gretl + lpsolve

Allin Cottrell

August 8, 2022

1 Introduction

As of the 2021d release gretl offers support for lpsolve, an open-source library for solving linear programming problems.

While linear programming is not part of the standard econometrics toolkit it surely falls under the rubric adopted in the founding of the Econometric Society in 1930, namely "to promote studies that aim at a unification between the theoretical-quantitative and the empirical-quantitative approach to economic problems."[1] And some well known econometric studies have used linear programs as a complement to estimation methodology—see for example Ferrier and Lovell (1990).

Section 7 gives specifics per platform. Note that on Linux you have to install the lpsolve package to activate gretl's support.

2 LP basics

We're not going to say much here about Linear Programming in general, or lpsolve's `lp` file format in particular, since there's plenty of good material at http://lpsolve.sourceforge.net/5.5/. But we'll say enough to establish notation for what follows, and give a brief overview of `lp` format.

The basic expression of an LP problem in matrix form is

$$\begin{aligned} \text{maximize} \quad & c'x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where x is an n-vector of variables and c an n-vector of coefficients, such that $c'x$ constitutes the objective function. In the constraints on the variables A is an $m \times n$ matrix of coefficients and b is an m-vector of upper limits. Note that in `lpsolve` non-negativity constraints are implicit for all elements of x; if you need to allow any negative values you must contradict this assumption explicitly, as in "`x1 >= -100;`".

The native lpsolve file format—for which the standard filename extension is "`.lp`"—has the following basic characteristics.

- Names of variables must start with a letter.
- A comment can be opened with "`/*`" and closed with "`*/`", or "`//`" can be used for a single-line comment.
- All statements are terminated with a semicolon.
- The first statement should give the objective function, as a linear combination of variables (or a single variable).

[1] See https://www.econometricsociety.org/society/about.

- Constraints are expressed as a linear combination of variables followed by an operator, followed by a constant. The operator must be <=, <, >=, > or =. But note that lpsolve takes all inequality constraints to be real-valued, so <= and < are strictly equivalent, as are >= and >.
- A trailing statement starting with the keyword `int` can be given to identify variables that are supposed to be integer valued, as in "`int A, B;`".

In `lp` format, each term of a linear combination must take the form of a numerical constant (or an implicit 1) followed by the name of a variable, as in "`3 x`" or just "`3x`". Multiplication is implicit (the symbol "*" is accepted but is not required or idiomatic). Parentheses are not accepted. Here's an example constraint:

```
A - 10 m1A - 20 m2A - 30 m3A = 0;
```

See Section 8 for a full example of usage.

3 Implementation of lpsolve support

There are two main ways of using lpsolve from within gretl.

- In the GUI program you can load or compose a linear program in the native format of lpsolve, send it for solution, and retrieve the results as native gretl objects.
- Using either the GUI program or the command-line program `gretlcli`, you can formulate a linear program in various ways and call the `lpsolve()` function to obtain the solution in the form of a gretl bundle.

Some details follow.

Via the GUI script editor

A script in the default format of lpsolve can be loaded into gretl by via the File/Script files menu or drag and drop. Gretl expects a filename suffix of "`.lp`" for such files. In the script editor you can modify the file if you wish, and execute it via the toolbar or the `Ctrl+R` shortcut (Figure 1). The solution will be shown in an output window. This window's toolbar will include a bundle button which pulls down a menu allowing you to save the solution data (Figure 2)

Via the `lpsolve` function

The `lpsolve()` function has this signature:

```
bundle lpsolve (const bundle specs)
```

The input bundle, `specs`, must hold a specification of the problem to be solved and may hold some optional elements. The problem specification itself may take any of these three forms:

- Under the key `lp_filename`, the name of an `lp` file.
- Under the key `lp_buffer`, a multi-line string in `lp` format. Such a string can be composed using gretl's `sprintf` function and string concatenation. You can think of this as an `lp` file existing in memory rather than on disk.
- Under the key `lp_bundle`, a gretl bundle in which a linear program is represented by (primarily) a set of matrices. The requirements for such a bundle are set out in section 6 below.

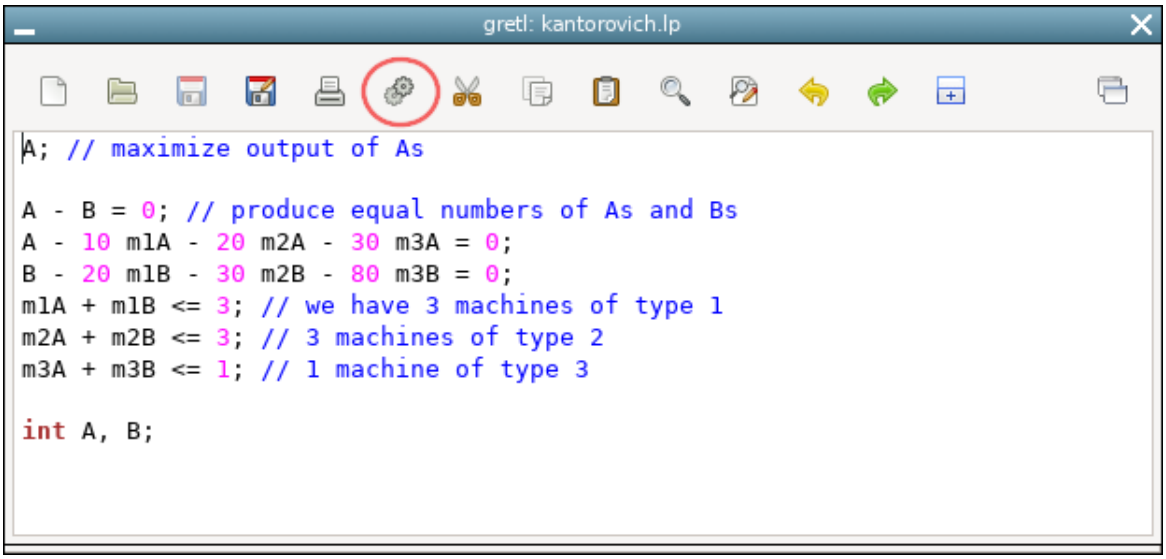

Figure 1: Editing an lpsolve script. The execute icon is the sixth from the left.

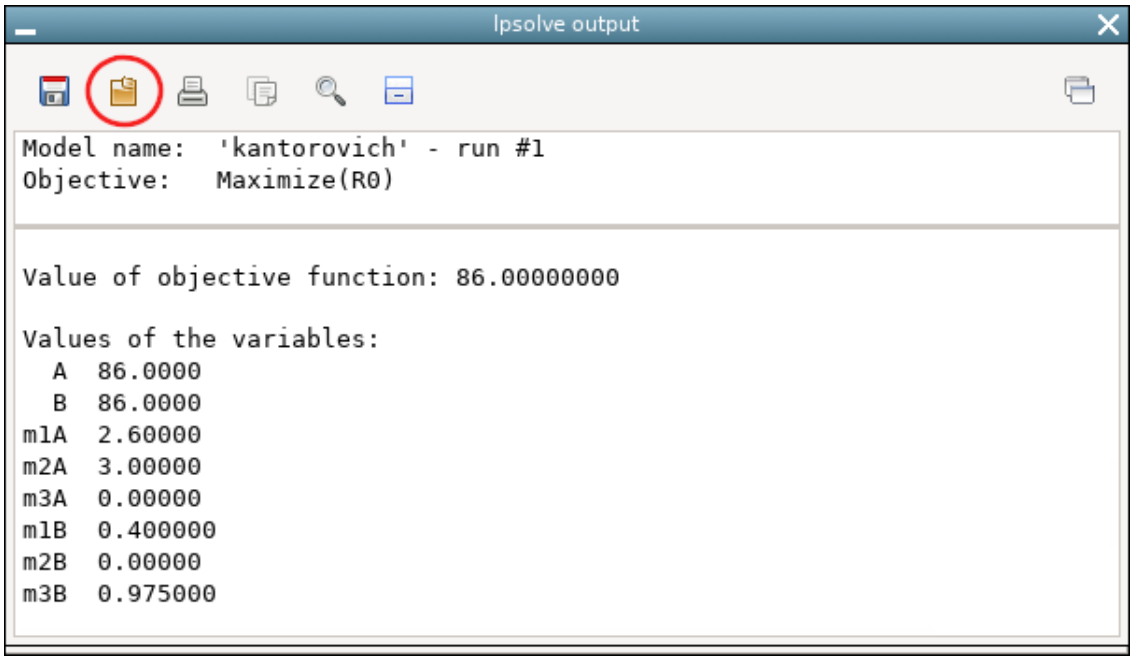

Figure 2: Partial output from an lpsolve run. The bundle-menu icon is the second from the left.

4 Top-level options

By "top-level" we mean that these options should be included directly in the `specs` bundle passed to `lpsolve()`. They apply regardless of the method by which the linear program is specified.

- `minimize` (boolean, default 0): If this has a non-zero value the objective is minimized; by default it is maximized.
- `verbose` (boolean, default 0): If this has a non-zero value details of the solution process and results are printed. Otherwise `lpsolve()` runs silently unless the solver fails.
- `sensitivity` (boolean, default 0): If this has a non-zero value additional output is provided as described in section 5.
- `model_name` (string): If this is included it is shown when verbose output is produced.

5 Contents of the return bundle

Let n and m denote, respectively, the number of variables whose values are to be optimized and the number of linear constraints.

The bundle returned by the lpsolve function includes the following:

- `objective`: a scalar, the optimized value of the objective function.
- `variable_values`: an n-vector holding the optimized values of the variables.
- `constraint_values`: an m-vector holding the values of the constrained magnitudes at the solution.
- `shadow_prices`: an m-vector holding the Lagrange multipliers associated with the constraints.

In addition, if the lpsolve library version is 5.5.2.11 or higher a scalar `accuracy` value is included. This gives the accuracy with which the constraints are met; it is 0 at an exact solution, larger values indicate approximation.

If the sensitivity option is activated the bundle contains an additional $(n+m)\times 3$ matrix under the name `sensitivity`. This holds all the dual variables along with the lower and upper limits of their validity.

6 Specifying a linear program via bundle

A bundle given as `lp_bundle` must have at least three members, as follows.

1. `objective`: a row vector of length n holding the coefficients on the variables in the linear objective function. Zeros must be inserted for variables that do not feature in the objective. If column names are attached to `objective` the variables are named accordingly in the printed output (if any) and the rows of `variable_values` in the output bundle are named.
2. `constraints`: a matrix of dimension $m \times (n+1)$. Row i holds the coefficient on each of the variables in constraint i (again, with zeros inserted for any variables not referenced in the constraint), followed by the right-hand side value. If row names are attached to `constraints`, these names are used in the output.
3. `ctypes`: a specification of the types of the constraints (inequalities or equalities). It can take the form of an array of m strings, each of which must be "<=", ">=" or "=", or alternatively an m-vector in which these types are encoded as 1, 2 or 3, respectively.

If you require that certain variables be integer-valued, you should supply a fourth member of `lp_bundle`: a matrix under the key `intvars`. This is a vector holding the 1-based indices of the variables in question.

7 Platform specifics

In the gretl packages for MS Windows and macOS, the lpsolve library (as of this writing, version 5.5.2.11) is included; no additional installation is required.

On Linux, however, it's easy to install the lpsolve library via the package manager so we leave that up to the user. Here are the commands that should be used on three popular Linux distributions:

Debian	`sudo apt-get install lp-solve`
Fedora	`sudo dnf install lpsolve`
Arch	`sudo pacman -S lpsolve`

After executing the relevant command it may be necessary to follow up with

```
sudo ldconfig
```

to ensure that the newly installed library can be found by gretl.

8 Example

Consider L. V. Kantorovich's illustrative "Example 1" of 1939 (translated in Kantorovich (1960)), one of the first linear programs ever posed and solved. We require that two components of a finished product, A and B, be produced in equal number and that this number be maximized. The constraints take the form of the numbers and capacities of machines of three different types that are available for producing the components. The producer has three machines of type 1, three of type 2, and one of type 3. The capacities of these machines (maximum output per working day) are

	type 1	type 2	type 3
A	10	20	30
B	20	30	80

To set up the LP problem we introduce the variables m1A, m2A, m3A, m1B, m2B and m3B, where mij represents the type i machine-days ($i = 1, 2, 3$) devoted to production of component j (j = A, B). The lpsolve script for this problem was shown in Figure 1; we reproduce it in Listing 1 in case you want to copy and paste. Listing 2 shows a simple script which passes `kantorovich.lp` to the lpsolve library and returns a bundle containing the results.

Listing 3 shows how the same problem can be specified by means of a gretl bundle rather than an `lp` script. The bundle method may appear to be more cumbersome, but note that it permits expression of the constraints in a structured manner, using primary information such as (in the Kantorovich example) the matrix P which holds the machine productivities.

Listing 4 is a self-contained script for execution by gretl which solves the problem (without printing anything) and illustrates usage of information from the output bundle. In the optimal solution to the Kantorovich problem 86 As and Bs are produced. The script shows the usage rates of the three machine types: machines of type 1 and 2 are fully employed but the type 3 machine has a little down-time, being employed to 97.5% of its capacity. The second piece of analysis shows that machines of types 2 and 3 are fully specialized according to their comparative advantage, while the type 1 machines produce both As and Bs.

References

Ferrier, G. D. and C. A. K. Lovell (1990) 'Measuring cost efficiency in banking: Econometric and linear programming evidence', *Journal of Econometrics* 46: 229–245. URL https://doi.org/10.1016/0304-4076(90)90057-Z.

Kantorovich, L. V. (1960) 'Mathematical methods of organizing and planning production', *Management Science* 6(4): 363–505.

Listing 1: Kantorovich's Example 1 as file `kantorovich.lp`

```
A; // maximize output of As

A - B = 0; // produce equal numbers of As and Bs
A - 10 m1A - 20 m2A - 30 m3A = 0;
B - 20 m1B - 30 m2B - 80 m3B = 0;
m1A + m1B <= 3; // we have 3 machines of type 1
m2A + m2B <= 3; // and 3 machines of type 2
m3A + m3B <= 1; // and 1 machine of type 3

int A, B; // require that A, B values be integers
```

Listing 2: Executing Kantorovich example in gretl

```
bundle b = _(lp_filename="kantorovich.lp", verbose=1)
bundle solution = lpsolve(b)
```

Listing 3: Kantorovich example specified via bundle

```
set verbose off
string vnames = "A B m1A m2A m3A m1B m2B m3B"
nvars = 8

matrix Obj = zeros(1, nvars)
Obj[1] = 1
cnameset(Obj, vnames)
print Obj

# machine productivities (rows A B, cols m1 m2 m3)
matrix P = {10, 20, 30; 20, 30, 80}

matrix C = zeros(6, nvars+1)
C[1,1:2] = {1, -1}
C[2,1] = 1
C[2,3:5] = -P[1,]
C[3,2] = 1
C[3,6:8] = -P[2,]
C[4:6,3:9] = I(3) ~ I(3) ~ {3,3,1}'
string rnames = "A=B Asum Bsum nm1 nm2 nm3"
rnameset(C, rnames)
print C
strings Ops = defarray("=", "=", "=", "<=", "<=", "<=")

bundle lpb = _(objective=Obj, constraints=C, ctypes=Ops, intvars={1,2})
bundle specs = _(lp_bundle=lpb, model_name="kantorovich", verbose=1)
bundle ret = lpsolve(specs)
```

Listing 4: Use of information from the bundle returned by `lpsolve`

Input:

```
set verbose off
bundle b = _(lp_filename="kantorovich.lp")
ret = lpsolve(b)
printf "\nTotal number of As and Bs produced: %d\n\n", ret.objective

matrix vv = ret.variable_values
matrix cv = ret.constraint_values

matrix u = cv[4:6]'
cnameset(u, "m1 m2 m3")
printf "Usage levels of machines:\n\n%8g\n", u

# machine productivities (rows: A, B)
matrix P = {10, 20, 30; 20, 30, 80}

# Outputs per machine type
matrix Q = zeros(2,3)
Q[1,] = vv[3:5]' .* P[1,]
Q[2,] = vv[6:8]' .* P[2,]
Q ~= sumr(Q)
cnameset(Q, "m1 m2 m3 sum")
rnameset(Q, "A B")
printf "Numbers of As and Bs produced per machine type:\n\n%6g\n", Q
```

Output

```
Total number of As and Bs produced: 86

Usage levels of machines:

      m1      m2      m3
       3       3   0.975

Numbers of As and Bs produced per machine type:

      m1    m2    m3   sum
A     26    60     0    86
B      8     0    78    86
```