Writing problem files

The problem file is a text file that states the problem in a format similar to the way you would write it down on paper. Any text editor or word processor can be used to create it: the one requirement is that it produce ordinary text files. Many word processors produce files containing special codes for formatting purposes. LINEAR would produce error messages on encountering such characters. However, most word processors have a "non-document" mode or some other way of producing a file without those special characters.

LINEAR itself does not contain an editor. Normally you would produce the problem file with your text editor before starting LINEAR. When LINEAR is running, unless you have a resident "notepad" editor such as SideKick, you must temporarily leave the program to produce or alter a problem file. The procedure of leaving LINEAR, editing a file, and returning to LINEAR to load the file can be made more convenient using the '**E**dit' command if LINEAR is run from a suitable batch file (see the section **Batch files and editing** in chapter 4).

It is also possible to enter the problem file directly from the keyboard. This is not recommended except for very small problems because of the likelihood of errors (once a line is entered it can not be called back for editing).

Here is the sample problem file (EXAMPLE1.PRB on the distribution disk) that was used in the Introduction:

```
constraints 2;
variables 3 x1 x2 x3;
maximize
20 x1 + 30 x2 + 15 x3 [ profit ];
1.5 x1 + x2 + x3 <= 100 [ m1 ];
x1 + 2 x2 + x3 <= 100 [ m2 ];
end
```

There are several additional items that can be included in a problem file, but this one contains all that is necessary:

(1) the **constraints declaration**: the word 'constraints', followed by the number of constraints and a semicolon. You may abbreviate 'constraints' as 'con' or 'cons'.

(2) the **variables declaration**: the word 'variables' followed by the number of variables, a list of all the variables used, and a semicolon. You may abbreviate 'variables' as 'var' or 'vars'. Variable names must start with a letter and may include letters, digits and the underscore '_'. They may be any length, but only the first 11 characters are recognized. Do not use any of the following names:

```
ABS    ARCTAN    COS     EXP    LN      MAX
MIN    RANDOM    ROUND   SIN    SQRT
```

Avoid names consisting of 'C_' and a number, which LINEAR uses as constraint names.

(3) the **objective**: 'maximize' or 'minimize' (abbreviated 'max' or 'min'), the expression to be maximized or minimized, and a semicolon.

(4) the constraints, separated by semicolons.

(5) the word 'end'.

In all of this, you may use upper and lower case letters interchangeably. LINEAR never distinguishes between the two.

It might sometimes seem annoying to have to count the constraints and count and list the variables at the beginning of the file, but this is actually a valuable feature for detecting errors. One of the most common errors is misspelling a variable name; having the variables listed at the beginning allows LINEAR to catch these errors immediately on loading the file (unless the misspelling happens to match another variable).

The objective and the constraints are written in ordinary mathematical style, and may run over several lines. The constraints may involve '`<=`', '`>=`' or '`=`'. You may use '`<`' and '`>`' as abbreviations of '`<=`' and '`>=`' respectively. The terms may appear in any order, and on either side of the inequality or equality: thus the first constraint might have been written as

```
100 - x2 >= 1.5 x1 + x3;
```

or even

```
200-.5X1-X2>=100+X3+X1;
```

Spacing is almost completely optional. There is one situation where a space is necessary: between a number and a name starting with the letter '`e`'. In that case, without a space LINEAR would assume that "scientific notation" is being used, e.g. '`2E3`' means 2×10^3 or 2000, while '`2 E3`' is 2 times the variable '`E3`'.

Constraints such as '`x1>=0`' need not be stated, as LINEAR always assumes that all variables must be ≥ 0.

The objective and constraints may be named, by placing the name in square brackets at the start or before the concluding semicolon. These names follow the same rules as variable names. They must all be distinct (and remember that only the first 11 characters count). If no name is specified, the objective is called '`OBJECTIVE`' and the constraints are called '`C_1`', '`C_2`', etc.

Errors

It is very easy to make a mistake. I have tried to make LINEAR in such a way that it is easy to identify errors and take corrective action.

When LINEAR detects any error, it will display an error message and a menu:

```
Continue                    Explain
```

`E``xplain` will produce a more detailed description of the error and any actions that '**`C`**`ontinue`' would take in trying to correct it, and return to this menu.

`C``ontinue` may (depending on the error) simply return to the main menu, or else keep going, perhaps taking some corrective action, as noted by '**`E`**`xplain`'.

— You will sometimes want to use the $\boxed{\text{Esc}}$ key to exit to the main menu instead of using '**`C`**`ontinue`'.

If the error is in a problem file being loaded, the error message will include the offending line of the file, with a pointer to the approximate place the error occurs, such as the following:

```
 Line  5
y + abs(2-z) < 3 [ c1 ]  ;
          ^
Error - Missing parenthesis
```

In this example, the error occurred in line 5 of the file: the variable '`z`' was inside parentheses, which is not allowed. As written, this is not a linear constraint and therefore unsuitable for LINEAR. It could be replaced, however, by the two linear constraints '`y + 2 - z < 3`' and '`y - 2 + z < 3`'.

In many cases, including this one, '**C**`ontinue`' will stop processing this constraint and skip to the next semicolon. The hope is to avoid having the one error spawn multiple error messages. The following appears on the screen:

```
Begin skip
y + abs(2-z) < 3 [ m1 ]  ;
                        ^ End skip
```

The '`Begin skip`' and '`End skip`' are to show you what was skipped over. While looking for the next semicolon, LINEAR completely ignores everything except illegal characters and end of file. This includes the constraint name, so LINEAR will provide this constraint with a name consisting of '`C_`' and a number. Then it resumes processing with the next constraint.

In most cases LINEAR will still end by saying that the file was loaded, but of course the tableau that was loaded is not likely to be correct. You could try to correct the tableau with the '**C**`hange`' commands (see chapter 5), but it is usually best to edit the problem file to correct the error.

Additional features

The following example shows several more features that can be used in problem files: echoed and non-echoed comments, user input, parameters, random numbers and arithmetic in coefficients.

```
" Sample problem #2 by R. Israel, 28/04/87
"
constraints 5;
parameters 4
 p = "Enter a value for p: ? " ?;
 q = "Enter a value for q: ? " ?;
 pi = 4 arctan(1) ;    ! this is 3.14159...
 r = round(random + p - 1/2) ;
!       r is 1 with probability p, 0 with probability 1-p
variables 3 x1 x2 x3 ;
maximize x1 + x2 - x3  ;
x1 + abs(1/p+q) x2 <= min(r,p,q) ;
x2 + p q x3 >= round(5 r) - 3^(-p) x1  ;
3.1 q x1 + p x2 + x3 = ln(p) ;
x1 - x3 =  sin(q r) x2  ;
x1 + x3 = 3.14159265358979 - pi  ;
 ! rhs should be 0
end.
```

An **echoed comment** is anything between quotation marks '`"  "`'. It will be sent to the screen, exactly as written, when the file is loaded. This includes the carriage return/line feed at the end of a line if the comment extends over more than one line. If I had written

```
" Sample problem #2 by R. Israel, 28/04/87"
```

all on one line, the next echoed comment

```
"Enter a value for p: ? "
```

would have appeared on the same line of the screen instead of the next one. Echoed comments are useful for verifying that the right version of the problem file is being loaded, as well as in prompts for user input.

User input can be accepted using '`?`'. Whenever this occurs in the problem file, LINEAR waits for a line to be entered, and uses this as if it were part of the problem file. This is most useful for obtaining values for adjustable parameters, although in principle any part of a problem file can be entered in this way. To help the user out, an echoed comment such as `"Enter a value for p:  ?"` should be used as a prompt. Note that the cursor moves to the next line when the user presses $\boxed{\hookleftarrow}$ at the end of the line, so that in the example `"Enter a value for p:  ?"` and `"Enter a value for q:  ?"` will appear on different lines.

A **non-echoed comment** is anything that follows an exclamation '`!`' until the end of a line, such as '`!  this is 3.14159...`' in the example. These are simply ignored by the program. Use echoed comments for documentation in the problem file (e.g. explaining the meaning of parameters, constraints and variables).

Often you will want to solve the same basic problem several times, with changes in a few parameters. Each of these parameters may affect several coefficients in different ways, so that re-editing the problem file and making all the changes in coefficients could be a laborious process. To help in this, the **parameters declaration** and **arithmetic expressions in coefficients** are provided.

The (optional) **parameters declaration** comes between the constraints declaration and the variables declaration. It consists of the word '`parameters`' followed by the number of parameters and the list of parameters and values. There may be up to 115 parameters. Names of parameters follow the same rules as names of variables. You should not give a parameter the same name as a variable. The name of each parameter is followed by '`=`' and the parameter value, which may be given as a number or an expression. You may wish to use '`?`' to have the user enter the parameter value from the keyboard. The values given to the parameters in this declaration can then be referred to by the parameter names in the objective and constraints. Separate the value of a parameter from the name of the next parameter by '`,`' or '`;`', and end the list of parameters with '`;`'.

Note: these are what I call "data parameters". Their values are set at the time the problem is loaded. Two quite different types of parameters ("objective parameters" and "rhs parameters") are used in parametric programming with the '**`A`**`nalyze`' command, where the effects of changes in those parameters can be studied.

The word '`random`' can be used in place of a parameter. It produces a random number between 0 and 1 (with a uniform distribution). The random number generator is initialized using the system clock the first time '`random`' is used, so each time the problem file is loaded a different sequence of numbers will be produced.

An **expression** may involve numbers, any parameters already defined, the usual arithmetic operations (including '`^`' for powers), and the functions '`abs`', '`arctan`', '`cos`', '`sin`', '`exp`', '`ln`', '`sqrt`', '`round`', '`max`', and '`min`'. Multiplication may be indicated with either '`*`' or simply a blank space. However, be sure to include this space: don't write '`pq`' if you mean '`p q`', or LINEAR will interpret it as a single name. The arguments of functions must be within parentheses, e.g. '`sin(p)`', not '`sin p`'. '`max`' and '`min`' can take any number of arguments, separated by commas. Expressions are evaluated using the usual rules of algebra: powers have a higher priority than multiplication and division, which in turn have higher priority than addition and subtraction. Multiplication and division have equal priority, as do addition and subtraction. Thus

```
    a / b c + d e ^ f
```

is interpreted as

```
    ((a / b) * c) + (d * (e ^ f)).
```

(If in doubt about this, use parentheses).

In the objective function and constraints, the coefficients may be expressions instead of numbers. The rules for these expressions are the same as for expressions as values of parameters. Note that a coefficient must come before the variable it multiplies: thus if '`p`' is a parameter and '`x`' is a variable, '`p x`' is allowed but '`x p`' is not. Moreover, variables are not allowed within parentheses, e.g. '`p(x-y)`' must be replaced by '`p x - p y`' if '`x`' and '`y`' are variables.

Some advice on style

This section is about form, not content — for advice on modelling, see the section **Modelling** in chapter 2.

The features discussed in the last section are particularly important in any problem that will be used more than once, especially if it will be used by more than one person, or over an extended period of time. They allow the problem file to be self-documenting and easily modified. There is little to lose by including them, while omitting them can lead to a lot of effort being wasted.

As mentioned, an echoed comment is useful for identifying the problem. This will avoid the possibility of accidentally using the wrong problem file, especially when there are several files with similar names on the disk, or different versions with the same name on different disks. Use a consistent convention for identification. Include the date of the last modification.

Non-echoed comments may be used to indicate the significance of parameters, variables and constraints. What is obvious to you when you write the problem file may not be so obvious later when someone is trying to figure out what it means. To some extent, the names of the parameters, variables and constraints can help here, if they are well chosen. There is a tendency for students, in particular, to call the variables '`X1`', '`X2`', etc., because those names are used in teaching the theory and methods of linear programming. However, in practice it is much better to use descriptive or mnemonic names so that the significance of a variable is apparent at a glance. This is especially true in large problems. Typical names might look like '`P3U2M5`'. A non-echoed comment might explain

```
! PiUjMk: product i from unit j in month k
```

In a large problem, there will typically be several groups of similar constraints. To make the problem easier to read and understand, keep these groups together in the problem file, headed by a non-echoed comment describing the type of constraint, and leave blank space between the groups.

In formulating a problem, there is often some calculation necessary to set up the numbers involved. Parameters and arithmetic expressions allow LINEAR to do this work for you. The amount of effort saved may be minimal **if** the problem will only be run once. However, if the problem is used or modified at a later date, there can be considerable advantages:

— you may only need to change one or two parameter values instead of correcting many individual numbers.

— clarity is gained by showing how the numbers in the problem are obtained.

If the problem will be run a number of times with different values of a few parameters, it will be easier to use the '**?**' feature to ask the user for the value each time, rather than editing the problem each time. If the number of parameters to change is larger, editing the file will be more convenient. In this case it will be much better to have the changes all made in one place, the parameters declaration, rather than scattered through the file.

Consult the problem files on the distribution disk for some examples of the use of these features.

Another way to set up a problem is to use a spreadsheet program to produce a DIF file (see the next section). This can be especially useful if the problem data come from a database, or require more extensive calculations than LINEAR itself can conveniently provide.

DIF files

The ability to load and save DIF files provides a means of communicating between LINEAR and most popular spreadsheet packages. A typical case might involve

(1) setting up the problem in a spreadsheet,
(2) saving it from the spreadsheet in DIF format,
(3) loading it into LINEAR,
(4) solving it there,
(5) saving the resulting tableau in DIF format,
(6) loading it back into the spreadsheet, and
(7) using the results in the spreadsheet.

The DIF format was chosen because most spreadsheet packages can either work directly with this format, or translate files between this format and their own formats (e.g. with the TRANS utility of Lotus 1-2-3).

It is also possible to load into LINEAR a DIF file produced by LINEAR, i.e to use DIF files in the same way as TAB files. However, TAB files are to be preferred for such purposes: they retain complete accuracy (while the DIF file may involve some loss of accuracy), are usually smaller in size and quicker to use, and include parameters (which DIF files omit).

Below is an example of part of a Lotus 1-2-3 worksheet that can be made into a DIF file as input for LINEAR.

```
       A        B        C     D     E       F
1   MAX        X1       X2    X3
2   PROFIT     1        1    -1
3   CON1       1        3.5   0     <=     4.5
4   CON2       0.1      1     5.9   >=       7
5              9        2     1      =      12
6              8        2            =      10
7              1              1      =       2
```

(the 1 to 7 on the left and A to F on the top are 1-2-3's row and column labels).

The format must follow this one closely:

(1) At the top left of the worksheet is '`MAX`' or '`MIN`', indicating whether to maximize or minimize the objective.

(2) The rest of the top row consists of the names of the variables, labelling the columns. The two rightmost columns should be left unlabelled, corresponding to the symbols '`>`', '`=`' etc. and the RHS column.

(3) The rows of the worksheet below the top row correspond to the objective function and the constraints. The objective must come first. The labels for these rows are in the extreme left column (these may be left blank, in which case LINEAR will supply default names as it does for a PRB file).

(4) The main body of the worksheet contains the coefficients of the objective function and constraints. Zero entries may be left blank. otherwise each entry is a number (the actual worksheet cell may contain a formula, but LINEAR will only see the value). Note that for the objective function the entries are the actual coefficients, not (as in a tableau for a maximization problem) the coefficients with signs reversed.

(5) The second-last column must contain '`<=`', '`>=`' or '`=`' for each constraint. '`>=`' and '`<=`' may be abbreviated as '`<`' and '`>`'. There should be a blank in this column for the objective function.

(6) The last column must contain the right-hand-side entries for each constraint. There may be an entry here for the objective function, which would be a constant term in that function.

To make this into a DIF file, you must first (in Lotus 1-2-3) use ' `/FS`' (File Save) to save the worksheet to a file, then use the Lotus TRANS utility to translate this into a DIF file. This DIF file can be loaded by LINEAR's '**`L`**`oad `**`D`**`if`' command.

If the linear programming problem is only part of a larger 1-2-3 worksheet, you will have to use '`/FX`' (File eXtract) to select only that part of the worksheet to put in the file: LINEAR will not work with a DIF file that contains any entries (even blanks) outside of the linear programming problem. However, there is a slight complication: the Lotus TRANS utility will not work with a file produced by '`/FX`', only '`/FS`'. Therefore, you must do the following (starting in 1-2-3):

(1) use '`/FX`' to extract the linear programming problem from the worksheet.

(2) use '`/FR`' (File Retrieve) with the file created in step (1).

(3) use '`/FS`' (File Save) to save the file.

(4) exit 1-2-3 and use TRANS to translate the saved file to DIF format.

(5) start LINEAR and '**L**`oad`' the DIF file.

Some spreadsheet packages will give you a choice of saving a file "by rows" or "by columns". It does not matter which you choose: LINEAR can determine which method was used (by looking for the '`<`', '`>`' and '`=`' symbols), and load the file the correct way. In fact, the worksheet could be transposed (rows interchanged with columns) and the result would be the same as far as LINEAR is concerned.

The example produces the following tableau (as presented by the '**T**`ableau` **S**`how tableau`' command):

```
         X1      X2      X3      RHS
 PROFIT -1.0000 -1.0000  1.0000   0.0
   CON1  1.0000  3.5000  0.0      4.50000
   CON2 -0.1000 -1.0000 -5.9000  -7.00000
*   C_3  9.0000  2.0000  1.0000  12.00000
*   C_4  8.0000  2.0000  0.0     10.00000
*   C_5  1.0000  0.0     1.0000   2.00000
```

Using '**S**`olve`', we obtain the following tableau (as shown by '**T**`ableau` **S**`how tableau`'):

```
                    *        *
          CON1      C_5      C_4      RHS
 PROFIT     0.1538  -1.0000   0.2308  1.0000
   CON2     0.7538   5.9000  -0.8192  0.0
     X2     0.3077   0.0     -0.0385  1.0000
*   C_3     0.0     -1.0000  -1.0000  0.0
     X1    -0.0769   0.0      0.1346  1.0000
     X3     0.0769   1.0000  -0.1346  1.0000
```

Note, by the way, that although the artificial variable `C_3` is still basic, its value is 0, so we do have the optimal solution (in fact, constraint `C_3` is redundant, as it is the sum of `C_1` and `C_2`). The negative shadow price of `C_5` is also harmless, as this is an artificial variable.

We can now use '**S**`ave` **D**`if`' to save this tableau, and then TRANS to translate the DIF file to a 123 worksheet. TRANS will ask you whether the DIF file was saved "by rows" or "by columns": choose "by rows". The resulting 123 worksheet looks like this:

```
        A          B       C          D      E      F
1  MAX      CON1        *C_5   *C_4
2  PROFIT   -0.15385    1      -0.23077            1
3  CON2      0.75385    5.9    -0.81923    <=      0
4  X2        0.30769    0      -0.03846    <=      1
5  C_3       0         -1      -1           =      0
6  X1       -0.07692    0       0.13461    <=      1
7  X3        0.07692    1      -0.13461    <=      1
```

In using your spreadsheet to interpret the results, the following points should be noted:

(1) Any artificial variables still in the basis label rows with '`=`' signs. All the other rows, besides the objective, have '`<=`' signs. Artificial variables that have left the basis are indicated in the top row with a '`*`' before the name.

(2) The values of the objective function and basic variables are in the column on the right.

(3) The objective function row conforms to the convention used for input of DIF files. Thus for a maximization problem the signs here (except in the last column) are reversed from what they would be in '`Tableau Show tableau`', and the negatives of the shadow prices are shown rather than the shadow prices themselves.