation Problem;

SETS:
  WAREHOUSES / WH1 WH2 WH3 WH4 WH5 WH6/: CAPACITY;
  VENDORS / V1 V2 V3 V4 V5 V6 V7 V8/   : DEMAND;
  LINKS( WAREHOUSES, VENDORS): COST, VOLUME;
ENDSETS

! The objective;
  MIN = @SUM( LINKS( I, J):
   COST( I, J) * VOLUME( I, J));

! The demand constraints;
  @FOR( VENDORS( J):
   @SUM( WAREHOUSES( I): VOLUME( I, J)) =
    DEMAND( J));

! The capacity constraints;
  @FOR( WAREHOUSES( I):
   @SUM( VENDORS( J): VOLUME( I, J)) <=
    CAPACITY( I));

! Here is the data;
DATA:
  CAPACITY = 60 55 51 43 41 52;
  DEMAND = 35 37 22 32 41 32 43 38;
  COST = 6 2 6 7 4 2 5 9
       4 9 5 3 8 5 8 2
       5 2 1 9 7 4 3 3
       7 6 7 3 9 2 7 1
       2 3 9 5 7 2 6 5
       5 5 2 2 8 1 4 3;
ENDDATA
END
```