`pplex` - A Tool for Teaching the Simplex Method

Joanna Bauer* Marc Bezem* Andreas Halle*

October 11, 2012

Abstract

Linear programs occur frequently in various important disciplines, such as economics, management, and engineering. The simplex method is the best known and most widely used method to solve linear programs. Therefore, it is taught to a wide range of students with varying background in mathematics. We present the software `pplex` for supporting the classroom presentation of the simplex method. Distinctive features of our tool are: simple command line interface, visualization (two variables), file input in standard LP format, portability, and that it is free software.

1 Introduction and Motivation

In an optimization problem[1], the objective is to decide on how to utilize given resources such that they maximize some "profit" (or minimize some "cost"), while satisfying a set of additional constraints on the resources. This is commonly modelled by associating the possible decisions with variables, such that the profit and the constraints can be formulated as functions of these variables. If the profit function, the so-called *objective (function)*, is linear and the constraints can be formulated as linear (in)equalities, then the optimization problem has a *linear programming (LP)* formulation. A specific instance of an LP formulation is called a *linear program*. A linear program is solved by identifying those values for the variables that maximize the objective while satisfying all constraints. Linear programs occur frequently in economics, management, and engineering.

The simplex method is by far the most widely used method for solving linear programs. It is listed as one of the top 10 algorithms of the twentieth century in [1]. Given the many applications of LP, the simplex method is taught to a wide range of students, including students with a weak background in mathematics and algorithms.

*University of Bergen, Department of Informatics, P.O.Box 7803, N-5020 Bergen, Norway

[1]Readers familiar with the simplex method can skip this introduction and fast forward to the motivation halfway page 5.

Most lecturers on linear programming need to demonstrate the simplex method step-by-step *without being distracted by detailed calculations in elementary linear algebra*. Let us start by introducing the simplex method by a simple example:

Maximize	x	$+$	y				(objective function)
subject to	$2x$	$+$	y	$\leq$	6		(constraint)
and	$7x$	$+$	$13y$	$\leq$	40		(constraint)
and	x	,	y	$\geq$	0		(two constraints)

where the variables x and y range over the real numbers. We write $x, y \geq 0$ to indicate that both x and y are required to be non-negative.

A *feasible* point is an assignment of real numbers to the variables such that all constraints are satisfied. The inequality constraints correspond to halfspaces (here halfplanes), and the constraints of a program define the *polyhedron* containing the feasible points. In the example above there exist feasible points, since the polyhedron given by the four inequalities contains, among other points, the origin (see Figure 1).

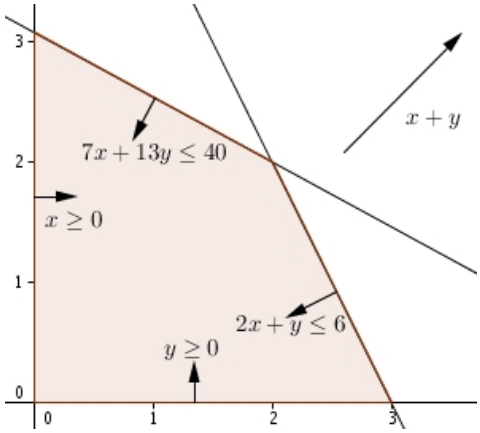


Figure 1: Geometric interpretation of our example linear program

Since inequalities cannot be manipulated algebraically as easily as equations, a first step is to reformulate the program such that it is expressed by linear equations in combination with simple inequalities of the form *variable* ≥ 0. For this purpose, we introduce two new variables, here u and v, called *slack variables*, measuring to what extent the linear inequalities are satisfied. We also introduce a new variable ζ for the objective function $x + y$. Thus our linear program is brought into the following form, called a *dictionary* in [2]:

ζ	$=$			x	$+$	y
u	$=$	6	$-$	$2x$	$-$	y
v	$=$	40	$-$	$7x$	$-$	$13y$
x, y, u, v	$\geq$	0				

Since we now deal with two equalities in four variables, assigning values to any variable pair determines the values of the remaining two variables.

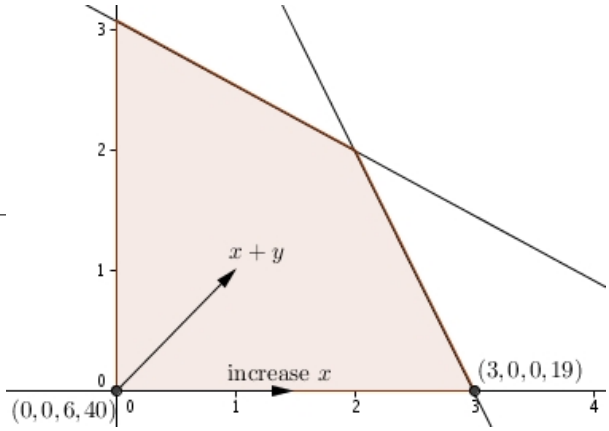
Geometric considerations lead to the observation that the maximal value of the objective subject to the constraints (linear equalities and inequalities of the form *variable* ≥ 0) is attained in a vertex of the polyhedron. Through the introduction of the slack variables, each edge of the polyhedron now corresponds to one variable being zero. Thus assigning zero to a pair of variables corresponds to the intersection point of the corresponding pair of edges. This intersection point either lies outside the polyhedron (f.ex. for $y = v = 0$), or it is a vertex of the polyhedron and thus a potential candidate for the optimal solution. For this reason, we are especially interested in assignments to (x, y, u, v) where two variables are set to zero. Such an assignment is called a *basic solution*. The variables being zero are called *non-basic* variables.

Every basic solution corresponds to a dictionary with the non-basic variables on the right side of "=", enforcing the variables on the left (the *basic* variables) to equal the constants in the equations. If one of the constants is negative, then the corresponding basic solution is not feasible. (Geometrically, a non-feasible basic solution lies outside of the polyhedron with respect to the inequality corresponding to the variable that is assigned a negative value.) We call a dictionary *(in)feasible* if the corresponding basic solution is (not) feasible.

In the example above, we start in the origin $x = y = 0$ and get value 0 for the objective $\zeta = x + y$. Clearly, there is room for improvement here, since in fact the value 0 is the *minimal* value of the objective under the given constraints. How can we improve on this value? The answer is simple: increase x or y or both. The simplex method chooses one non-basic variable and tries to increase it as much as possible, while keeping the other non-basic variables constant zero. Let's try to increase x keeping $y = 0$. How much can x increase? We see that $0 \leq u = 6 - 2x - y$ allows us to increase x to 3, and that the other linear equation allows an even larger increase. Since both $u \geq 0$ and $v \geq 0$ must be respected, we increase x to 3, thereby getting $u = 0$ as a consequence. Moreover, the objective $x + y$ evaluates to 3.

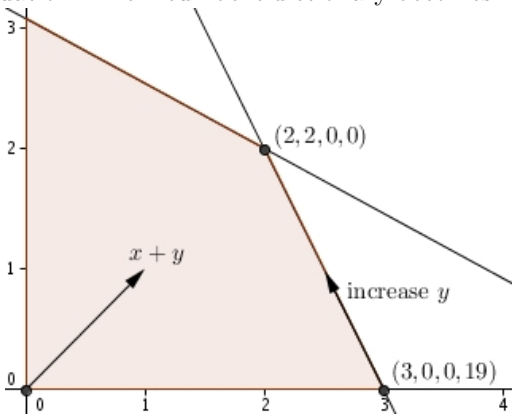
With its value increased to 3, variable x does not qualify as a non-basic variable anymore. Fortunately, the variable u has become zero. Interchanging the roles of x and u as non-basic and basic variable, respectively, restores the invariant of the dictionary form. The limiting equation $u = 6 - 2x - y$ makes it possible to express x in u and y such that x can be eliminated from the right-hand side of the dictionary form. The equation $u = 6 - 2x - y$ is called the *pivot* and interchanging the roles of a basic and a non-basic variable is called *pivoting*. In the dictionary, pivoting is done by standard row operations in linear algebra. The incumbent dictionary becomes:

$$\begin{array}{rcrcrcr} \zeta & = & 3 & - & 0.5u & + & 0.5y \\ \hline x & = & 3 & - & 0.5u & - & 0.5y \\ v & = & 19 & + & 3.5u & - & 9.5y \end{array}$$



The next step starts by observing that the objective improves if we increase y while keeping u constant zero. We see that $0 \leq v = 19+3.5u-9.5y$ allows to increase y to 2, and that the other equation allows an even larger increase. Hence, we pivot y and v using the second equation. The incumbent dictionary becomes:

$$\begin{array}{rcrcrcr} \zeta & = & 4 & - & \frac{6}{19}u & - & \frac{1}{19}v \\ \hline x & = & 2 & - & \frac{13}{19}u & + & \frac{1}{19}v \\ y & = & 2 & + & \frac{7}{19}u & - & \frac{1}{19}v \end{array}$$



The good news is that we have found the maximum, as all coefficients in the objective are negative. The maximum value 4 of the objective is attained in the point $x = y = 2$. These values are integers by the design of the example, and can also be obtained graphically. Obtaining the last dictionary algebraically is a painful (but useful) exercise.

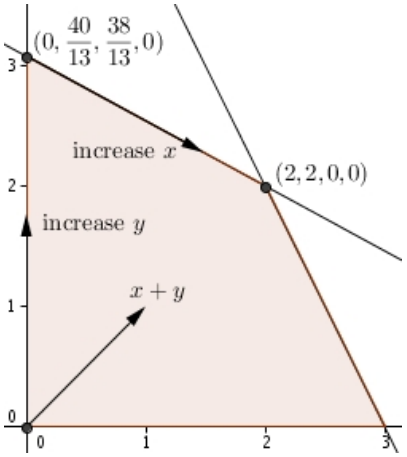
An interested student may now ask the question: Can't we start with y instead of with x? This very good question deserves a detailed answer. The lecturer (our hero) starts boldly calculating on the blackboard. Clearly, $0 \leq v = 40-7x-13y$ allows us to increase y to $\frac{40}{13}$, and the other linear equation allows an even larger increase. Hence we pivot and get, again by standard row oper-

$$\begin{array}{rcrcrcr} \zeta & = & \frac{40}{13} & + & \frac{6}{13}x & - & \frac{1}{13}v \\ \hline u & = & \frac{38}{13} & - & \frac{19}{13}x & + & \frac{1}{13}v \\ y & = & \frac{40}{13} & - & \frac{7}{13}x & - & \frac{1}{13}v \end{array}$$

ations, the new dictionary:

At this point the lecturer almost regrets his willingness to answer the student's question in detail, but manages as by miracle to finally produce the following dictionary:

$$\begin{array}{rcrcrcr} \zeta & = & 4 & - & \frac{6}{19}u & - & \frac{1}{19}v \\ \hline x & = & 2 & - & \frac{13}{19}u & + & \frac{1}{19}v \\ y & = & 2 & + & \frac{7}{19}u & - & \frac{2}{19}v \end{array}$$



A Happy Ending? Not yet. Some attentive students point out that the two final dictionaries are not completely identical and ask for an explanation. After some discussion, it turns out that the fraction $\frac{2}{19}$ in the (last) final dictionary is correct, and that the corresponding fraction $\frac{1}{19}$ in the (previous) final dictionary was wrong. The mistake may have gone unnoticed, since it didn't spoil the answer, the maximum stays 4 at $x = y = 2$. However, the mistake may have confused a student working through the details at a later moment.

What do we conclude from the above example? It is certainly useful to demonstrate some linear algebra calculations explicitly. Nevertheless, linear algebra should be a prerequisite for a course in linear programming. The details of linear algebra should not distract from the important issues in linear programming. These issues include:

- The choice of the pivot.
- What if the initial dictionary is not feasible?
- Duality theory.
- Efficiency considerations.

- Sensitivity analysis.
- Important special cases such as network problems.

Even for simple examples it is unnatural (and often impossible) to design them in such a way that the linear algebra calculations stay simple. A computerized tool for these calculations facilitates their demonstration. The benefits of such a tool are four-fold: no precious class-room time is wasted on elementary calculations, no calculation errors distract attention, the demonstration of larger, more interesting examples becomes feasible, and understanding of the simplex method is enhanced by showing the geometric interpretation of two-dimensional problems along executing the simplex method. The idea of having a tool for experimenting with the simplex method is, of course, not new. See, for example, [7, 8]. However, tools tend to gradually go out-of-date on newer platforms than those used to develop them. Distinctive features of our tool are: simple command line interface, geometric interpretation (so far for two variables), file input in standard LP format, portability, and it is free software as defined by [6].

This paper is organized as follows. In Section 2 we introduce our tool `pplex`, applied to several examples in Section 3. We conclude in Section 4.

2 A tool for teaching the Simplex method

Our contribution `pplex` [4], a *pedagogical implementation of the Simplex method*, is free software distributed under the GNU General Public License [5]. It runs under Java 6 and Java 7 and is portable to any supportive platform.

We start by demonstrating `pplex` on the example from the introduction. The input file for our example reads:

```
max               x  +   y
subject to       2x  +   y <= 6
                 7x  + 13y <= 40
```

Note that the inequalities $x, y \geq 0$ are implicitly assumed, and omitted in both the input file and the generated dictionaries.

The program `pplex` launches with the prompt `pplex>`. We read the above input file, which is confirmed OK:

```
pplex> read input/nik2.lps
Read input/nik2.lps OK.
```

To show the initial dictionary of this program one writes after the prompt:

```
pplex> show
 ζ =       +     x +      y
w1 =  6.00 - 2.00x -      y
w2 = 40.00 - 7.00x - 13.00y
```

Numbers are by default displayed with two decimals precision, whereas calculations are performed in double precision arithmetic. We can now let `pplex` execute a pivot by specifying the variable by its *column* (here 1 for x and 2 for y) and the linear equation by its *row*. Here comes the command and the resulting dictionary:

```
pplex> pivot 1 1
 ζ =  3.00 - 0.50w1 + 0.50y
 x =  3.00 - 0.50w1 - 0.50y
w2 = 19.00 + 3.50w1 - 9.50y
```

The final (optimal) dictionary is calculated after the next command:

```
pplex> pivot 2 2
 ζ = 4.00 - 0.32w1 - 0.05w2
 x = 2.00 - 0.68w1 + 0.05w2
 y = 2.00 + 0.37w1 - 0.11w2
```

Responding to the question of the student about starting with variable y, one can roll back to the first dictionary by two `undo`'s and pivot with y and the second linear equation:

```
pplex> undo

pplex> undo

pplex> pivot 2 2
 ζ = 3.08 + 0.46x - 0.08w2
w1 = 2.92 - 1.46x + 0.08w2
 y = 3.08 - 0.54x - 0.08w2
```

In an effortless way one now obtains the same final dictionary:

```
pplex> pivot 1 1
 ζ = 4.00 - 0.32w1 - 0.05w2
 x = 2.00 - 0.68w1 + 0.05w2
 y = 2.00 + 0.37w1 - 0.11w2
```

The lecturer may use the opportunity to call the students' attention to an important implementation issue: Numerically, exactly the same result is obtained by two different pivot sequences. Is that due to, or despite of the implementation using double precision arithmetic? In general, it is impossible to answer which of the two is the case. Higher precision means smaller rounding errors, but at the same time, smaller errors become visible.

3 Examples of Using `pplex`

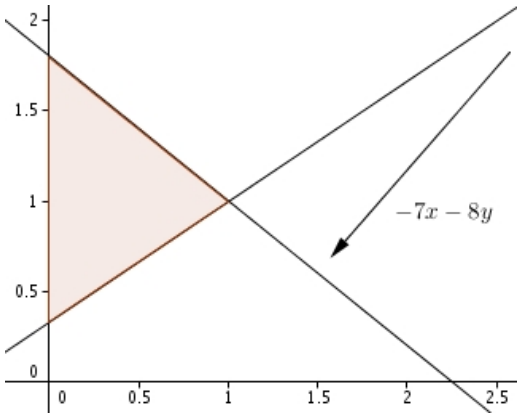
In this section we present examples using duality and examples illustrating degeneracy, cycling and unboundedness.

3.1 Duality

Giving an introduction into LP duality theory is beyond the scope of this paper. Therefore we assume knowledge of duality theory throughout this section.

In Sections 1 and 2, we studied an example of an initially feasible dictionary. What if the initial dictionary is not feasible, that is, if the origin is not a feasible solution? Here is an example:

```
max           -7x - 8y
subject to     2x - 3y <= -1
               4x + 5y <=  9
```



The feasible region of this program is the triangle with points $(0, \frac{1}{3}), (0, 1.8), (1, 1)$ in the x, y-plane, on which we maximize the objective $-7x - 8y$. Clearly, the maximum $-\frac{8}{3}$ is to be found in the point $(0, \frac{1}{3})$. However, we cannot proceed as in the previous section since the initial dictionary is not feasible:

```
pplex> read input/nik8.lps
Read input/nik8.lps OK.

pplex> show primal
 ζ =        - 7.00x - 8.00y
w1 = - 1.00 - 2.00x + 3.00y
w2 =   9.00 - 4.00x - 5.00y
```

One possible approach in such a case is to solve the dual instead:

```
pplex> show dual
-ξ =      +     y1 - 9.00y2
z1 = 7.00 + 2.00y1 + 4.00y2
z2 = 8.00 - 3.00y1 + 5.00y2
```

The negative coefficients of the original objective $-7x - 8y$ show up in the dual dictionary as the positive constants 7.00 and 8.00 of the first and the second linear equation, respectively. Since they are positive, the dual dictionary is feasible. The maximum value $\frac{8}{3}$ of $-\xi$ in the dual program is found after one pivot:

```
pplex> pivot dual 1 2
-ξ =  2.67 - 0.33z2 - 7.33y2
```

```
z1 = 12.33 - 0.67z2 + 7.33y2
y1 =  2.67 - 0.33z2 + 1.67y2
```

The primal version of this dictionary shows that we have indeed found the maximum $-\frac{8}{3}$ of ζ for $x = 0,\ y = \frac{1}{3}$ in the primal program:

```
pplex> show primal
 ζ = - 2.67 - 12.33x - 2.67w1
 y =   0.33 +  0.67x + 0.33w1
w2 =   7.33 -  7.33x - 1.67w1
```

In the last example, the dual dictionary was feasible. If both the primal and the dual dictionary are infeasible, the linear program is solved in two phases: In the first phase, we modify the linear program into one which *is* dually feasible, by substituting all coefficients in the objective with negative values. By solving the modified dual program we find a feasible solution of the original primal program. In the second phase, we solve the original primal program starting from the feasible solution found in the first phase. The two phases are demonstrated in `pplex` by the following example:

```
pplex> show primal
 ζ =        + 2.00x + y
w1 = - 1.00 -     x + y
w2 =   2.00 -     x - y
```

The objective is negated by `replace -2 -1`, yielding the dually feasible dictionary:

```
pplex> replace -2 -1
 ζ  =        - 2.00x - y
w1 = - 1.00 -     x + y
w2 =   2.00 -     x - y
```

The commands `pivot dual 1 2` and `show primal` yield the feasible final dictionary:

```
 ζ = - 1.00 - 3.00x - w1
 y =   1.00 +     x + w1
w2 =   1.00 - 2.00x - w1
```

This is now a mock solution since the original objective was $2x+y$, not $-2x-y$. Now the original objective must be reinstated, replacing basic variables by their right-hand sides: $2x + y = 2x + (1 + x + w_1)$. This is achieved by the command `reinstate`:

```
pplex> reinstate
 ζ = 1.00 + 3.00x + w1
 y = 1.00 +     x + w1
w2 = 1.00 - 2.00x - w1
```

It now takes only one step to find the maximum value $2\frac{1}{2}$ for $2x + y$ in $(\frac{1}{2}, 1\frac{1}{2})$:

```
pplex> pivot 1 2
 ζ = 2.50 - 1.50w2 - 0.50w1
 y = 1.50 - 0.50w2 + 0.50w1
 x = 0.50 - 0.50w2 - 0.50w1
```

3.2 Degeneracy and Cycling

If the constant of a pivot is zero, pivoting does not improve the value of the basic solution (*degenerated pivot*; geometrically, one stays at the same vertex of the polyhedron). Degenerated pivots occur commonly during execution of the simplex method. Usually, they are unproblematic, because one of the subsequent pivots improves the objective. Here is an example:

```
max               x  + 10y
subject to       2x  +   y <= 6
                  x  +  2y <= 6
                  x  +   y <= 4
```

The lines corresponding to these constraints intersect in the point $(2, 2)$. After `pivot 1 1` and `pivot 2 3` we continue as follows:

```
 ζ = 22,00 + 9,00w1 - 19,00w3
 x =  2,00 -     w1 +      w3
w2 =       -     w1 +  3,00w3
 y =  2,00 +     w1 -  2,00w3

pplex> pivot 1 2
 ζ = 22,00 - 9,00w2 + 8,00w3
 x =  2,00 +     w2 - 2,00w3
w1 =       -     w2 + 3,00w3
 y =  2,00 -     w2 +     w3

pplex> pivot 2 1
 ζ = 30,00 - 5,00w2 - 4,00x
w3 =  1,00 + 0,50w2 - 0,50x
w1 =  3,00 + 0,50w2 - 1,50x
 y =  3,00 - 0,50w2 - 0,50x
```

`pivot 1 2` is degenerated and the objective does not improve. In the geometric interpretation, we stay in $(2, 2)$. Before the degenerated pivot, this point is the intersection of the two lines $w_1 = w_3 = 0$. After the degenerated pivot this same point is the intersection of the two lines $w_2 = w_3 = 0$. The essential difference

in the two dictionaries is visualized by depicting lines corresponding to a *basic variable being zero* in COLOR.

In the above example, a degenerated pivot can easily be avoided: Degeneracy does not occcur if we start by increasing the variable y, which has the largest coefficient. Also, `pivot 2 2` instead of `pivot 2 3` avoids degeneracy (not in the dictionary, though). In class, one can now address degeneracy and cycling in general. The standard example [3, 2] of the (rare) possibility of cycling with the largest-coefficient-rule can be demonstrated effortlessly using `pplex/input/cycling.lps`.

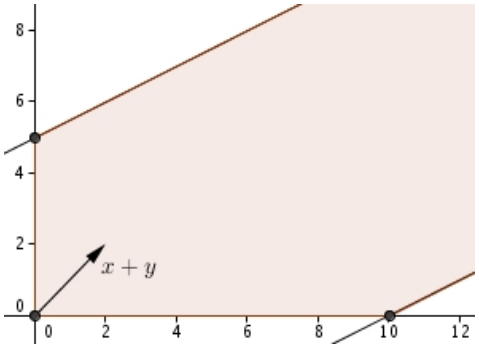
3.3 Unboundedness

If the polyhedron defined by the constraints is *unbounded* in a direction in which the objective increases, the linear program has no solution since the objective can take arbitrarily large values:

```
pplex> read input/nik6.lps
Read input/nik6.lps OK.

pplex> show primal
 ζ =       + x +     y
w1 = 10.00 - x + 2.00y
w2 = 10.00 + x - 2.00y

pplex> pivot
 ζ = 10.00 - w1 + 3.00y
 x = 10.00 - w1 + 2.00y
w2 = 20.00 - w1
```



This dictionary is feasible and all coefficients of y are non-negative. This means that y can increase unboundedly, yielding arbitrarily high values for the objective. The command `pivot`, which can pick a suitable pivot itself, discovers this:

```
pplex> pivot
Program is unbounded.
```

3.4 Introduction into Sensitivity Analysis

Once an optimum is found, it is natural to ask how sensitive it is to the accuracy of the available data. As an introduction into sensitivity analysis, the lecturer may for example ask the students: How much can we fiddle with the coefficients in the objective function without changing the optimal solution? Students can use `pplex` to experiment with the optimal dictionary. Consider for example the following variation of our first example, where the objective and the first inequality are almost parallel:

```
pplex> read input/nik9.lps
```

```
Read input/nik9.lps OK.

pplex> show
 ζ =       + 2.01x +      y
w1 =  6.00 - 2.00x -      y
w2 = 40.00 - 7.00x - 13.00y

pplex> pivot
 ζ =  6.03 - 1.01w1 - 0.00y
 x =  3.00 - 0.50w1 - 0.50y
w2 = 19.00 + 3.50w1 - 9.50y
```

The optimal solution is not longer (2,2), but (3,0). Note the rounding error in the coefficients of w_1 and y in the objective. Pedagogically, an ideal occasion to introduce another feature of `pplex`:

```
pplex> show primal 3
 ζ =  6.030 - 1.005w1 - 0.005y
 x =  3.000 - 0.500w1 - 0.500y
w2 = 19.000 + 3.500w1 - 9.500y
```

Increasing the precision to three decimals by the command `show primal 3` eliminates the rounding error in this example.

4 Classroom Experience and Conclusion

The benefits of `pplex` are four-fold: no precious class-room time is wasted on elementary calculations, no calculation errors distract attention, the demonstration of larger, more interesting examples becomes feasible, and understanding of the simplex method is enhanced by showing the geometric interpretation of two-dimensional problems along executing the simplex method. There is also a danger associated with the use of such tools in teaching: one should make sure that the students really understand the elementary calculations the tool is performing.

So far we have only experience with using `pplex` in one course with 11 students ...

References

[1] Computing in Science and Engineering, volume 2, no. 1, 2000.

[2] R.J. Vanderbei, *Linear Programming, Foundations and Extensions*, 3rd edition, Kluwer, 2008.

[3] V. Chvátal, *Linear Programming*, W.H. Freeman and Company, 1983.

[4] `https://github.com/andern/lpped/`

[5] http://www.gnu.org/licenses/

[6] http://www.gnu.org/philosophy/free-sw.html

[7] http://campuscgi.princeton.edu/~rvdb/JAVA/pivot/simple.html

[8] http://campuscgi.princeton.edu/~rvdb/JAVA/pivot/advanced.html

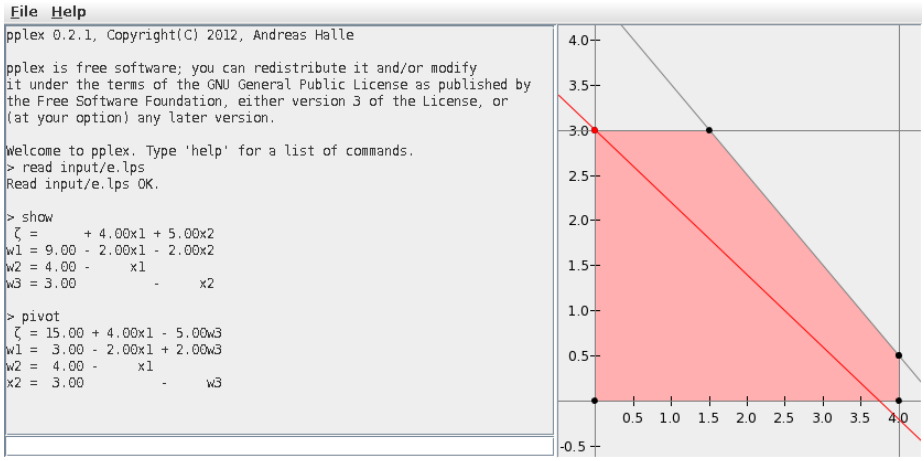


Figure 2: Example of the Graphical User Interface