Linear Programming

Documentation Version: January 15, 2001

Contents

Chapter 1

Installation

1.1 UNIX

If you are unfamiliar with UNIX, see your system administrator or system documentation for information on the system commands referred to below. The device names given are probably correct for your system.

1. Use `cd` to make the directory containing **GAUSS** the current working directory.
2. Use `tar` to extract the files.

   `tar xvf` *device_name*

   If this software came on diskettes, repeat the `tar` command for each diskette.

The following device names are suggestions. See your system administrator. If you are using Solaris 2.x, see Section 1.1.1.

Operating System	3.5-inch diskette	1/4-inch tape	DAT tape
Solaris 1.x SPARC	`/dev/rfd0`	`/dev/rst8`	
Solaris 2.x SPARC	`/dev/rfd0a` (vol. mgt. off)	`/dev/rst12`	`/dev/rmt/1l`
Solaris 2.x SPARC	`/vol/dev/aliases/floppy0`	`/dev/rst12`	`/dev/rmt/1l`
Solaris 2.x x86	`/dev/rfd0c` (vol. mgt. off)		`/dev/rmt/1l`
Solaris 2.x x86	`/vol/dev/aliases/floppy0`		`/dev/rmt/1l`
HP-UX	`/dev/rfloppy/c20Ad1s0`		`/dev/rmt/0m`
IBM AIX	`/dev/rfd0`	`/dev/rmt.0`	
SGI IRIX	`/dev/rdsk/fds0d2.3.5hi`		

1.1.1 Solaris 2.x Volume Management

If Solaris 2.x volume management is running, insert the floppy disk and type

```
volcheck
```

to signal the system to mount the floppy.

The floppy device names for Solaris 2.x change when the volume manager is turned off and on. To turn off volume management, become the superuser and type

```
/etc/init.d/volmgt off
```

To turn on volume management, become the superuser and type

```
/etc/init.d/volmgt on
```

1.2 DOS

1. Place the diskette in a floppy drive.
2. Log onto the root directory of the diskette drive. For example:

   ```
   A:<enter>
   cd\<enter>
   ```

3. Type: **ginstall** *source_drive target_path*

source_drive	Drive containing files to install with colon included For example: **A:**
target_path	Main drive and subdirectory to install to without a final \ For example: **C:\GAUSS**

   A directory structure will be created if it does not already exist and the files will be copied over.

*target_path***src**	source code files
*target_path***lib**	library files
*target_path***examples**	example files

4. The screen output option used may require that the DOS screen driver ANSI.SYS be installed on your system. If ANSI.SYS is not already installed on your system, you can put the command like this one in your CONFIG.SYS file:

   DEVICE=C:\DOS\ANSI.SYS

   (This particular statement assumes that the file ANSI.SYS is on the subdirectory DOS; modify as necessary to indicate the location of your copy of ANSI.SYS.)

1.3 Differences Between the UNIX and DOS Versions

- In the DOS version, when the global **__output** = 2, information may be written to the screen using commands requiring the ANSI.SYS screen driver. These are not available in the current UNIX version, and therefore setting **__output** = 2 may have the same effect as setting **__output** = 1.

- If the functions can be controlled during execution by entering keystrokes from the keyboard, it may be necessary to press *Enter* after the keystroke in the UNIX version.

- On the Intel math coprocessors used by the DOS machines, intermediate calculations have 80-bit precision, while on the current UNIX machines, all calculations are in 64-bit precision. For this reason, **GAUSS** programs executed under UNIX may produce slightly different results, due to differences in roundoff, from those executed under DOS.

Chapter 2

Linear Programming

by
Donna Calhoun
Aptech Systems

2.1 Introduction

This module contains procedures for solving small scale linear programming problems.

A linear programming problem is a optimization problem presented in the following typical manner:

$$
\begin{array}{lll}
(*) \quad \text{maximize:} & \displaystyle\sum_{j=1}^{n} c_j x_j & \\
\text{subject to:} & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \leq b_i & (i = 1, 2, ...m) \\
& l_j \leq x_j \leq u_j & (j = 1, 2, ...n)
\end{array}
$$

where a, b, c, l and u are user-supplied vectors and matrices. The expression $c \cdot x$ is called the *objective function*, the system $\{a_i \cdot x \leq b_i\}_{i=1}^{m}$ make up the *constraints*, and the inequalities $l_j \leq x_j \leq u_j$ describe the *variable bounds*.

If the constraints in $(*)$ can be satisfied and the problem is not unbounded, then $(*)$ has an *optimal solution* and an *optimal value*. In this case, x is the optimal solution and the value of the expression $c \cdot x$ at the optimal solution is the optimal value.

To solve the above problem and its variations, **simplex** uses the two-phase standard revised simplex method with an eta factorization similar to the product form of the inverse.

2.2 Getting Started

GAUSS 3.1.0+ is required to use these routines.

2.2.1 README Files

The file `README.lp` contains any last minute information on this module. Please read it before using the procedures in this module.

2.2.2 Setup

In order to use the procedures in the *LINEAR PROGRAMMING* module the `simplex` library must be active. This is done by including `simplex` in the **LIBRARY** statement at the top of your program:

```
library simplex,nlsys,pgraph;
```

This enables **GAUSS** to find the procedures contained in this module.

If you plan to make any right-hand references to the global variables (described in section 2.3.2 and under the **simplex** function definition in Chapter 3) you also need the statement:

```
#include simplex.ext;
```

Finally, to reset global variables in succeeding executions of the program the following instruction can be used:

```
lpset;
```

This could be included with the above statements without harm and would insure the proper definition of the global variables for all executions of the program.

The version number of each module is stored in a global variable. For the *LINEAR PROGRAMMING* module, this global is:

_lp_ver 3×1 matrix, the first element contains the major version number, the second element the minor version number, and the third element the revision number.

If you call for technical support, you may be asked for the version of your copy of this module.

2.3 About the simplex Procedure

To call **simplex**, use the following syntax:

```
{ x,optval,retcode } = simplex(a,b,c,l,u);
```

Or, simply

```
call simplex(a,b,c,l,u);
```

if return values are not needed.

In the above, a is an MxN matrix of coefficients for the M constraint equations in N unknowns, b is an Mx1 vector or Mx2 matrix which defines the bounds on the constraints, c is an Nx1 vector containing coefficients of the objective function, and l and u are Nx1 vectors and define the lower and upper bounds on the variables, respectively. The values returned by **simplex** are the final solution (x), the value of the objective function upon termination of the algorithm (*optval*), and a return code (*retcode*). Other output information such as dual variables, a final basis, and quality of the solution are returned in global variables. See the discussion of global control variables in Section 2.3.2.

The variable names used here are standard and are used throughout this documentation and the example programs.

2.3.1 Setting Up Arguments to simplex

Here is a sample program:

```
library simplex;
lpset;

a = { 2 -3 4 1 3,
      1 7 3 -2 1,
      5 4 -6 2 3 };

b = { 1, 1, 22 };

c = { 8, -9, 12, 4, 11 };

l = 0;
u = 1e200;
__output = 1;
output reset;

{ x,optval,retcode } = lpprt(simplex(a,b,c,l,u));

output off;
```

Here **simplex** uses all default values (maximization problem with $\leq$ constraints), which were initialized with the statement

```
lpset;
```

and returns the final solution, the value of the objective function upon termination, and a return code. Output from **simplex** is sent to the screen and a disk file.

In this example, the arguments are set up explicitly in the program. In general, however, the user can set up arguments to **simplex** in any way he or she chooses. The data may be loaded from ASCII files, from matrices previously saved to disk in an `.fmt` file, or set up explicitly in the program calling **simplex**.

As the above sample program illustrates, the arguments l and u may take on the values $+\infty$ or $-\infty$. In **simplex**, $-\infty$ is represented by $-1e200$ and $+\infty$ by $1e200$. By setting $l = 0$ and $u = 1e200$, the variables x_j are restricted to nonnegative values. Here are examples of two other ways to set up l and u:

(1)

```
l = -1e200;
u = 50;
```

(2)

```
l = { 0, -1e200, -50, -1e200 };
u = { 1e200, 0, 50, 1e200 };
```

In (1), all variables are bounded below by $-\infty$ and above by 50.

In (2), the variables are restricted as follows:

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\leq 0 \\ -50 \leq x_3 &\leq 50 \\ -\infty \leq x_4 &\leq +\infty \end{aligned}$$

The argument b is used to provide upper and/or lower bounds for the constraint expressions and, if desired, to define constraint types. Usually, though, constraint types ($\leq$, $\geq$, $=$) are defined using global variables. This is discussed next. For more details on defining b see the **simplex** function definition in Chapter 3. Please note that **simplex** cannot handle free constraints. Do not set $b_i = \pm 1e200$ for any i.

2.3.2 The Global Control Variables

Once the arguments to **simplex** are set up, you probably want to customize your program. Almost all aspects of the linear programming problem, including the constraint type and variable bounds, can be modified by changing the value of one or more of the global control variables provided by **simplex**. A complete list of all globals is given under the **simplex** function definition in Chapter 3. Described below are some of the aspects that the user can customize and the global variable used in each case:

- Whether **simplex** should solve the minimization or maximization problem. (**_lpmin**)
- The variables bounds. In the case where the user wants simply to define variables as nonnegative, nonpositive or free, a global variable may be used to indicate variable types, rather than explicitly setting l and u. (**_lpxcnst**)
- The constraint type ($\leq$, $\geq$, $=$). (**_lpcnst**).

- Whether to report a feasible solution only (i.e., terminate with Phase I) or return the optimal solution (i.e., continue on to Phase II). (**_lpstflg**)
- Maximum number of iterations that the algorithm is allowed to execute. (**_lpmaxit**)
- The choice of starting value. This is useful if the user plans to solve several similar problems (e.g., problems which vary only in the b vector). (**_lpstrtx**)
- The type of output desired. The user can specify whether **simplex** should produce output suitable for a disk file or the screen. Also, the user may customize the output with a title, variable names and header of his/her choosing. (**__output**, **_lpname**, **__title**, **__altnam**, **__header**)

Globals also control more advanced options of the solution process. For a brief discussion on how to control these options, see Section 2.5. The advanced options include:

- The tie breaking rules which are used to determine the entering variable and the leaving variable. (**_lprule**)
- The tolerances used in determining the entering and leaving variables. (**_lpeps1**, **_lpeps2**)
- Other tolerances used to minimize roundoff errors. (**_lpeps3**, **_lpcstol**, **_lpfstol**)
- The number of solutions returned. If the solution first found by **simplex** is not unique, the user can specify how many more optimal solutions **simplex** should attempt to find. (**_lpsoltn**, **_lpuniq**)

Using the Global Control Variables

To use the global variables, simply assign the desired value to the selected global variable in the **GAUSS** program before calling **simplex** (but after the call to **lpset**, if any). The following is an example of how to solve a minimization problem with equality constraints and free variables:

```
library simplex;
lpset;
a = trunc(100*rndu(20,30));
b = 100*ones(20,1);
c = ones(30,1);
```

```
_lpxcnst = 0;   /* All variables are free             */
_lpmin = 1;     /* Solve minimization problem         */
_lpcnst = 3;    /* Constraints are all equalities     */
__output = 1;   /* Output suitable for disk file      */
_lpname = "Y";  /* Variable name to be used in output */

output file = lp1.out reset;
{ y, optval, retcode } =  lpprt(simplex(a,b,c,0,0));
output off;
```


By setting **_lpmin** = 1, the minimum value of the objective function is computed. By setting **_lpcnst** = 3, the constraint equations are treated as equalities. Here **_lpcnst** is a scalar, but in general it can be a Mx1 vector where each element describes the corresponding equation type. Also note that instead of setting $l = -1e200$ and $u = 1e200$, this program uses the global variable **_lpxcnst** to specify that the variables should be unrestricted. In this case, the values of l and u are ignored. (The two methods are equivalent. The user may choose whichever is preferable.)

The global variable **__output** has been set to specify that information generated during the iterative stages of **simplex** should be sent to the output file `lp1.out`. The call to **lpprt** prints a final report to the same disk file. In general, **__output** controls the output produced by the procedure **simplex** and can take the value 0, 1 or 2, where 0 is no output, 1 is disk output (suitable for an output file) and 2 is screen output (suitable only for the screen). Final reports can be generated with either **lpprt** (for disk file) or **lpview** (for screen display). Both final report formats report the return code, the value of the objective function upon termination, the total number of iterations required by **simplex**, final solution x (with an indication of which variables are basic upon termination), the quality of the solution, the value of the constraints and the dual variables. If output is directed to the screen, the state of each constraint is also reported.

2.4 Example Programs

These and other example programs, `simp`n`.e`, can be found on the `examples` subdirectory.

EXAMPLE 1

This program solves a straightforward linear programming problem. By default, the maximization problem is solved, the variables are restricted to nonnegative values, and the constraints are all of the type $\leq$. Since **__output** = 2, information generated during the iterations is formatted for output to the screen. The call to **simplex** is nested inside a call to **lpview**, so the final report also will be suitable only for the screen.

```
/*
**  simp1.e
*/
library simplex;
lpset;
a = { 2 -6 2 7 3 8,
     -3 -1 4 -3 1 2,
      8 -3 5 -2 0 2,
      4 0 8 7 -1 3,
      5 2 -3 6 -2 -1 };

b = { 1, 2, 4, 1, 5 };

c = { 18, -7, 12, 5, 0, 8 };

__output = 2;
_lpname ="Y";
__title = "SIMP1.E";

{ y,optval,retcode } = lpview(simplex(a,b,c,0,1e200));
```

EXAMPLE 2

A more complicated example might look like this:

```
/*
** simp2.e
*/

library simplex;
lpset;
a = { 3 1 -4 2 5 1,
     -5 4 2 -3 2 3,
      1 1 2 1 1 2 };

b = { 3, 25, 4 };

c = { -5, 2, 3, 3, 6, 1 };

l = { 0, 2, -1e200, -3, -1e200, -1e200 };

u = { 1e200, 10, 0, 3, 1e200, 1e200 };

_lpcnst = { 1, 1, 3 };
__output = 1;
output file = lp2.out reset;
```

```
{ x, optval, retcode } = lpprt(simplex(a,b,c,l,u));

output off;
```

Here **_lpcnst** is a 3x1 vector which indicates that the first two constraints are $\leq$ constraints and the final constraint is an equality constraint. The variables should all satisfy the inequalities:

$$\begin{aligned} x_1 &\geq 0 \\ 2 \leq x_2 &\leq 10 \\ x_3 &\leq 0 \\ -3 \leq x_4 &\leq 3 \end{aligned}$$

x_5 and x_6 are free variables.

Results of the algorithm's progress and the final solution report are sent to the output file `lp2.out`.

EXAMPLE 3

In this example both the primal and the dual problem are solved. The fundamental principle of linear programming states that the optimal value of the primal is equal to the optimal value of the dual problem (assuming both have optimal solutions). Then the solution to one problem is compared with the dual variables of another. Here, the statement

```
#include simplex.ext;
```

is needed because a right-hand reference to a global variable (in this case, **_lpdlvar**) has been made.

```
/*
** simp3.e
** This example illustrates how to solve
** both a primal and dual problem
*/

library simplex;
#include simplex.ext;
lpset;
a = { 4  0 -1  1,
      2  1  4 -1,
     -3  2  0 -8,
      1  1  1  1 };
```

```
b = { 2, 12, -31, 12 };

c = { -2, -9, -1, 6 };

l = 0;
u = 1e200;
_lpmin = 1;
_lpcnst = { 3, 2, 3, 1 };
__output = 0;
__title = "PRIMAL PROBLEM";
output file = lp3.out reset;

{ x, optp, retcp } = lpprt(simplex(a,b,c,l,u));

dlp = _lpdlvar;
lpset;
_lpmin = 0;
_lpxcnst = { 0, 1, 0, -1 };    /*  l and u set to 0 below */
_lpcnst = 1;
__output = 0;
__title = "DUAL PROBLEM";
_lpname = "Y";

{ y, optd, retcd } = lpprt(simplex(a',c,b,0,0));

dld = _lpdlvar;
output off;
```

EXAMPLE 4

Suppose that the user wishes to solve multiple linear programming problems using the same objective function, constraints and variable bounds, but wishes to modify the b vector. This can of course be done by solving each problem from scratch, but this approach is unnecessarily time-consuming. An alternative method is to solve one problem from scratch and use the optimal solution and final basis found in that problem to initialize the following problems. This can be done by setting the global variable **__lpstrtx**.

For this example the variables a, **__lpcnst**, **__lpxcnst** and c have already been saved to disk in `.fmt` files for use by multiple programs. The variables x and **__lpbasis** have been saved to disk by a previously run program. All the current program needs to do is create the new b vector and initialize **simplex** by setting up **__lpstrtx**. Here, a is 20x10. The columns of the matrix *bmat* are the b vectors used by the two problems.

```
/*
** an example similar to simp4.e
*/

library simplex;
#include simplex.ext;
lpset;

load a, x, c;
load _lpcnst = lpcnst;
load _lpxcnst = lpxcnst;
load _lpbasis = lpbasis;
load bmat[20,2]  = b.arg;
b = bmat[.,2];
m = rows(a);
n = cols(a);
tmp = zeros(m+n,1);
tmp[_lpbasis] = ones(m,1);
_lpstrtx = x~tmp;
__output = 0;

{ x, optval, retcode } = simplex(a,b,c,0,0);

save x, lpbasis;
```

Here, the first column of *bmat* was used to solve the original problem from scratch, and the second column is now being used to solve the second problem. Because this problem is starting with a value calculated to be feasible and optimal by a previous similar problem, this problem may require as few as half the number of iterations it would require were it starting from scratch. The new optimal solution and the final basis are saved to disk for use by another problem. Again, note the use of the statement

```
#include simplex.ext;
```

to make the global variable **_lpbasis** accessible.

2.5 Advanced Topics Using simplex

By changing the values of the tolerances **_lpeps1**, **_lpeps2**, **_lpeps3**, **_lpfstol** and **_lpcstol**, the user can affect the speed and accuracy of **simplex**. Also, if **simplex** is returning a return code of 5 or 13, these tolerances can be modified to encourage **simplex** to return a more informative code.

If **simplex** is returning a return code of 13, **_lpeps1** or **_lpeps2** are probably set too high. Generally, by setting **_lpeps1** lower the number of variables from which **simplex** chooses the variable entering the basis is increased. The more variables from which **simplex** has to choose, the more likely it is that it will find one which doesn't cause numerical errors.

Another tolerance which the user might wish to modify is **_lpeps3**. Briefly, **_lpeps3** determines how well x, the intermediate solution determined at each iteration, should satisfy a particular expression. By modifying the value of **_lpeps3**, the user can have some affect on how much time **simplex** requires and on the accuracy of the final solution. In general, increasing the value of **_lpeps3** reduces the amount of time **simplex** requires, and decreasing **_lpeps3** should improve the accuracy of the solution.

Two other tolerances, **_lpfstol** and **_lpcstol**, are used to determine whether an optimal solution found during Phase I is feasible and whether the x found during a particular iteration satisfies the constraints to within the user's specifications.

In solving a linear programming problem, the user may find that **simplex** reports that the problem is infeasible, but also reports that the value of the objective function at termination is very small—i.e., less than 10^{-5}. In this case, the user should consider increasing **_lpfstol** to at least as large as the value of the objective function returned by **simplex**. This guarantees that **simplex** will proceed to Phase II and attempt to find an optimal solution to the problem.

Due to scaling differences among constraints, the user may wish to allow differences among what it takes to satisfy those constraints. That is, a greater degree of infeasibility may be allowed in those constraints with larger coefficients. **_lpcstol** can be set to a scalar, in which case all constraints are satisfied to within the same degree of accuracy, or to an Mx1 vector (M = **ROWS(***a***)**), in which case each constraint uses the tolerance in the corresponding element of **_lpcstol**.

A return code of 5 indicates that the algorithm required more iterations than allowed by the global variable **_lpmaxit**. If cycling is not occurring, simply increase the value of **_lpmaxit**. If it is suspected that cycling is occurring, change the value of **_lprule[2]**. Changing the rules used to choose entering and leaving variables may decrease the number of iterations required by **simplex**. It should be noted, however, that cycling is very rare.

2.6 References

Chvatal, Vašek 1983. *Linear Programming.* New York: W. H. Freeman and Company.

Chapter 3

Linear Programming Reference

Library

simplex

Purpose

Formats and prints the output from **simplex**. This printout is suitable for output to a disk file.

Format

{ *x*,*optval*,*retcode* } = **lpprt**(*x*,*optval*,*retcode*);

Input

x	Kx1 vector, the solution returned from **simplex**.
optval	scalar, the value of the objective function upon termination.
retcode	scalar, the return code returned by **simplex**.

Output

Same as input.

Globals

__altnam Nx1 character vector, alternate variable names to be used for printout purposes. These names are used in the iterations printout and in the final report. By default, the iterations report uses numbers to indicate variables and the final solution report uses **__lpname**.

__header string. This is used by **lpprt** to display information about the date, time, version of module, etc. The string can contain one or more of the following characters:

"t"	print title (see **__title**)
"l"	bracket title with lines
"d"	print date and time
"v"	print version number of module
"f"	print file name being analyzed (NOT USED BY **simplex**)

Example:

```
__header = "tld";
```

Default = "tldvf".

_lpname string, single character, the variable name to be used for output purposes. Default = "X".

__title string, message printed at the top of the screen and printed out by **lpprt** and **lpview**. By default, no title is printed.

Remarks

The call to **simplex** can be nested inside the call to **lpprt**:

```
{ x,optval, retcode } = lpprt(simplex(a,b,c,l,u));
```

lpprt requires that the output globals returned by **simplex** still be in memory. Either nest the call to **simplex** inside the call to **lpprt** (the safest way) or call **lpprt** directly after the call to **simplex**.

Globals

_lpmin, **_lpcnst**, **_lpdlvar**, **_lpbasis**, **_lpitnum**, **_lpstate**, **_lpax**, **_lpqual**, **_lp_phse**, **_lp_b**

Source

`lpprt.src`

Library

simplex

Purpose

Initializes **simplex** global variables to default values.

Format

lpset;

Input

None

Output

None

Remarks

Putting this instruction at the top of all programs that invoke **simplex** is generally good practice. This prevents globals from being inappropriately defined when a program is run several times, or when a program is run after another program has executed that calls **simplex**.

lpset calls **gausset**.

Source

simplex.src

Library

`simplex`

Purpose

Creates a screen display of the final results returned from **simplex**. This display allows the user to page through the values of constraints upon termination, the dual variables and the final solution. The state of each constraint is reported and slack variables are marked.

Format

{ *x*,*optval*,*retcode* } = **lpview(***x*,*optval*,*retcode***);**

Input

x	Kx1 vector, the solution returned from **simplex**.
optval	scalar, the value of the objective function upon termination.
retcode	scalar, the return code returned by **simplex**.

Output

Same as input.

Globals

__altnam	Nx1 character vector, alternate variable names to be used for printout purposes. These names are used in the iterations printout and in the final report. By default, the iterations report uses numbers to indicate variables and the final solution report uses **_lpname**.
__header	string. This is used by **lpview** to display information about the date, time, version of module, etc. In **lpview**, if **__header** is set to anything other than the null string (""), a complete header is printed.
_lpname	string, single character, the variable name to be used for output purposes. Default = "X".
__title	string, message printed at the top of the screen and printed out by **lpprt** and **lpview**. By default, no title is printed.

Remarks

The call to **simplex** can be nested inside the call to **lpview**:

```
{ x,optval, retcode } = lpview(simplex(a,b,c,l,u));
```

lpview requires that the output globals returned by **simplex** still be in memory. Either nest the call to **simplex** inside the call to **lpview** (the safest way) or call **lpview** directly after the call to **simplex**.

lpview makes use of the `ANSI.SYS` escape sequences. If the user aborts execution of **simplex** during screen output, the **GAUSS** editor may remain in reverse video. To return the command mode screen to its original settings, run the command:

```
scrnrst;
```

This clears the screen.

Globals

_lpmin, **_lpcnst**, **_lpdlvar**, **_lpbasis**, **_lpitnum**, **_lpstate**, **_lpax**, **_lpqual**, **_lp_phse**, **_lp_b**

Source

`lpprt.src`

Library

`simplex`

Purpose

Computes the optimal value of a linear objective function subject to linear inequality constraints and bounds on variables. The problem typically is of the form:

$$\begin{aligned} \text{maximize:} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to:} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (i = 1, 2, ...m) \\ & l_j \leq x_j \leq u_j \qquad (j = 1, 2, ...n) \end{aligned}$$

Format

{ *x*,*optval*,*retcode* } = **simplex**(*a*,*b*,*c*,*l*,*u*);

Input

a MxN matrix of constraint coefficients. The problem should not be supplied with slack variables.

b Mx1 vector or Mx2 matrix. If *b* is Mx2, the constraint expressions are bounded below by the first column and above by the second column. That is,

$$b_{i1} \leq \sum_{j=1}^{n} a_{ij} x_j \leq b_{i2} \quad (i = 1, 2, ...m)$$

This format can be used to generate all three constraint types. For example:

$$3x_1 + 4x_2 - 13x_3 \geq 24$$

is equivalent to:

$$24 \leq 3x_1 + 4x_2 - 13x_3 \leq 1e200$$

Note the use of 1e200 for $+\infty$. This is also used below in describing variable ranges.

c Nx1 vector containing the coefficients of the objective function.

l Nx1 vector or a scalar, containing the lower bounds of x. Use $-1e200$ for $-\infty$. If l is a scalar, it is assumed that all elements of x have the same lower bound.

u Nx1 vector or a scalar, containing the upper bounds of x. Use 1e200 for $+\infty$. If u is a scalar, it is assumed that all elements of x have the same upper bound.

Output

x (N+M)x1 vector containing either (1) an optimal solution to the original problem, (2) the x values which minimize the sum of the infeasibilities or (3) the last solution found before it was determined that the problem was unbounded or that the algorithm was unable to continue. The last M elements contain the values of the slack variables.

optval scalar, the value of the objective function upon termination of **simplex**. This may be the optimal value, the minimum sum of the infeasibilities or the largest (or smallest) value found before it was determined that the problem was unbounded or that the algorithm was unable to continue.

retcode scalar, return code:

0 An optimal solution was found.

1 The problem is unbounded.

2 The problem is infeasible.

5 Maximum number of iterations exceeded. Cycling may be occurring.

13 Algorithm unable to find a suitable variable to enter the basis. Set **_lpeps1** or **_lpeps2** to a lower value or change **_lprule[1]** to another value.

If the return code is negative, the program terminated in Phase I.

Global Input

_lpcnst scalar, or Mx1 vector used to describe each equation type. Values for each equation type are:

1 Corresponding constraint is $\leq$.

2 Corresponding constraint is $\geq$.

3 Corresponding constraint is $=$.

For example,

```
_lpcnst = { 1, 3, 3, 2 };
```

If **_lpcnst** is a scalar, it is assumed that all constraints are of the same type. Default = 1.

_lpcstol scalar, or Mx1 vector. Tolerances used to determine whether a constraint has been violated. Default = 10^{-8}.

_lpeps1 scalar. This is the smallest value around which a pivot is performed. If during any iteration a value exceeds **_lpeps1** in absolute value, it is a possible candidate for entering the basis. Default = 10^{-8}.

_lpeps2 scalar. The algorithm will not divide by a number smaller than this. Default = 10^{-8}.

_lpeps3 scalar. This is used to determine how often to refactor the basis. Roughly, if $|Ax_c - b|$, where x_c is the current value of x, is greater than **_lpeps3** in any element in any iteration, the basis is refactored immediately. If $|Ax - b|$ never exceeds **_lpeps3**, then the basis is refactored when either the eta file gets too large for available memory or when scanning the file becomes too time-consuming. Default = 10^{-8}.

_lpfstol scalar. Tolerance used to determine whether a solution achieved at the end of Phase I is feasible. If the sum of artificial variables at the end of Phase 1 does not exceed **_lpfstol**, then the solution at that point is considered feasible. Default = 10^{-13}.

_lpmaxit scalar, the maximum number of iterations the simplex algorithm iterates during either phase. Default = 300.

_lpmin scalar. If 1, **simplex** attempts to solve the minimization problem. If 0, the maximization problem is solved.
Default = 0.

_lpname string, single character, the variable name to be used for output purposes. Default = "X".

_lprule 2x1 vector, the tie breaking rules used to choose the entering and leaving variables. **_lprule[1]** specifies the tie breaking rule for the entering variable and can have the following values:

1 Smallest subscript rule.

2 Largest coefficient rule.

3 Largest increase rule.

4 A random selection is made.

_lprule[2] specifies the tie breaking rule for the leaving variable and can have the following values:

1 Smallest subscript rule.

2 Lexicographic rule. This rule is very time-consuming and memory-intensive.

3 A random selection is made.

The rule used to choose the entering variable can have an effect on the number of iterations required before the algorithm finds an optimal solution. Unfortunately, no general rules can be given about which rule to use. Using the smallest subscript rule for the leaving variable guarantees that off-optimal cycling does not occur. This rule, however, may force the algorithm to go through more iterations than might otherwise be necessary.
Default = { 2, 1 }.

__lpsoltn scalar, the number of optimal solutions that **simplex** should attempt to find. Default = 1.

__lpstflg scalar. If 1, only a feasible solution is returned. If 2, **simplex** continues to Phase II and attempt to find an optimal solution. Default = 2.

__lpstrtx (N+M)x1 or (N+M)x2 vector. If **__lpstrtx** is (N+M)x1, then it is used to initialize Phase I. The first N elements are the values of the original variables and the last M elements are the values of the slack variables. If **__lpstrtx** is (N+M)x2, the first column should contain an x with which to initialize the algorithm, and the elements of the second column should be set to 1 if the corresponding first column element is basic, 0 if not. In both cases, the initial x must be feasible with respect to either l and u or to **__lpxcnst**. It need not be feasible with respect to the constraint equations. Default = 0.

__lpxcnst scalar, or Nx1 vector. This global may be used as an alternative to setting input arguments l and u in the case that the variables are to be nonnegative, nonpositive, or free. Values for this global are:

-1 Corresponding variable is nonpositive.

0 Corresponding variable is unrestricted (or free).

1 Corresponding variable is nonnegative.

If **__lpxcnst** is a scalar, it is assumed that all variables have the same restrictions. If this global has been set to a value other than its default, l and u are ignored. In this case, set
$l = u = 0$. Default = -2.

___altnam Nx1 character vector, alternate variable names to be used for printout purposes. These names are used in the iterations printout and in the final report. By default, the iterations report uses numbers to indicate variables and the final solution report uses **__lpname**.

__header string. This is used by the printout portion of **simplex** to display information about the date, time, version of module, etc. The string can contain one or more of the following characters:

"t"	print title (see **__title**)
"l"	bracket title with lines
"d"	print date and time
"v"	print version number of module
"f"	print file name being analyzed (NOT USED BY **simplex**)

Example:

```
__header = "tld";
```

Default = "tldvf".

__output scalar.

0 Nothing is written.

1 Serial ASCII output format suitable for disk files or printers.

2 (NOTE: DOS version only) output is suitable for screen only. `ANSI.SYS` must be active.

If **simplex** is terminated during execution (Ctrl-C is pressed, for example) and **__output** = 2, the screen may remain in reverse video. To restore the screen to its original settings, run the command:

```
scrnrst;
```

This clears the screen.

Default = 2.

__title string, message printed at the top of the screen and printed out by **lpprt** and **lpview**. By default, no title is printed.

Global Output

_lpax Mx1 vector, the value of each constraint upon termination. That is, **_lpax** = ax, where x is the solution upon termination of **simplex**.

_lpbasis Mx1 vector, the indices of the variables in the final basis. Normally, the indices returned in **_lpbasis** are in the range $[1, (M + N)]$. Occasionally, however, artificial variables may persist in the basis. In this case, indices are in the range $[1, (2 * M + N)]$.

_lpdlvar Mx1 vector, the dual variables.

_lpitnum 2x1 vector, the number of iterations required by each phase of **simplex**. The first and second elements correspond to the number of iterations required by Phase I and Phase II, respectively.

_lpqual scalar, reports the quality of the final solution. Quality is judged to be:

1 POOR
2 FAIR
3 GOOD
4 EXCELLENT

_lpstate Mx1 vector containing the state of each constraint upon termination of **simplex**. The states are:

-4 Equality constraint has been violated below.
-3 Equality constraint has been violated above.
-2 Constraint violates its lower bound.
-1 Constraint violates its upper bound.
0 Constraint strictly between its two bounds.
1 Constraint is at its lower bound.
2 Constraint is at its upper bound.
3 Equality constraint is satisfied.

_lpuniq scalar, if 1, the optimal solution x is unique. If 0, the solution is not unique. Set **_lpsoltn** equal to the number of optimal solutions that **simplex** should attempt to find.

_lpx (NxM)xP matrix, or scalar. If **_lpsoltn** is greater than 1, this global is an (N+M)xP matrix containing P optimal solutions. Otherwise, **_lpx** is set to zero.

Remarks

By default, **simplex** solves the problem:

$$\text{maximize:} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to:} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad (i = 1, 2, ...m)$$

$$x_j \geq 0 \qquad (j = 1, 2, ...n)$$

Please note that **simplex** cannot handle free constraints. Do not set $b_i = \pm 1e200$ for any i.

Example

```
library simplex;
#include simplex.ext;
lpset;
a = { 1 -4 3 3,
      1 3 -1 1,
      1 2 3 2,
      1 3 -2 1 };

b = { 2, -2, 3, -3 };

c = { 3, 1, 4, 2 };

__title = "THE PRIMAL PROBLEM";
_lpxcnst = { 0, 0, 1, 1 };

{ x, optp,retcp } = lpview(simplex(a,b,c,0,0));

lpset;
__title = "THE DUAL PROBLEM";
_lpcnst = { 3, 3, 2, 2 };
_lpmin = 1;

{ y, optd, retcd } = lpview(simplex(a',c,b,0,1e200));
```

Source

simplex.src

Index

T

U

V