International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

# SIMPLEX v. 2.17: an Implementation of the Simplex Algorithm for Large Scale Linear Problems. User's Guide

**Swietanowski, A.**

**IIASA Working Paper**

**WP-94-037**

**May 1994**

# Working Paper

## SIMPLEX v. 2.17: an Implementation of the Simplex Algorithm for Large Scale Linear Problems. User's Guide.

*Artur Święta nowski*

WP-94-37
May 1994

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 71521 □ Telex: 079 137 iiasa a □ Telefax: +43 2236 71313

# SIMPLEX v. 2.17: an Implementation of the Simplex Algorithm for Large Scale Linear Problems. User's Guide.

*Artur Świętanowski*

WP-94-37
May 1994



International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 71521 □ Telex: 079 137 iiasa a □ Telefax: +43 2236 71313

# Foreword

The research described in this Working Paper was performed at the Institute of Automatic Control, Warsaw University of Technology as part of IIASA CSA activities on *Methodology and Techniques of Decision Analysis.* While earlier work within this project resulted in the elaboration of prototype decision support systems (DSS) for various models, these systems were closed in their architecture. In order to spread the scope of potential applications and to increase the ability to meet specific needs of users, in particular in various IIASA projects, there is a need to modularize the architecture of such DSS. A modular DSS consists of a collection of tools rather than one closed system, thus allowing the user to carry out various and problem-specific analyses.

This Working Paper describes the SIMPLEX optimization solver for middle and large-size linear programming problems based on the modified simplex algorithm. SIMPLEX can be used for solving the relaxed mixed integer programming problem and thus providing an advanced basis for the MOMIP solver (documented in WP-94-35). Both solvers are designed and implemented as part of a wider linear programming library being developed within the project.

# Abstract

This document presents **SIMPLEX** - a linear optimizer for large scale linear programs. Issues like command line syntax, data input and output formats and interpretation of results are addressed. In order to make the interpretation of results and program's reports easier, some of the features of the implementation are also discussed. Finally, numerical test results give a measure of the code's true efficiency.

# Contents


**1 Introduction** 1

**2 Modifications to the standard algorithm** 1
- 2.1 Formulation of the problem . . . 2
- 2.2 Linear problem scaling . . . 2
- 2.3 Construction of a feasible initial basis . . . 3
- 2.4 Pricing and pivoting . . . 3
- 2.5 Dynamic numerical tolerances . . . 4
- 2.6 Detecting optimality and infeasibility . . . 4
- 2.7 Overcoming numerical difficulties . . . 5

**3 Using the program** 5
- 3.1 Command line syntax . . . 6
- 3.2 Solver report file . . . 8

**4 Input and output file formats** 11
- 4.1 Text input file format - fixed MPS . . . 11
- 4.2 Binary input and output using LP-DIT data transfer protocol . . . 12
- 4.3 Text solution file format . . . 12
- 4.4 Final (optimal) basis file format . . . 13

**5 Numerical results** 14

**6 Conclusions** 15

**7 Acknowledgements** 15

**A Compiling and linking SIMPLEX** 15
- A.1 Data types . . . 15
- A.2 Compile time reconfiguration . . . 16
- A.3 Compiling the solver on a PC–based computer . . . 17

# SIMPLEX v. 2.17: an Implementation of the Simplex Algorithm for Large Scale Linear Problems. User's Guide. *

*Artur Świętanowski***

## 1 Introduction

SIMPLEX is an experimental implementation of Dantzig's [3] revised simplex method for large scale linear programs. It contains quite a few of the enhancements typically available in commercial linear programming packages. Those include sparse inversion routines, scaling, crashing, non-standard pricing and pivoting techniques etc.

Section 2 gives a brief overview of those features with focus on their impact on program's reports' interpretation. Section 3 explains command line syntax and describes the typical program run, including the on-line activity report. Section 4 provides the reader with short descriptions of available input file formats (a subset of IBM's MPS standard[1] and the format proposed for LP-DIT[2]) and a more detailed definition of output file formats. Since it is possible for the solver to supply the user with the final (possibly optimal) basis, the format of the file containing the basis description is also covered in that section. Results of numerical experiments conducted so far, are presented in section 5. They are immediately followed by final conclusions in section 6. Appendix A explains how to compile and link the program on a computer, on which the software was not already tried, and how to change some of the operating characteristics to suit user's particular needs. Obviously, this appendix may be useful only for users who have access to source codes.

Throughout the document, it is assumed that the reader is familiar with most of the theoretical issues presented. Therefore, we will not provide a detailed description of the simplex method. Nor will we present lengthy discussions why one technique was used instead of another. Readers interested in such in-depth discussions are referred to Świętanowski [15] as well as to other papers listed in the references.

## 2 Modifications to the standard algorithm

The simplex method is very well understood and documented (see e.g.: Dantzig [3] and Murtagh [13]) and there exist several very efficient commercial implementations (as e.g. CPLEX of Bixby [2], OSL of Forrest and Tomlin [4] and MINOS ver 5.3 of Murtagh and Saunders [14]).

*This research was partially sponsored by the grant number P40301806 of the Polish Comitee for Scientific Research.

**Institute of Control and Computation Engineering, Faculty of Electroncs and Information Technology, Warsaw Uniwersity of Technology, ul. Nowowiejska 15/19, Warsaw.

[1] See e.g. [11] or [13] for more information. MPS standard description is also available in numerous other sources.

[2] See Makowski [12] for details.

The key to its effectiveness and ability to cope with numerical difficulties lies in the numerous enhancements and improvements that have been invented throughout the years by many researchers. While the basic Dantzig's idea and algorithm remain unchanged, the various tricks make the difference between a good and a poor simplex method implementation.

Quite a few of those tricks were used in SIMPLEX. Some of them are commonly known, but others are not as popular. As the optimization progresses, the program usually (see section 3) produces a report of its activity, which refers not only to user's understanding of the simplex method, but also to his (or her) knowledge of some details of this implementation. In order to make this report informative to the reader, it is recommended that he (or she) should at least leaf through this section and get acquainted with the terms used.

## 2.1 Formulation of the problem

We are concerned with a linear problem of minimizing objective function

$$\min \ \mathbf{c}^T \mathbf{x}$$

subject to constraints

$$\begin{array}{rcccl} & & a_{i_E}^T \mathbf{x} & = & b_{i_E} \\ & & a_{i_L}^T \mathbf{x} & \leq & b_{i_L} \\ & & a_{i_G}^T \mathbf{x} & \geq & b_{i_G} \\ b_{i_R} & \leq & a_{i_R}^T \mathbf{x} & \leq & b_{i_R} + r_{i_R} \\ \mathbf{l} & \leq & \mathbf{x} & \leq & \mathbf{u} \end{array}$$

where $\mathbf{c}$, $\mathbf{x} \in \Re^n$, $\mathbf{b}$, $\mathbf{r} \in \Re^m$ and $a_{i_E}$, $a_{i_L}$, $a_{i_G}$, $a_{i_R}$ ($i_E = 1, \ldots, m_E$, $i_L = 1, \ldots, m_L$, $i_G = 1, \ldots, m_G$, $i_R = 1, \ldots, m_R$, $m_E + m_L + m_G + m_R = m$) are respectively "equal to", "less than", "greater than" and "range" rows of the constraint matrix[3] $\mathbf{A}$, $\mathbf{A} \in \Re^{m \times n}$.

Depending on the values of simple bounds on primal variables $x_j$, $j \in \{1, \ldots, n\}$ each of them may belong into one of the following four categories:

- free variable ($l_j = -\infty$ and $u_j = +\infty$),
- non-negative variable ($l_j > -\infty$ and $u_j = +\infty$),
- bounded variable ($l_j > -\infty$ and $u_j < +\infty$) and
- fixed variable ($-\infty < l_j = u_j < +\infty$).

## 2.2 Linear problem scaling

It is common practice to scale a linear problem in order to improve its numerical properties. There exist many different definitions of a well (or badly) scaled problem. There are also quite a few different scaling schemes (see e.g. Tomlin [16]). Our approach is a simple one: we consider a problem well scaled if the absolute values of non-zeroes in each row and each column of the constraint matrix are distributed evenly around unity. To this end we find the smallest and the largest non-zero (with respect to absolute value) of each matrix row (or column) and divide this row (or column) by its geometric average.

We define a scaling quality estimate $c_{\mathbf{A}}$ for the constraint matrix $\mathbf{A}$ as:

$$c_{\mathbf{A}} = \log_{10} \frac{\max_{i,j} |a_{i,j}|}{\min_{i,j} |a_{i,j}|}$$

where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. We consider the matrix well scaled if $c_{\mathbf{A}} < 2$.

[3] This roughly corresponds to the format allowed by MPS standard for linear problem formulation – see section 4.

If the constraint matrix $\mathbf{A}$ is well scaled, scaling is not performed. Otherwise two consecutive passes of row and column scaling are performed. Scaling factors are computed so as not to introduce any roundoff errors. To this end we compute "ideal" scaling factors and then use their approximation by an integral power of two (for details see [15]).

## 2.3 Construction of a feasible initial basis

After converting the problem to standard (equality) form, there's a need to find an initial solution to the problem. From now on, $\mathbf{A}$ will refer to the constraint matrix of the problem in standard form (i.e. with slack variables added).

The algorithm used for initial basis construction is a variant of Bixby's [2] idea of dividing the variables into so-called preference sets, and then building the basis using as many columns of the original constraint matrix as possible. Our algorithm is simplified by requirement that a (permuted) triangular basis always has to be found. Since our basis inversion routines are not capable of recovery from numerical difficulties, the basis has to be invertible. For that reason, a numerical tolerance for initial basis pivot is also introduced. When the process is finished without producing a complete basis, the missing places are filled with unity columns and their corresponding artificial variables are added.

A solution $\mathbf{x_N} = \mathbf{0}$, $\mathbf{x_B} = \mathbf{B^{-1}b}$ obtained with this basis $\mathbf{B}$ is not necessarily feasible. We deal with this problem by modifying the initial problem to one, in which the basis we just found would provide a feasible initial solution. The new problem takes the following form:

$$\min \mathbf{c}^T\mathbf{x} + Mt$$

subject to constraints

$$\begin{aligned} \mathbf{Ax} + \lambda t &= \mathbf{b} \\ \mathbf{l} \leq \mathbf{x} &\leq \mathbf{u} \\ 0 \leq t &\leq 1 \end{aligned}$$

where $\lambda$ denotes a column for which the solution $\mathbf{x_B} = \mathbf{B^{-1}b}$, $\mathbf{x_N} = \mathbf{0}$ and $t = 1$ is feasible (see section 2.6 for details of recovering the optimal solution of the original problem from the solution of the modified problem). $M$ denotes a large positive penalty for non-zero values of the so-called super-artificial variable $t$. It is obvious that any infeasibility of the initial basic solution found by the crashing algorithm may be disposed of by adding a single column $\lambda$, which is computed as

$$\lambda = \mathbf{b} - \mathbf{Ax} = \mathbf{b} - \mathbf{Bx_B} - \mathbf{Nx_N} = \mathbf{b} - \mathbf{Bx_B}$$

and thus may also be called a signed residual vector.

Some efficiency considerations (see [15]) have led us to believe that it is advantageous to create a single super-artificial variable and its corresponding constraint matrix column for each row infeasibility (represented by a non-zero in the $\lambda$ vector above). As you will see later, the number of row infeasibilities is reported by the program as a number of super-artificial variables.

## 2.4 Pricing and pivoting

Pricing and pivoting schemes used are fairly standard. We use a certain combination of partial and multiple pricing which has a nice property of being as effective (in terms of total number of iterations) as full pricing, but nearly five times faster. This is achieved by remembering a few "next best" candidates from the previous iteration and checking their reduced costs in addition to the reduced costs of the columns belonging to the part of the constraint matrix that is to be scanned in current iteration. The reduced costs are not updated – both they and the dual variables are computed anew at each iteration[4].

[4] Future releases of **SIMPLEX** will most likely employ either Goldfarb's [7] steepest edge algorithm, or at least some kind of reduced costs updating technique.

During pivoting we employ Hariss' [10] technique for finding a numerically stable pivot that would not violate feasibility by more than a prescribed feasibility tolerance.

## 2.5 Dynamic numerical tolerances

Our algorithm is equipped with a set of numerical tolerances, that are adjusted on the run as the numerical stability of the consecutive simplex bases changes. The tolerances include:

- **Zero tolerance** $\delta_S$. Numbers with absolute value not larger than $\delta_S$ are considered zero.
- **Feasibility tolerance** $\delta_F$. We allow the solution to violate the simple bounds on variables by no more than $\delta_F$.
- **Optimality tolerance** $\delta_O$. We consider a move in direction $\eta$ profitable only if the reduced cost $\mathbf{c}^T\eta$ for this direction is smaller than $-\delta_O$.
- **Pivot tolerance** $\delta_P$. During a search for a basic column about to leave the basis, we consider only pivots with absolute values larger than $\delta_P$ to be actually non-zero (and thus eligible pivots).
- **Inverse pivot tolerance** $\mu$. A non-zero is eligible to become a pivot in the basis inverse representation only if it is equal at least $\mu$ times the absolute value of the largest non-zero of its row.
- **Minimum refactorization frequency** $f_{R\,min}$. The basis is inverted anew after at most $f_{R\,min}$ updates.
- **Maximum non-zero growth factor** $N_{GROWTH}$. The basis is reinverted if the largest absolute value of its factors exceeds $N_{GROWTH}$ times corresponding value from the latest factorization.
- **Maximum non-zero number factor** $N_{NUMBER}$. The basis is reinverted if the total length (number of non-zeros) of its factors exceeds $N_{NUMBER}$ times their number after the latest factorization.

The values of $N_{NUMBER}$ and $N_{GROWTH}$ remain unchanged throughout all iterations of the algorithm. Other values are subject to change whenever some serious numerical difficulties arise, or – conversely – the problem appears to be easier than at some earlier stage. When the program detects it is close to optimum, very strict numerical tolerances are taken in order to provide the user with an accurate solution.

To allow a reasonable level of control over the dynamic tolerances, they are all adjusted together, which is usually reported by the program – see section 3.

## 2.6 Detecting optimality and infeasibility

Our technique for obtaining the initial feasible basis is paid for by a certain complication of the method used for detecting optimality. As noted in section 2.3, the modified problem either is unbounded or has a feasible optimal solution. We may say that the modified problem is a relaxed version of the original one, therefore its unboundedness may still be detected in the standard manner and will not be the subject of this section. Indeed, we will (until the end of the current section) assume that the original problem is either bounded or infeasible.

When all artificial variables are equal to zero, an optimal solution to the modified problem (see section 2.3 for the modified problem formulation) is also an optimal solution to the original problem. If, however, some of them should be non-zero, it may mean that either the penalty $M$ for their non-zero value is too small, or the original problem is infeasible. To decide what the actual reason is for the non-zero values of those variables, we increase the penalty $M$ tenfold.

If the solution still appears to be optimal after ten such operations (i.e. after an increase of $M$ by $10^{10}$), we report infeasibility of the original problem. Otherwise we continue to solve the problem with increased penalty. As experience with the NETLIB test problems[5] has shown, the value of penalty $M$ we choose initially is sufficient in almost all cases.

### 2.7 Overcoming numerical difficulties

While most of the test problems that we have worked with so far are solved without encountering numerical difficulties, we found it more than worthwhile to introduce some precautionary measures as well as a few strategies aimed at solving numerically difficult problems. The precautionary measures are:

- periodical re-inversion of the basis,
- using threshold pivot tolerance $\mu$ in inversion routines,
- using pivot tolerance $\delta_P$ during search for the column leaving basis and
- monitoring the growth of number and size of non-zeros in current basis inverse representation.

Numerical difficulties which have not yet manifested themselves by producing a singular basis, may still be detected by periodically checking the primal and dual residuals and primal infeasibility. Large residuals cause the program to assume the problem is numerically difficult and adjust tolerances accordingly[6].

Even bigger residuals will eventually lead to backing out of a number of iterations[7]. The purpose of such an action will be to go back to a basis which is presumed to be numerically stable. The columns introduced into the basis between that stable basis and the moment when the difficulties were detected are marked as dangerous and are not allowed to go back into the basis until they are the only ones with favorable reduced costs. If – despite the precautions – a basis singularity occurs, we also back out of some of the last iterations (as described above).

## 3 Using the program

The simplex algorithm described (with its modifications) in section 2 of this paper has been implemented as a large scale, sparse linear problem optimizer `SIMPLEX`.

The program is primarily designed to work under control of the UNIX System V operating system. However only the time measuring functions are dependent on the operating system function calls. Thus porting the program to e.g. VMS environment should not pose any problem.

It is also possible to compile and run the program in the DOS operating system environment with DOS version 3.30 or later. As PC-based computers are not particularly well suited for truly large scale computations, there are many limitations when `SIMPLEX` is run on a PC equipped with a DOS system. For example, its time measuring functions and binary input in LP-DIT format are not available. For more details on this topic consult appendix A.3.

The program accepts a fixed MPS format input file in which the linear problem is defined (see also section 4). Current version solves only minimization problems. Alternatively, a binary input file in LP-DIT format may be used[8]. At runtime the program may generate a report of its activity and store it either in a log file, or display it on the screen, or do both.

[5] A collection of standard test problems for linear programming codes' evaluation was proposed by Gay [5]. Currently a so called NETLIB test collection is available by `ftp` from `research.att.com` when using anonymous user account `netlib`.

[6] Appropriate message is put in the report file, or on the screen, or both.

[7] Program reports such an action using term "`backtracking`".

[8] For more information on LP-DIT's origin and purpose as well as it's operating characteristics and specifications, see [12]. A brief description may also be found in section 4.2.

If an optimal solution to the problem is found or the problem is deemed unbounded or infeasible, a file with the solution may be created. The file will contain the optimal solution found, or the primal and dual variables and reduced costs at the moment, where the infeasibility or unboundedness was detected. The file may either be stored in text form or in LP-DIT format. On request, the optimal basis may be output for use by other programs. The basis output format will be described in section 4.4.

In case the program is unable to find a solution and stops for another reason, the solution file is not created. There are three situations when the program may stop without arriving at any solution to the problem:

1. numerical difficulties were encountered that could not be overcome,
2. the iteration limit for the current problem was exceeded or,
3. a severe internal error has occurred in the program.

The last two reasons why the program may be aborted require an explanation. Since the program is essentially an experimental code, we had to take into account that in some cases it may fail to provide a solution. It is therefore equipped with a mechanism for breaking the computation process when stalling occurs. If an iteration limit of $C(m+n)$ (where $m$ and $n$ denote number of binding rows of the constraint matrix and the number of structural variables respectively, and $C$ denotes a positive constant number specified by a user in one of the configuration files – see appendix A) is reached, the program aborts. It is also possible for the program to try to detect the results of some programming errors and as soon as such an error is detected, break the computations with an appropriate message. For more information on iteration limit setting, self debugging and related issues, see appendix A.

## 3.1 Command line syntax

The command line syntax of the SIMPLEX program is:

`simplex [<options>] <input_file> [ -b [<basis_file>] ]`

Currently available options are:

1. `-v <verb>`

   This option allows one to set the desired report verbosity level (see section 3.2 for the meaning of the verbosity levels as well as report file formats).

   **Recognized values of `<verb>`:** `none`, `line`, `low`, `high`.

2. `-t <bin_solution_file>`

   `<bin_solution_file>` specifies a name of a binary (LP-DIT format) solution file to be created. Since LP-DIT functions perform random disk reads and writes, stream output is not possible.

3. `-s <text_solution_file>`

   Depending on the value of `<text_solution_file>` the solution will be output to standard output stream or to a named file.

   **Recognized values of `<solution_file>`:**

   - `file_name` Solution will be written to a file called `file_name`. The program does not check if such a file already exists. In case the file cannot be opened, the program will terminate with an error message.
   - = Solution will be output to the standard output stream.

4. `-r <report_file>`

   At rutime the program may inform the user about the current state of computations by printing a report on the standard output (typically a screen) or to a file. The length and detail of the report depends on the setting of verbosity level (see point 1 above).

   The `-r` option tells the program to store the report in a file named `<report_file>` in addition to printing it on the screen.

   **Attention:** If verbosity level "none" is specified, this option is meaningless, since no report will be produced.

5. `-b` (appearing before `<input_file>`)

   This option takes no arguments. It allows you to turn off report output to the standard output stream ("b" stands for "blank"). This may be useful when the program is run in a batch (or in background) with all output directed to files.

   **Attention:** If verbosity level "none" is specified, this option is meaningless, since no report will be produced.

6. `-dit`

   If this switch is specified, the `<input_file>` is assumed to be an LP-DIT binary file. Fixed MPS format file is expected otherwise.

7. `-b` (appearing after `<input_file>`)

   The optional last argument `-b [<basis_file>]` may be used to indicate to the program that – if an optimal solution is found – the optimal basis should be output to a file.

   **Recognized values of `<basis_file>`:**

   If you specify `-b` = the basis file is output to the standard output stream. Otherwise the basis is stored in a file named `<basis_file>`.

   **Default value of `<basis_file>`:**

   If `<basis_file>` name is omitted, a name built of the problem name (e.g. as found in `NAME` section of the MPS input file) and "`.bas`" extension is assumed. If the option is altogether omitted, basis is not output at all.

   If the program is run in DOS environment period (".") characters in problem names are converted to underlines ("_").

Parameters `<verb>`, `<bin_solution_file>`, `<text_solution_file>`, `<report_file>` and `<basis_file>` may be optionally separated from their option specifiers (respectively `-v`, `-s`, `-t`, `-r`, `-b`) by whitespace. Character case in options is insignificant (except for file names, which are taken literally). Options may be given in any order (except, of course, the `-b` option), but may not be repeated. Repeated option is treated as a serious error and causes the program to be aborted.

By default:

- verbosity level is `low`,
- a fixed MPS format file is read,
- a binary solution file is not created,
- a text solution file is not created,
- a report is generated and output to the screen and
- the optimal basis is not output.

At least one input parameter (<input_file>) is required. It may either be the name of an input file (see also section 4), or "=" which tells the program to read input from its standard input stream. As was noted before, only MPS file may be read from a stream.

Additional arguments following the part of the list that was successfully parsed, are considered a serious error. In case of any error in the argument list, the program will produce appropriate error messages and a concise usage note (including command line syntax) and terminate.

Here are some examples of command lines together with their interpretation:

`simplex -s afiro.sol afiro`

| | |
|---|---|
| **Input file:** | Fixed MPS read from file `afiro`. |
| **Report verbosity:** | Set to default `low`. |
| **Report file:** | Written only to standard output. |
| **Solution file:** | Written to text file `afiro.sol`. |
| **Basis file:** | Not created. |

`zcat afiro.Z | simplex -t afiro.sol =`

| | |
|---|---|
| **Input file:** | Fixed MPS read from standard input (by a pipe). |
| **Report verbosity:** | Set to default `low`. |
| **Report file:** | Written only to standard output. |
| **Solution file:** | Written to binary file `afiro.sol`. |
| **Basis file:** | Not created. |

`simplex -vHigh -s solution -r report -b afiro`

| | |
|---|---|
| **Input file:** | Fixed MPS read from file `afiro`. |
| **Report verbosity:** | Set to `high`. |
| **Report file:** | Written only to file `report`. |
| **Solution file:** | Written to a text file named `solution`. |
| **Basis file:** | Not created. |

`simplex -t _sol -dit afiro -b afiro.bas`

| | |
|---|---|
| **Input file:** | LP-DIT format read from file `afiro`. |
| **Report verbosity:** | Set to default `low`. |
| **Report file:** | Written only to standard output. |
| **Solution file:** | Text file not created, binary file `_sol` written. |
| **Basis file:** | Written to a file named `afiro.bas`. |

## 3.2 Solver report file

Depending on the verbosity level, the program generates a report in one of three possible formats. Each type of report serves a different purpose. A report will not be generated at all if verbosity level is set to `none`. In all other cases, a report will be produced. For information on setting verbosity level see section 3.1.

Figure 1: Example of **SIMPLEX** report in `line` verbosity mode.

```
afiro      |    27|    32|      83|    9.606|1.356E+00|Unscaled!|
    27|      0|      0|      13| -4.647531428571E+02|
    0.05|    0.02|    0.00|    0.05|    0.12
```

When verbosity level is set to `line`, the whole report will be printed in one line with all information aligned to form a single line of a table. This form of report is useful when a series of

Table 1: Contents of the report line in `line` verbosity mode.

| Category | Columns | Meaning |
|---|---|---|
| LP problem statistics | 1 – 10 | linear problem name |
| | 12 – 17 | number of constraint matrix rows |
| | 19 – 24 | number of constraint matrix columns |
| | 26 – 33 | number of constraint matrix non-zeros |
| | 35 – 43 | constraint matrix density (in per cent) |
| Scaling quality | 45 – 53 | scaling quality estimate before scaling |
| | 55 – 63 | scaling quality estimate after scaling |
| Initial basis quality | 65 – 70 | number of original columns used |
| | 72 – 77 | number of artificial columns |
| | 79 – 84 | number of infeasibilities |
| Solution | 86 – 93 | total number of iterations |
| | 95 – 114 | optimal value of the objective function |
| Time spent in consecutive parts of the program | 116 – 123 | reading the MPS file |
| | 125 – 132 | conversion to standard form and scaling |
| | 134 – 141 | initial basis construction |
| | 143 – 150 | solution (simplex iterations) |
| | 152 – 159 | total execution time |

test problems is being optimized and a table of results will be needed. Each solved problem will add one row to the table. After manual addition of a table header, the table is ready. Figure 1 presents an example of one line report for NETLIB problem `afiro`[9].

Table 1 describes the contents of a single line report. As it can easily be seen, only the most essential information about the problem and its solution is presented. If the problem was not solved, or was found infeasible or unbounded, one of the following messages will appear in place of the objective function value: `Unsolved!`, `Stalled!`, `Infeasible!` or `Unbounded!`.

Figure 2: Various messages produced by `SIMPLEX` in `low` verbosity mode.

```
Numerical difficulties: successful refactorization.
Numerical difficulties: refactorization failed.
Column retry!
Iter:      100    Result:  3.79E-02 Infeasibility    =   3.15E-04
Iter:      100    Result:  3.79E-02 Primal Residuals =   3.15E-04
Iter:      100    Result:  3.79E-02 Dual Residuals   =   3.15E-04
Iter:      100    Result:  3.79E-02 Tolerances for harder LP
Iter:      100    Result:  3.79E-02 Tolerances for easier LP
Iter:      100    Result:  3.79E-02 Taking final tolerances
```

A short report (output when verbosity level is set to `low`) is the default. It provides the user with essentially the same information as the single line report does, but in a much more readable form. The report begins with the header information, which presents the program. Next, an input file format and a linear problem name are given. Next, information about the results of scaling and crashing are shown. The main part reports all numerical difficulties encountered and how they were handled. It also informs about all adjustments to numerical tolerances. In the best case scenario (that is in case of easy problems), this part may consist of one line stating

[9] As the output line is over 150 characters long it can only be printed here broken into three lines.

Figure 3: Example of output produced by SIMPLEX in low verbosity mode.

```
------------------------------------------------------------------------
SIMPLEX v. 2.17 (c) 1992, 1993 Artur Swietanowski
Developed in Warsaw University of Technology, Institute of Automatic Control
------------------------------------------------------------------------
Reading input file in fixed MPS format.
Linear problem name is AFIRO.

Init cond. = 1.356E+00 No scaling!
Cols:     27 orig.        0 artif.      2 sup.-artif.
Iter:       11 Result:  -4.44E+02 Taking final tolerances

Optimum reached.   Iter:       11 Result:  -4.647531428571E+02

Times:   Reading LP  To standard  Init. basis    Solution     Total
               0.05         0.02         0.02        0.05      0.13
```

that the final tolerances are taken. Finally, the time spent in different parts of the program are given[10].

Figure 2 presents all possible messages that you can encounter in the main part of this type of SIMPLEX activity report. Whenever residuals trigger some kind of alarm, or when tolerances are adjusted (which quite often goes together — see sections 2.5 and 2.7 for more details) the user is also provided with current value of objective function and the number of simplex iteration.

Figure 3 presents a full report in low verbosity mode. It is a report from the solution of a linear problem afiro from the NETLIB test problem collection. The first line under the header informs the user what is the value of the initial scaling quality estimate. The next line says that the initial basis was built using 27 original columns with no artificial columns added. The initial basis found was infeasible (2 super-artificial columns). Since it is a very simple problem, no difficulties were encountered and after 11 iterations, the problem was solved. Then follows the solution time broken up into four stages.

The fullest report possible is produced in high verbosity mode. It contains all the information presented by the short report presented before, additionally some problem statistics and – what's more important – a report line for each simplex iteration. Figure 4 presents an example of high verbosity mode report for the same NETLIB problem afiro. The information not seen in the previous example includes:

- statistics of the linear problem constraint matrix:
  - number of rows,
  - number of columns,
  - number of non-zeros and
  - density in per cent.
- statistics for constraint matrix rows:
  - number of rows with ranges[11],

[10] All times were measured using **times()** system function specific to UNIX System V environment. If the program is compiled and run under control of a different operating system, time measuring will not be performed and this part of report will be absent. For more details on time measuring in different operating systems see appendix A.

[11] Each range row is counted both here and in the count of rows of the type corresponding to its type.

  - number of equality rows,
  - number of "less than" rows and
  - number of "greater than" rows.

- statistics for constraint matrix variables:
  - number of fixed variables,
  - number of free variables,
  - number of bounded variables and
  - number of normal variables.

- information about each iteration:
  - number of iteration `Iter`,
  - reduced cost of the column entering the basis `RC`,
  - length of the step made in the direction implied by the entering column `STEP`,
  - number of the entering column `C IN`,
  - number of the column leaving the basis `C OUT` (this field may be empty during iteration, in which a bounded variable moves between its bounds and the basis is not changed),
  - space between the two last columns, in which an asterisk may appear to mark the iteration in which the basis was re-inverted,
  - the current objective function value (`COST`).

The messages signaling numerical difficulties are similar to those of the short report, only that there is no need to prefix them with the iteration number and the current objective function value.

# 4 Input and output file formats

This section deals with formats of input and output files of `SIMPLEX`. While the text input file format conforms to a well known standard, that does not need to be described in detail, the output files require a full description. The following sub-sections will provide the reader with all information necessary to produce input files as well as interpret the program output.

## 4.1 Text input file format - fixed MPS

Fixed MPS file is so well known, that we will only list the simplifying assumptions we have taken when implementing source file reading. Readers not familiar with the MPS file format are referred to book [14] by Murtagh, as well as to description of IBM's MPSX/370 linear algebra package [11].

A list of main differences between the MPS fixed format described there and the format `SIMPLEX` actually recognizes follows:

- all characters that stay outside of the fixed length fields are ignored without a warning,
- all labels in `COLUMNS` section are required,
- vector labels in `RHS`, `BOUNDS` and `RANGES` sections are ignored (consequently presence of only one vector in each of those sections is assumed),
- comments are not recognized (except full line comments starting with an asterisk "*" character in the line's first column),

- first free row is considered to represent the objective function, others are ignored,
- rows that are linear combinations of some other rows are not recognized,
- BV is recognized in bounds section and is interpreted as "binary variable" with lower bound of 0 and upper of 1,
- RHS section is optional (as it may be empty),
- negative values in RANGES section are converted to positive absolute values without a warning.

## 4.2 Binary input and output using LP-DIT data transfer protocol

Presently, one of the major uses of the SIMPLEX optimizer is solving of linear relaxations of mixed integer problems generated in binary form and passed to solver via LP-DIT protocols[12]. Format of the binary files is not a part of LP-DIT specification, which only defines access methods.

Whenever required, the linear problem is read from a named binary file (see also section 3.1 on command line syntax). This form of problem input is significantly faster and more reliable – it does not depend on text parsing and conversion to numerical values. Note, however, that two identical problems, one read in LP-DIT format and one read in MPS format will have slightly different optimal objective values and – in case of larger and more difficult problems – SIMPLEX may follow a different path towards the optimum. These slight differences result directly from roundoff errors which are caused by the fact, that LP-DIT passes floating point numbers in single precision, and SIMPLEX reads and stores numbers in double precision [13].

Problem solution may also be produced in a binary format (see also section 3.1) using similar techniques. Again file format is irrelevant – LP-DIT user is only provided with a data interchange interface.

## 4.3 Text solution file format

When requested SIMPLEX may produce a text solution file, which contains five sections:

1. **Header section:** contains linear problem name (as read from the input file) and solution status, which may be OPTIMAL, UNBOUNDED or INFEASIBLE.
2. **Equations section:** presents binding (i.e. non-free) rows of the constraint matrix in a table. The columns of the table contain:
   - row number (free rows are not counted),
   - row label,
   - constraint type (as read from the input file),
   - lower limit on row activity (or blank if there is none),
   - upper limit on row activity (or blank if there is none),
   - row activity and
   - slack value for this row (or blank for equality rows).
3. **Dual variables section:** lists values of dual variables for all non-free rows of the constraint matrix. The first two columns of the table are identical to the previous section, the third one contains the value of the dual variable.
4. **Columns section:** presents a table with the final values of all primal variables together with some data read from the input file. The table contains:

[12] For further information concerning LP-DIT see [12].

[13] For more information on SIMPLEX data types see also appendix A.

- column number,
- column label,
- variable type (as read from the input file),
- lower limit on variable value (`-inf` denotes infinite lower bound),
- upper limit on variable value (`inf` denotes infinite upper bound),
- variable value and
- reduced cost (or dual slack) for this variable.

5. **Objective section:** gives the final value of the objective function together with an estimate of the quality of the solution in form of infeasibility (computed as Euclidean norm of a vector of violation of primal variables' box constraints), primal residuals (an Euclidean norm of residual vector $[\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}]$) and dual residuals (a norm of $\left[\mathbf{B}^T\mathbf{y} - \mathbf{c_B}\right]$ vector).

Figure 5 presents an example solution file for a simple linear problem named `example`.

## 4.4 Final (optimal) basis file format

As some programs may wish to use optimal basis found by `SIMPLEX` optimizer, it is possible for `SIMPLEX` to produce a text file in which the final basis is defined. For the sake of readability and portability between different computers, we have decided to use a text format similar to fixed MPS.

The basis is defined in terms of basic columns and rows. A row is basic if its slack variable belongs in the optimal basis. Equality rows are also assumed to contain slack variables, which are fixed at zero. Each section begins with an indicator line – one which has a keyword starting in the first column. The three sections of the basis file are interpreted as follows:

1. `BASIS` section:

   It consists of a single indicator line starting with word "`BASIS`" in the first column. Columns 15 to 22 are occupied by problem name (as read from the input file and truncated to 8 characters). A word describing the status of the solution begins in column 25. It may be one of:

   - `OPTIMAL` for a problem for which an optimal solution has been found,
   - `INFEASIBLE` for a problem deemed infeasible,
   - `UNBOUNDED` for a problem which has been found to be unbounded and
   - `UNSOLVED` for a problem that has not been solved (see also section 3 for an explanation when this may happen).

   If a problem has not been solved, this section is also the last section of the basis file. If the problem was infeasible or unbounded, the basis is simply the last simplex basis produced during the simplex iterations.

2. `COLUMNS` section:

   It starts with an indicator line with word `COLUMNS`. In columns 25 to 36 of the indicator line, the number of columns of the problem may be found. After the indicator line, data lines follow. Each one consists of:

   - "`BA`" (for a basic variable),
   - "`UP`" (for a non-basic variable placed on its upper bound) or
   - "`LO`" (for a non-basic variable placed on its lower bound)

   in columns 2 and 3, followed by the variable's label in columns 5 to 12.

3. ROWS section:

   It starts with ROWS indicator line. In columns 25 to 36 of this line, the number of problem's rows is found. After the indicator line, data lines follow. Each one consists of

   - "BA" (for a constraint for which a slack is found in the final basis),
   - "UP" (for a constraint, which was active in the last iteration with activity on upper bound) or
   - "LO" (for a constraint, which was active in the last iteration with activity on lower bound)

   in columns 2 and 3, followed by the row's label in columns 5 to 12.

The file ends with an "ENDATA" indicator line. An example optimal basis file for the same linear problem example, for which a solution was given in figure 5, is shown in figure 6.

# 5 Numerical results

We shall present in this section some computational results that demonstrate the efficiency and reliability of our experimental simplex method implementation on some linear programs from the NETLIB test collection of Gay [5]. A more comprehensive overview and discussion of those results may be found in Gondzio *et al.* [8].

Table 2 first lists dimensions of the LP tests used in our analysis. The first four columns of the table contain: problem name, number of constraints $m$, number of variables (excluding slacks) $n$ and number of non-zero elements of the constraint matrix (excluding free rows and right-hand side vector) $n_{nz}$. Then comes comparison of the performance of our code with those of two well-known commercial simplex implementations: CPLEX v. 1.0 of Bixby [2] and MINOS 5.3 of Murtagh and Saunders [14].

Efficiency of all three codes compared is represented by iteration counts and (where available) computation times. For MINOS, only iteration counts are given (data collected in this column was repeated after Table 3 of Gill et al. [6]). For CPLEX and SIMPLEX both iteration counts and computation times on the same SUN SparcStation are reported[14]. Both CPLEX and SIMPLEX were run with default settings of parameters controlling stability (see Bixby [2] and Świętanowski [15] respectively for details).

The results of the tables show that the simplex implementation described in this paper is reasonably efficient, at least in terms of the number of iterations needed to solve an LP problem. With default settings of parameters controlling stability, our code succeeded for all but one of 53 NETLIB tests, which we consider a satisfactory result. For all 52 problems, an exact optimal solution (with at least 8 digits of accuracy) was found. Due to excessive numerical difficulties we were not able to solve a problem called PILOT.

Data collected in the tables shows that our code needs up to an order of magnitude more time than CPLEX to solve a given problem, nevertheless iteration counts are comparable. We have to comment on it to avoid straightforward conclusion that could be drawn after analyzing the results from table 2. CPLEX is a commercial LP system (one of the best currently available) with linear algebra dedicated to a given computer. Its level of efficiency cannot easily be reached. We made the comparison to demonstrate our code's true efficiency, but we are aware that a lot of work still has to be done for SIMPLEX to reach the speed of commercial software.

[14] Since the test results have been gathered in June 1993, the code has overgone some changes. Both iteration counts and times are different, if they are measured now. Notably average time per iteration has been lately decreased twofold, while the iteration counts have not (on the average) increased.

# 6 Conclusions

We treat the code as a good starting point in search for more effective methods for linear program solution. We are currently working on implementation of several enhancements that will improve SIMPLEX's efficiency. The most important of them are the use of steepest-edge pricing (see e.g. Goldfarb and Reid [7]) and presolving (see e.g. [1]). The experience with SIMPLEX has also led us to believe that a new data storage scheme needs to be developed in order to increase method's speed (especially when steepest edge pricing or reduced cost updates are performed).

Independently, we shall use SIMPLEX as a basis for experiments with new simplex and simplex-like approaches (as e.g. method of Wierzbicki [17]) that aim at exploiting possibilities of parallel and/or distributed computations.

# 7 Acknowledgements

The author wishes to thank Dr Jacek Gondzio for his continuous support and guidance during the development of the SIMPLEX optimizer. I also would like to express my gratitude to Prof. Andrzej Ruszczyński from the Warsaw University of Technology, Institute of Control and Computation Engineering, who provided the initial versions of all factorization routines, as well as most of the linear problem input procedures.

# A Compiling and linking SIMPLEX

SIMPLEX was compiled and run in a few UNIX System V environments, including SUN SparcStation, DEC workstation series 5000 with a MIPS processor, and 386 and 486 PC's running BSD 4.3 or SCO UNIX. In all cases, GNU C++ compilers versions 2.4 to 2.5.7 were used. Although an executable version is available only for a SUN SparcStation computer, we believe that porting the solver to other environments should be relatively easy. In this section, the reader will find instructions describing how this can be done.

As was mentioned in the preceding sections, it is possible to define some of the operating characteristics at compile time. We will now explain what can be changed in this manner. Finally, some remarks on running SIMPLEX in a PC–based environment will be presented.

## A.1 Data types

Since C++ programming language does not define exactly integral and floating point data types' precision, for the sake of portability it had to be assumed, that instead of standard data types, we will use a set of user defined data types, for which representation and precision will be known. To that end we define in "`machine.h`" header file a set of five data types specially suited for a few computer – operating system – compiler combinations.

Currently, the GNU C++ compiler versions 2.4 to 2.5.7 are supported on the following computers running listed operating systems[15]:

| Computer type | Operating system |
|---|---|
| Sun SparcStation | SunOS, Solaris |
| DEC 5000/xxx | ULTRIX 4.3 |
| 386 or 486 PC | BSD 4.3, SCO UNIX |

In the header file, you will find definitions of the following data types:

- `Int_T` is used for row and column indexing operations. Thus its range limits the maximum number of rows and columns. It has to be a signed type.

  **Recommended**: at least 16 bits long signed integral type.

[15]The table does not list the (also already defined) data types' set for a PC based DOS system and a Borland C++ v. 3.1 (or later) compiler. For reasons — see appendix A.3.

- `Long_T` is used during Markowitz pricing and limits the number of non–zeros in constraint matrix and basis factorization. It has to hold numbers at least two times longer than `Int_T`. Since it is used in indexing operations, it may not be longer than predefined `size_t` C++ (and C) type.

  **Recommended**: at least 32 bits signed integral type.

- `Real_T` is a type for all floating point arithmetic and data storage.

  **Recommended**: 64 bits long floating point type.

- `Unsigned_T` will soon become obsolete, but is still needed now. Should be an unsigned integral type of the same length as `Int_T`.

- `Short_T` is used to hold signed data, that are never longer than 8 bits.

  **Recommended**: a signed integral type exactly 8 bit long.

If the "`machine.h`" header file does not include data definition for a computer system on which the program is compiled, an error message will be issued during compilation. Following the above mentioned guidelines you may easily add necessary definitions to the header file.

## A.2 Compile time reconfiguration

In `SIMPLEX`, not all configuration changes may be done via command line options or configuration files. A few of the configurable features may only be changed at compile time. To make such changes as easy as possible, the author provided a configuration header file "`compile.h`". By defining or failing to define certain macros, you decide which parts of code should be compiled, or select values of some constants used at runtime.

Last, but not least, the `SIMPLEX` program is equipped with a large number of safeguards against common programming errors. When configured to do so, it will check its internal data integrity and, whenever an anomaly is found, issue an error message and terminate. A message of this kind could look like this:

```
FATAL ERROR: solver.cc: Solve: Internal error #11.
```

In the "`compile.h`" file, there are several commented macro definitions containing the word "`DEBUG`". Each one of them is responsible for detecting errors belonging to one category, or errors detected in one source file. If one should ever experience problems that one suspects may be a result of a programming error, one ought to try running the program compiled with all "`DEBUG`" macros not commented, then contact the author of the code. Additionally, "`COMP_ERROR_ABORT`" macro lets one decide whether a fatal error (see example above) will call ANSI C function `abort()` before terminating. Calling `abort()` may result in producing a `core` file, which will make post mortem debugging possible.

Meanings of all other macro definitions that you may wish to modify are explained in comments in "`compile.h`" file. Those include:

- `SCALE_ROWS` which determines, whether the scaling algorithm will scale constraint matrix rows,

- `SCALE_COLS` which determines, whether the scaling algorithm will scale columns,

- `PASSES` denotes number of passes to be performed by the scaling routine[16],

- `TIMER_ON` allows to switch the time measuring functions on or off,

[16] This is meaningful only when both rows and columns of the constraint matrix are scaled.

- `ITER_LIM_MULT` is used to specify the iteration limit relative to linear problem dimension as $C(m+n)$, where $m$ and $n$ denote number of constraint matrix binding rows and columns respectively, and $C$ is a positive constant given by the user.
- `INCLUDE_LP_DIT_SUPPORT` decides if LP-DIT support functions will be available.

### A.3 Compiling the solver on a PC–based computer

PC–based computers' capabilities are seriously limited by 64 KB memory segments and an idea of memory models [17]. The author has not yet had a chance to work with a C++ compiler that would make those ideas obsolete, but has chosen to ignore the problem. It is therefore possible to compile and run SIMPLEX on a PC–based system, but responsibility to choose an appropriate memory model for the sizes of problems that are to be solved, rests with the user. The program worked successfully when compiled with a Borland C++ compiler version 3.1 in a "large" memory model. The size of MPS source file containing the largest solvable problems is approximately 200 KB.

Since `times()` function (which we use to measure time spent in different parts of the program) is available only in UNIX System V environment, time measuring is not performed in DOS. Likewise LP-DIT library for DOS was not available at the time the code was written. Both time measuring functions and LP-DIT support are automatically excluded from compilation and linking when the program is compiled in DOS environment.

## References

[1] Andersen E. D., Andersen K. D. (1993) *Presolving in Linear Programming*, Technical Report, Departament of Mathematics and Computer Sciences, Odense University, Denmark.

[2] Bixby R. (1992) *Implementing the simplex method: the initial basis*, ORSA Journal on Computing 4, No 3, pages 267–284.

[3] Dantzig G. B. (1963) *Linear Programming And Extensions*, Princeton 1963.

[4] Forrest J. J. H., Tomlin J. A. (1992) *Implementing the simplex method for the Optimisation Subroutine Library*, IBM Systems Journal 31, No 2, pages 11–25.

[5] Gay D. M. (1985) *Electronic mail distribution of linear programming test problems*, Mathematical Programming Society COAL Newsletter.

[6] Gill P. E. Murray W. Saunders M. A. Wright M. H. (1989) *A practical anti-cycling procedure for linearly constrained optimization*, Mathematical Programming 45, pages 437–474.

[7] Goldfarb D., Reid J. K. (1977) *A practicable steepest-edge simplex algorithm*, Mathematical Programming 12, pages 361–371.

[8] Gondzio J., Ruszczyński A., Świętanowski A. (1993) *Towards Another Efficient Implementation of the Simplex Method*, to appear as a technical report of the Institute of Automatic Control, Warsaw University of Technology.

[9] Gould N. I. M., Reid J. K. (1989) *New crash procedures for large systems of linear constraints*, Mathematical Programming 45, pages 475–501.

[10] Harris P. M. J. (1973) *Pivoting selection methods of the Devex LP code*, Mathematical Programming 5, pages 30–57.

[17] Consult your compiler's documentation for details on the concept of memory models, the size of largest memory block that can be allocated for one vector and other memory–related limitations.

[11] *IBM Mathematical Programming System Extended/370 (MPSX/370): Program Reference Manual*, IBM SH19-1127.

[12] Makowski M. (1994) *LP-DIT Data Interchange Tool for Linear Programming Problems (version 1.20)*, IIASA Working Paper No WP-94-36, Laxenburg, Austria.

[13] Murtagh B. (1981) *Advanced Linear Programming, Computation and Practice*, McGraw-Hill, New York.

[14] Murtagh B., Saunders M. A. (1987) *MINOS 5.1 User's guide*, Technical Report SOL 83–20R, Department of Operations Research, Stanford University, Stanford, California 1983 (revised 1987).

[15] Świętanowski A. (1993) *A modern implementation of the revised simplex method for large scale linear programming*, MSc. Thesis, Institute of Automatic Control, Warsaw University of Technology, Warsaw 1993 (in Polish).

[16] Tomlin J. A. (1975) *On Scaling Linear Programming Problems*, Mathemal Programming Study 4, pages 146–166.

[17] Wierzbicki A. P. (1993) *Augmented simplex: a modified and parallel version of simplex method based on multiple objective and subdifferential optimization approach*, Manuscript, Institute of Automatic Control, Warsaw University of Technology, Warsaw 1993.

Figure 4: Example of output in **high** verbosity mode.

```
------------------------------------------------------------------------------
SIMPLEX v. 2.17 (c) 1992, 1993 Artur Swietanowski
Developed in Warsaw University of Technology, Institute of Automatic Control
------------------------------------------------------------------------------
Reading input file in fixed MPS format.
Reading NAME.
Reading ROWS.
Reading COLUMNS.
Reading RHS.
Reading ENDATA.

Linear problem name is AFIRO.

 Problem Statistics      Rows      Cols Non-zeros   Density
                           27        32        83     9.606

   Ranges    E rows    L rows    G rows     Fixed      Free   Bounded    Normal
        0         8        19         0         0         0         0        32

Init cond. = 1.356E+00 No scaling!
Cols:     27 orig.        0 artif.      2 sup.-artif.

 Iter       RC     STEP  C IN C OUT                 COST
    1 -3.8E-01  7.0E-06     3     47     3.94451E+04
    2 -1.4E+00  7.9E-06    19     45     3.94451E+04
    3 -7.3E-01  1.8E-05    17     16     3.94451E+04
    4 -1.1E+00  7.9E-06    30     46     3.94451E+04
    5 -4.8E-01  4.4E+01    29     39     3.94240E+04
    6  3.2E+02  6.2E+01    52            1.97120E+04
    7  3.2E+02  6.2E+01    51            2.11200E+01
    8 -1.2E+01  2.6E+00     2     32    -9.83011E+00
    9 -3.0E-01  5.9E+01    14     33    -2.75159E+01
   10 -8.7E-01  4.8E+02    16     38    -4.43633E+02

Taking final tolerances

Optimum reached.   Iter:       11 Result:  -4.647531428571E+02

Times:     Reading LP  To standard  Init. basis    Solution     Total
                0.07         0.00         0.00         0.08      0.15
```

Figure 5: Example solution file format.

```
LP PROBLEM NAME:  example
STATUS:           OPTIMAL

EQUATIONS
Row No Label    Type   Lower limit   Upper limit          Value          Slack
------ -------- ---- ------------- -------------  -------------  -------------
0      ROW1      LE  -2.000000E+00  4.000000E+00  -2.000000E+00   6.000000E+00
1      ROW2      GE   5.000000E+00                 6.000000E+00  -1.000000E+00
2      ROW3      EQ   8.000000E+00  8.000000E+00   8.000000E+00
3      ROW4      LE                 4.000000E+00  -8.000000E+00   1.200000E+01

DUAL VARIABLES
Row No Label     Dual variable
------ --------  -------------
0      ROW1       2.000000E+00
1      ROW2       0.000000E+00
2      ROW3       0.000000E+00
3      ROW4       0.000000E+00

COLUMNS SECTION
Index  Label    Type   Lower bound   Upper bound   Primal value   Reduced cost
------ -------- ---- ------------- -------------  -------------  -------------
0      X1        PL   0.000000E+00           inf   2.000000E+00   0.000000E+00
1      X2        MI           -inf -2.000000E+00  -2.000000E+00  -1.000000E+00
2      X3        FX   2.000000E+00  2.000000E+00   2.000000E+00  -1.000000E+00
3      X4        FR           -inf           inf   1.000000E+01   0.000000E+00
4      X5        UP  -2.000000E+00  2.000000E+00  -2.000000E+00   1.000000E+00

OBJECTIVE:                -6.000000000000E+00
PRIMAL RESIDUALS:                  0.000000E+00
DUAL RESIDUALS:                    0.000000E+00
INFEASIBILITY:                     0.000000E+00
```

Figure 6: Example optimal basis file format.

```
BASIS         example   OPTIMAL
COLUMNS                 5
 BA X1
 UP X2
 LO X3
 BA X4
 LO X5
ROWS                    4
 LO ROW1
 BA ROW2
 LO ROW3
 BA ROW4
ENDATA
```

Table 2: Numerical results – comparison with MINOS and CPLEX commercial codes.

| Name | Rows | Columns | Non-zeros | MINOS | CPLEX | | simplex | |
|---|---|---|---|---|---|---|---|---|
| | | | | iter. | iter. | time | iter. | time |
| 25FV47 | 821 | 1571 | 10400 | 6646 | 2679 | 65.1 | 6758 | 393.4 |
| 80BAU3B | 2262 | 9799 | 21002 | 10166 | 10635 | 166.2 | 11319 | 2485.9 |
| ADLITTLE | 56 | 97 | 383 | 121 | 94 | 0.1 | 97 | 0.4 |
| AFIRO | 27 | 32 | 83 | 9 | 10 | 0.0 | 13 | 0.0 |
| BANDM | 305 | 472 | 2494 | 457 | 283 | 1.9 | 620 | 17.2 |
| BEACONFD | 173 | 262 | 3375 | 91 | 31 | 0.1 | 24 | 0.4 |
| BORE3D | 233 | 315 | 1429 | 144 | 113 | 0.4 | 204 | 3.9 |
| BRANDY | 220 | 249 | 2148 | 369 | 174 | 1.1 | 1218 | 29.0 |
| CAPRI | 271 | 353 | 1767 | 271 | 438 | 2.2 | 565 | 8.2 |
| CZPROB | 929 | 3523 | 10669 | 1694 | 1178 | 10.8 | 1272 | 53.7 |
| E226 | 223 | 282 | 2578 | 545 | 361 | 1.9 | 785 | 14.6 |
| ETAMACRO | 400 | 688 | 2409 | 567 | 718 | 3.7 | 1744 | 36.7 |
| FFFFF800 | 524 | 854 | 6227 | 796 | 683 | 4.5 | 858 | 23.0 |
| GANGES | 1309 | 1681 | 6912 | 757 | 605 | 9.6 | 500 | 17.5 |
| GFRD-PNC | 616 | 1092 | 2377 | 672 | 525 | 3.1 | 884 | 16.1 |
| GROW7 | 140 | 301 | 2612 | 184 | 275 | 2.5 | 176 | 3.0 |
| GROW15 | 300 | 645 | 5620 | 425 | 621 | 11.3 | 363 | 11.3 |
| GROW22 | 440 | 946 | 8252 | 634 | 892 | 21.8 | 500 | 21.6 |
| ISRAEL | 174 | 142 | 2269 | 251 | 204 | 1.0 | 181 | 3.6 |
| NESM | 662 | 2923 | 13288 | 3228 | 4094 | 31.0 | 4501 | 184.5 |
| PILOT | 1441 | 3652 | 43167 | 13723 | 7402 | 768.3 | — | — |
| PILOT4 | 410 | 1000 | 5141 | 1543 | 1015 | 14.6 | 2213 | 127.6 |
| PILOT.JA | 940 | 1988 | 14698 | 6487 | 4365 | 139.0 | 13590 | 2088.6 |
| PILOT.WE | 722 | 2789 | 9126 | 5458 | 2652 | 76.2 | 8226 | 1148.3 |
| RECIPE | 91 | 180 | 663 | 27 | 41 | 0.1 | 33 | 0.1 |
| SC105 | 105 | 103 | 280 | 457 | 56 | 0.2 | 62 | 0.4 |
| SC205 | 205 | 203 | 551 | 141 | 129 | 0.7 | 144 | 2.1 |
| SCAGR7 | 129 | 140 | 420 | 98 | 87 | 0.2 | 166 | 1.2 |
| SCAGR25 | 471 | 500 | 1554 | 338 | 400 | 3.3 | 675 | 15.7 |
| SCFXM1 | 330 | 457 | 2589 | 375 | 324 | 1.5 | 505 | 9.5 |
| SCFXM2 | 660 | 914 | 5183 | 670 | 669 | 5.3 | 1024 | 32.4 |
| SCFXM3 | 990 | 1371 | 7777 | 1008 | 1050 | 11.6 | 1595 | 71.1 |
| SCORPION | 388 | 358 | 1426 | 178 | 168 | 0.8 | 339 | 5.0 |
| SCRS8 | 490 | 1169 | 3182 | 743 | 561 | 4.6 | 859 | 23.3 |
| SCSD1 | 77 | 760 | 2388 | 303 | 161 | 0.5 | 292 | 3.1 |
| SCSD6 | 147 | 1350 | 4316 | 1306 | 427 | 1.4 | 655 | 12.2 |
| SCSD8 | 397 | 2750 | 8584 | 3138 | 1320 | 8.2 | 1321 | 50.8 |
| SCTAP1 | 300 | 480 | 1692 | 264 | 207 | 0.9 | 383 | 5.1 |
| SCTAP2 | 1090 | 1880 | 6714 | 761 | 548 | 4.7 | 966 | 36.6 |
| SCTAP3 | 1480 | 2480 | 8874 | 904 | 786 | 9.2 | 1319 | 67.3 |
| SEBA | 515 | 1028 | 4352 | 351 | 234 | 1.4 | 29 | 1.0 |
| SHARE1B | 117 | 225 | 1151 | 266 | 175 | 0.6 | 273 | 2.5 |
| SHARE2B | 96 | 79 | 694 | 173 | 111 | 0.2 | 143 | 0.9 |
| SHELL | 536 | 1775 | 3556 | 299 | 494 | 3.0 | 626 | 11.7 |
| SHIP04L | 402 | 2118 | 6332 | 291 | 353 | 1.6 | 236 | 5.9 |
| SHIP04S | 402 | 1458 | 4352 | 163 | 276 | 1.2 | 179 | 3.6 |
| SHIP08L | 778 | 4283 | 12802 | 474 | 804 | 6.6 | 495 | 22.6 |
| SHIP08S | 778 | 2387 | 7114 | 254 | 429 | 3.6 | 281 | 9.3 |
| SHIP12L | 1151 | 5427 | 16170 | 959 | 969 | 12.5 | 963 | 52.7 |
| SHIP12S | 1151 | 2763 | 8178 | 437 | 528 | 6.3 | 509 | 20.4 |
| SIERRA | 1227 | 2036 | 7302 | 648 | 455 | 4.4 | 805 | 33.3 |
| STAIR | 356 | 467 | 3856 | 577 | 417 | 6.6 | 1230 | 76.9 |
| STANDATA | 359 | 1075 | 3031 | 65 | 153 | 0.7 | 145 | 2.2 |
| VTP.BASE | 198 | 203 | 908 | 152 | 114 | 0.3 | 455 | 4.3 |