GLPK

GNU Linear Programming Kit
Version 2.1
Implementation of the Revised Simplex Method
February 2001

Andrew Makhorin

The GLPK package is a part of the GNU project, released under the aegis of GNU.

Copyright © 2000, 2001 Andrew Makhorin, Department for Applied Informatics, Moscow Aviation Institute, Moscow, Russia. All rights reserved.

Free Software Foundation, Inc., 59 Temple Place — Suite 330, Boston, MA 02111, USA.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.

Table of Contents

Preface 1

1 Theoretical Background 3

1.1 Fundamentals 3
1.2 Primal simplex method 6
1.3 Dual simplex method 8
1.4 Computing components of the simplex table 9
1.5 Explicit formulae for updating the simplex table 11
1.6 Steepest edge technique 12
1.7 Advanced ratio test based on Harris technique 16
1.8 Scaling LP problem 17

2 Simplex Method Routines 19

2.1 Structure of common block 19
2.2 Operations on basis matrix 21
2.2.1 `invert_b` — rebuild representation of the basis matrix 21
2.2.2 `ftran` — perform forward transformation 21
2.2.3 `btran` — perform backward transformation 22
2.2.4 `update_b` — update representation of the basis matrix 22
2.3 Operations on simplex table 23
2.3.1 `check_rsm` — check common block for correctness 23
2.3.2 `eval_xn` — determine value of non-basic variable 23
2.3.3 `eval_bbar` — compute values of basic variables . . 24
2.3.4 `eval_pi` — compute simplex multipliers 24
2.3.5 `eval_cbar` — compute reduced costs of non-basic variables 24
2.3.6 `check_rr` — check relative residual 25
2.3.7 `check_bbar` — check basis solution for primal feasibility 25
2.3.8 `check_cbar` — check basis solution for dual feasibility 26
2.3.9 `eval_col` — compute column of the simplex table 26
2.3.10 `eval_zeta` — compute row of the inverse 27
2.3.11 `eval_row` — compute row of the simplex table . . 27
2.3.12 `change_b` — change basis 28
2.4 Primal steepest edge routines 28
2.4.1 `init_gvec` — initialize the vector γ 28

2.4.2 `update_gvec` — update the vector γ 29
2.4.3 `exact_gvec` — compute exact value of γ 30
2.4.4 `check_gvec` — check accuracy of the vector γ . . . 30
2.5 Dual steepest edge routines 31
2.5.1 `init_dvec` — initialize the vector δ 31
2.5.2 `update_dvec` — update the vector δ 31
2.5.3 `exact_dvec` — compute exact value of δ 33
2.5.4 `check_dvec` — check accuracy of the vector δ 33
2.6 Primal simplex method routines 33
2.6.1 `pivot_col` — choose non-basic variable (primal) 33
2.6.2 `pivot_row` — choose basic variable (primal, standard technique) 34
2.6.3 `harris_row` — choose basic variable (primal, Harris technique) 35
2.7 Dual simplex method routines 36
2.7.1 `dual_row` — choose basic variable (dual) 36
2.7.2 `dual_col` — choose non-basic variable (dual, standard technique) 37
2.7.3 `harris_col` — choose non-basic variable (dual, Harris technique) 38

References 41

Index 43

Preface

This document describes GLPK[1] implementation of the revised simplex method for solving large-scale linear programming and other related problems.

Although the first part of the document contains necessary theoretical information, this document is not intended to be used as a tutorial. It is assumed that the reader has basic knowledges in the linear programming theory.

[1] GLPK is acronym for GNU Linear Programming Kit.

1 Theoretical Background

1.1 Fundamentals

The GLPK implementation of the revised simplex method assumes the following formulation of the linear programming (LP) problem:

$$\begin{aligned} Z &= \widehat{c}^T x_S + c_0 \rightarrow \min\ (\max) \\ x_R &= \widehat{A} x_S \\ l_R &\le x_R \le u_R \\ l_S &\le x_S \le u_S \end{aligned} \tag{1}$$

where: x_S — vector of structural variables;
x_R — vector of auxiliary variables;
Z — objective function;
$\widehat{c}$ — vector of coefficients of the objective function at the structural variables;
c_0 — constant term of the objective function;
$\widehat{A}$ — matrix of coefficients of equaility constraints;
l_R, u_R — vectors of lower and upper bounds of auxiliary variables;
l_S, u_S — vectors of lower and upper bounds of structural variables;

Bounds of auxiliary variables can be finite as well as infinite. Besides, a lower bound can be equal to an upper bound, therefore the following five types of variables are possible:

Bounds of variable	Type of variable
$-\infty < x_k < +\infty$	Free (unbounded) variable
$l_k \le x_k < +\infty$	Variable with lower bound
$-\infty < x_k \le u_k$	Variable with upper bound
$l_k \le x_k \le u_k$	Double-bounded variable
$l_k = x_k = u_k$	Fixed variable

For the sake of simplicity we may do the following:

a) we may disregard the case of maximization, because replacing Z by $(-Z)$ we can reduce it to the case of minimization;

b) we may disregard the constant term c_0, because it doesn't affect on the optimal solution;

c) from the point of view of the simplex method there is no difference between auxiliary and structural variables. Thus, it is more convenient to introduce the united vector of variables

$$x = (x_R, x_S) \tag{2}$$

d) using (2) we may introduce the expanded matrix of constraint coefficients

$$A = (I| - \widehat{A}), \tag{3}$$

where I is the unity matrix that corresponds to the auxiliary variables, $\widehat{A}$ is the original matrix of constraint coefficients from (1) that corresponds to the structural variables;

e) using (2) we also may introduce the expanded vector of objective coefficients

$$c = (0, \widehat{c}), \tag{4}$$

where 0 is zero subvector that corresponds to the auxiliary variables, $\widehat{c}$ is the original vector of objective coefficients from (1) that corresponds to the structural variables.

Summing up all notices above we can rewrite the LP problem (1) in the following form:

$$Z = c^T x \tag{5}$$
$$Ax = 0 \tag{6}$$
$$l \le x \le u \tag{7}$$

Let's assume that the LP problem has m auxiliary variables and n structural variables. Then the constraint matrix A has m rows and $m + n$ columns.

Since the matrix A includes a non-singular submatrix of order m (for instance, the unity submatrix), it has full row rank, i.e. $\mathrm{rank}(A) = m$. Therefore it can be divided into two submatrix:

$$A = (B \mid N), \tag{8}$$

where B is a square non-singular $m \times m$ matrix called *basis matrix*; N is a $m \times n$ matrix.

Using (8) the system of equality constraints (6) can be written in the form

$$Bx_b + Nx_N = 0, \tag{9}$$

where x_B is a vector of *basic variables* corresponding to columns of the matrix B, x_N is a vector of *non-basic variables* corresponding to columns of the matrix N.

The matrix B is non-singular, due to that the basic variables as well as the objective function can be explicitly expressed through the non-basic variables. So, from (9) we have

$$x_B = -B^{-1}Nx_N. \tag{10}$$

Substituting x_B from (10) to (5) we also have

$$Z = c^T x = c_B^T x_B + c_N^T x_N = c_B^T(-B^{-1}Nx_N) + c_N^T x_N = (c_N^T - c_B^T B^{-1}N)x_N. \tag{11}$$

Thus, using the decomposition (8) and accounting (10) and (11) the LP problem (5)—(7) can be rewritten in the form called *the simplex table*:

$$Z = d^T x_N \tag{12}$$
$$x_B = \widetilde{A}x_N \tag{13}$$
$$l_B \le x_B \le u_B \tag{14}$$
$$l_N \le x_N \le u_N \tag{15}$$

The simplex table (12)—(15) includes the following two principal components[1]

$$d = (d_j) = c_N - N^T B^{-T} c_B = c_N + \widetilde{A}^T c_B \tag{16}$$

$$\widetilde{A} = (\alpha_{ij}) = -B^{-1} N \tag{17}$$

where d — vector of *reduced costs* (or *marginal values*) of non-basic variables (d_j shows by what quantity the objective function increases or decreases when the non-basic variable $(x_N)_j$ increases or decreases by one); $\widetilde{A}$ — matrix of *influence coefficients* (α_{ij} shows by what quantity the basic variable $(x_B)_i$ increases or decreases when the non-basic variable $(x_N)_j$ increases or decreases by one).

Basic solution of the LP problem (5)—(7) that corresponds to the basis B is a solution which satisfies to the equality constraints (6) and where non-basic free variables are equal to zero and other non-basic variables are on their bounds.

Let the set of non-basic variables x_N is divided into four subsets:

$$x_N = x_F \cup x_L \cup x_U \cup x_S, \tag{18}$$

where: x_F — subset of non-basic free variables;
x_L — subset of non-basic variables which are on lower bounds;
x_U — subset of non-basic variables which are on upper bounds;
x_S — subset of non-basic fixed variables.

Then the basic solution can be written in the form:

$$x = (x_B, x_N), \tag{19}$$

where values of non-basic variables x_N are determined as

$$(x_N)_j = \begin{cases} 0, & if\ (x_N)_j \in x_F \\ (l_N)_j, & if\ (x_N)_j \in x_L \\ (u_N)_j, & if\ (x_N)_j \in x_U \\ (l_N)_j = (u_N)_j, & if\ (x_N)_j \in x_S \end{cases} \tag{20}$$

and values of basic variables are determined by (13). Besides, the value of the objective function Z can be determined by direct substitution of values of basic and non-basic variables into (5).

Basic solution is called *primal feasible basic solution* if all variables satisfy to the conditions (7).

Let the vector β denote values of basic variables x_B in some basic solution. Non-basic variables satisfy to the conditions (7) by definition. Therefore the basic solution is primal feasible if

$$(l_B)_i \le \beta_i \le (u_B)_i,\ i = 1, \ldots, m. \tag{21}$$

Basic solution is called *dual feasible basic solution* if all reduced costs (16) satisfy to the following conditions:

$$\begin{aligned} &if\ (x_N)_j \in x_F\ then\ d_j = 0 \\ &if\ (x_N)_j \in x_L\ then\ d_j \ge 0 \\ &if\ (x_N)_j \in x_U\ then\ d_j \le 0 \\ &if\ (x_N)_j \in x_S\ then\ -\infty < d_j < +\infty \end{aligned} \tag{22}$$

[1] Here and further B^{-T} means $(B^T)^{-1}$.

i.e. if there is no non-basic variable which changing in the feasible direction can improve (decrease) the objective function.

Basic solution which is primal and dual feasible at the same time is *optimal solution* of the LP problem (5)—(7).

The primal simplex method is an iterative numerical procedure which starting from some primal feasible basic solution tries to make it dual feasible. Analogously *the dual simplex method* is a procedure which starting from some dual feasible solution tries to make it primal feasible.

The revised (primal or dual) simplex method is a variant of the simplex method which doesn't keep the simplex table (12)—(15) explicitly.[2] Instead that it computes only necessary components of the simplex table.

1.2 Primal simplex method

Let there is some initial primal basic solution defined by (8) and (18). The primal revised simplex method for solving the LP problem (5)—(7) includes the following principal steps.

Step 1. [Computing current values of variables.] Compute current values of non-basic variables using the formula (20).

Compute current values of basic variables β using the formula (10). (Because the current basis solution is assumed to be primal feasible, the primal feasibility conditions (21) should be satisfied.)

On this step the current value of the objective function Z also can be computed by direct substitution of the computed values of variables into the formula (5).

Step 2. [Computing reduced costs.] Compute reduced costs d of non-basic variables using the formula (16).

Step 3. [Choosing non-basic variable.] Choose a non-basic variable $(x_N)_q$ which doesn't satisfy to the dual feasibility conditions (22), i.e. that variable for which:

$$\begin{aligned} &(x_N)_q \in x_F \ and \ d_q \neq 0, \ or \\ &(x_N)_q \in x_L \ and \ d_q < 0, \ or \\ &(x_N)_q \in x_U \ and \ d_q > 0. \end{aligned} \tag{23}$$

where d_q is the reduced cost of the variable $(x_N)_q$. The conditions (23) mean that the variable $(x_N)_q$ changing in feasible direction (i.e. increasing if $(x_N)_q \in x_F \cup x_L$ or decreasing if $(x_N)_q \in x_F \cup x_L$) can improve (decrease) the objective function. If there is no such non-basic variable, STOP — the current basis solution is optimal because it is primal and dual feasible.

Step 4. [Computing influence coefficients.] Compute q-th column of the matrix $\widetilde{A}$ (see (17)) which contains influence coefficients for the chosen non-basic variable $(x_N)_q$.

Step 5. [Choosing basic variable.] Let the chosen non-basic variable $(x_N)_q$ is changing in the feasible direction. On this step it is necessary to detect one of the following three cases:

[2] A variant of the simplex method which uses explicit representation of the simplex table is called *the classic simplex method*. It doesn't allow exploiting advantages of sparse computations and therefore it doesn't fit for large-scale LP problems due to its low efficiency.

1) some basic variable reaches its lower bound first, or

2) some basic variable reaches its upper bound first, or

3) the non-basic variable $(x_N)_q$ reaches its opposite bound first.

In order to formalize this step let's consider i-th row of the simplex table (13) that corresponds to i-th equality constraint:

$$(x_B)_i = \alpha_{i1}(x_N)_1 + \ldots + \alpha_{iq}(x_N)_q + \ldots + \alpha_{in}(x_N)_n. \tag{24}$$

Let $\theta_i \geq 0$ is absolute value of change of the non-basic variable $(x_N)_q$ at which $(x_B)_i$ reaches its (lower or upper) bound:

a) if $(x_N)_q$ is increasing, i.e. if $d_q < 0$

$$\theta_i = \begin{cases} \infty, & \alpha_{iq} = 0 \\ [(u_B)_i - \beta_i]/\alpha_{iq}, & \alpha_{iq} > 0 \\ [(l_B)_i - \beta_i]/\alpha_{iq}, & \alpha_{iq} < 0 \end{cases} \tag{25}$$

b) if $(x_N)_q$ is decreasing, i.e. if $d_q > 0$

$$\theta_i = \begin{cases} \infty, & \alpha_{iq} = 0 \\ [\beta_i - (l_B)_i]/\alpha_{iq}, & \alpha_{iq} > 0 \\ [\beta_i - (u_B)_i]/\alpha_{iq}, & \alpha_{iq} < 0 \end{cases} \tag{26}$$

Obviously the basic variable reaching its bound before other basic variables is determined by the minimal θ_i. Therefore the choice is based on the quantity

$$\theta = \min[\min_i \theta_i, (u_N)_q - (l_N)_q] \tag{27}$$

If $\theta = \theta_p$, choose the basic variable $(x_B)_p$.

If $\theta = (u_N)_q - (l_N)_q < \theta_i$ for all $i = 1, \ldots, m$, the case (3) takes place.

If $\theta = \infty$, STOP — the problem has unbounded solution (i.e. it has no dual feasible solutions), because the objective function can unlimitedly decrease.

Step 6. [Change the basis.] If on the step 5 the basic variable $(x_B)_p$ was chosen, replace this variable in the set of basic variables x_B by the chosen non-basic variable $(x_N)_q$. From the point of view of (8) such replacement means that in the matrix B the column B_p should be replaced by the column N_q and vice versa, in the matrix N the column N_q should be replaced by the column B_p.

If on the step 5 the case (3) took place, just move the chosen non-basic variable $(x_N)_q$ from its current bound to the opposite one. (In this case the sets x_B and x_N and the matrices B and N are not changed.)

In both cases the adjacent basis solution remains primal feasible, besides the non-basic variable $(x_N)_q$ increases or decreases by the quantity θ, and the objective function Z decreases by the quantity $|d_q|\theta$.

After changing the basis continue the search starting from the step 1.

1.3 Dual simplex method

The dual simplex method is based on the dual equality constraints

$$A^T\pi + \lambda = c, \tag{28}$$

where $\pi = (\pi_i)$ is the vector of Lagrange multipliers that correspond to the primal equality constraints (6), $\lambda = (\lambda_j)$ is the vector of Lagrange multipliers that correspond to the primal bound constraints (7), c is the vector of coefficients of the objective function. Variables π_i and λ_i are called *dual variables* for the LP problem (5).

From the complementarity conditions it follows that in case of basis solution the decomposition (19) defines the corresponding decomposition of the dual variables

$$\lambda = (\lambda_B, \lambda_N), \tag{29}$$

where Lagrange multipliers λ_B that correspond to basic variables should be equal to zero, and Lagrange multipliers λ_N that correspond to non-basic variables should satisfy to the dual feasibility conditions.

Using (8) and (29) the system (28) can be written in the decomposed form:

$$B^T\pi + \lambda_B = c_B \tag{30}$$
$$N^T\pi + \lambda_N = c_N \tag{31}$$

therefore

$$\pi = -B^{-T}\lambda_B + B^{-T}c_B \tag{32}$$
$$\lambda_N = -N^T\pi + c_N = N^TB^{-T}\lambda_B + (c_N - N^TB^{-T}c_B) = \widetilde{A}^T\lambda_B + d \tag{33}$$

where $\widetilde{A}$ is the matrix of influence coefficients (17), d is the vector of reduced costs of non-basic variables (16).

Let there is some initial dual feasible basic solution defined by (8) and (29). The dual revised simplex method for solving LP problem (5)—(7) includes the following principal steps.

Step 1. [Computing current values of variables.] Compute current values of non-basic variables using the formula (20).

Compute current values of basic variables β using the formula (10).

On this step the current value of the objective function Z also can be computed by direct substitution of the computed values of variables into the formula (5).

Step 2. [Computing reduced costs.] Compute reduced costs d of non-basic variables using the formula (16). (Because the current basis solution is assumed to be dual feasible, the dual feasibility conditions (22) should be satisfied. Note that the reduced costs d are current values of Lagrange multipliers λ_N for non-basic variables.)

Step 3. [Choosing basic variable.] Choose a basic variable $(x_B)_p$ which doesn't satisfy to the primal feasibility conditions (14), i.e. that variable for which:

$$\beta_p < (l_B)_p \ or \ \beta_p > (u_B)_p. \tag{34}$$

If there is no such basic variable, STOP — the current solution is optimal, because it is primal and dual feasible.

Step 4. [Computing influence coefficients.] Compute p-th row of the matrix $\widetilde{A}$ (see (17)) which contains influence coefficients for the dual variable $(\lambda_B)_i$ that corresponds to the chosen basic variable $(x_B)_i$. (Note that p-th row of the matrix $\widetilde{A}$ is p-th column of the matrix $\widetilde{A}^T$; see (33).)

Step 5. [Choosing non-basic variable.] Let the dual variable $(\lambda_B)_p$ that corresponds to the chosen basic variable $(x_B)_p$ is changing in the feasible direction. On this step it is necessary to detect one of the following two cases:

1) some dual variable $(\lambda_N)_j$ being non-negative reaches its lower (zero) bound first, or

2) some dual variable $(\lambda_N)_j$ being non-positive reaches its upper (zero) bound first.

In order to formalize this step let's consider j-th row of the transposed (dual) simplex table (33):

$$(\lambda_N)_j = \alpha_{1j}(\lambda_B)_1 + \ldots + \alpha_{pj}(\lambda_B)_p + \ldots + \alpha_{mj}(\lambda_B)_m + d_j. \tag{35}$$

Let θ_j is the absolute value of cahnge of the dual variable $(\lambda_B)_p$ at which $(\lambda_N)_j$ reaches its (lower or upper) zero bound:

$$\theta_j = \begin{cases} |d_j/\alpha_{pj}|, & if\ \beta_p < (l_B)_p, \alpha_{pj} > 0, (x_N)_j \in x_F \cup x_L \\ & or\ \beta_p > (u_B)_p, \alpha_{pj} > 0, (x_N)_j \in x_F \cup x_U \\ & or\ \beta_p < (l_B)_p, \alpha_{pj} < 0, (x_N)_j \in x_F \cup x_U \\ & or\ \beta_p > (u_B)_p, \alpha_{pj} < 0, (x_N)_j \in x_F \cup x_L \\ \infty & in\ other\ cases \end{cases} \tag{36}$$

Obviously the dual variable reaching its bound before other dual basic variables is determined by the minimal θ_j. Therefore the choice is based on the quantity

$$\theta = \min_j \theta_j. \tag{37}$$

If $\theta = \theta_q < \infty$, choose the non-basic variable $(x_N)_q$. Otherwise, if $\theta = \infty$, STOP — the problem has no primal feasible solutions (i.e. the dual objective function can unlimitedly increase).

Step 6. [Change the basis.] Replace the chosen basic variable $(x_B)_p$ in the set of basic variables x_B by the chosen non-basic variable $(x_N)_q$. From the point of view of (8) such replacement means that in the matrix B the column B_p should be replaced by the column N_q and vice versa, in the matrix N the column N_q should be replaced by the column B_p. (Note that the adjacent basis solution remains dual feasible.)

After changing the basis continue the search starting from the step 1.

1.4 Computing components of the simplex table

It was said above that the revised simplex method doesn't keep the simplex table (12)—(15) explicitly (as the classic simplex method does). Instead that it uses only necessary components of the simplex table.

Analysis of the primal and dual variants of the revised simplex method shows that all computations related to the simplex table can be implemented by means of some representation of the basis matrix B, which provides the following four basic operations:

1) building the representation for a given basis matrix B (for example, for the basis matrix that corresponds to an initial basic solution), i.e. *reinverting* the basis matrix;

2) performing so called *forward transformation (FTRAN)* of a given vector u, i.e. solving the system $Bu' = u$;

3) performing so called *backward transformation (BTRAN)* of a given vector u, i.e. solving the system $B^T u' = u$;

4) updating the representation of the current basis matrix B if one column of B has been replaced by other column.

Let's consider how these operations can be used for computing principal components of the simplex table.

The primal simplex method needs β (the current values of basic variables), d (reduced costs of non-basic variables), and $\widetilde{A}_q$ (q-th column of the matrix $\widetilde{A}$ that corresponds to the chosen non-basic variable $(x_N)_q$). The dual simplex method needs β, d, and α_p (p-th row of the matrix $\widetilde{A}$ that corresponds to the chosen basic variable $(x_B)_p$).

The formula (10) shows that the vector β can be computed by means of FTRAN:

$$\beta = B^{-1}(Nx_N), \tag{38}$$

where N is the submatrix of A built of columns of non-basic variables, x_N is the vector of the current values of non-basic variables defined by (20).

Analogously, the vector d can be computed by means of BTRAN as it follows from the formulae (16) and (32):

$$\pi = B^{-T} c_B \tag{39}$$

$$d = c_N - N^T \pi \tag{40}$$

where π is the vector of the current values of *simplex multipliers* (i.e. Lagrange multipliers that correspond to the equality constraints), c_B and c_N are the subvectors of coefficients of the objective function at, respectively, basic and non-basic variables.

Let

$$e_i = (0, \ldots, 0, 1, 0, \ldots, 0)^T \tag{41}$$

is the unity vector with all elements are zero except i-th element which is one. Then from the formula (17) we have:

$$\widetilde{A}_q = \widetilde{A} e_q = -B^{-1} N_q \tag{42}$$

$$\alpha_p = e_p^T \widetilde{A} = -e_p^T B^{-1} N = -(B^{-T} e_p)^T N = -\zeta_p^T N \tag{43}$$

where N_q is q-th column of the matrix N (see (8)), $\zeta_p = B^{-T} e_p$ is p-th column of the matrix B^{-T} (or p-th row of the matrix B^{-1}). Thus, the components $\widetilde{A}_q$ and α_p also can be computed by means of FTRAN and BTRAN operations.

The current GLPK implementation of the revised simplex method allows using one of two representations of the basis matrix: EFI (Elimination Form of the Inverse) and RFI (Reid's Form of the Inverse) [1]. However, due to modular structure of GLPK it is possible to use any other user-supplied representation of the basis matrix.

1.5 Explicit formulae for updating the simplex table

Although the revised simplex method never explicitly updates the simplex table, sometimes it is convenient to know explicit formulae for updating the simplex table (i.e. those formulae which could be used in the classic simplex method).

In order to derive these formulae let's write the simplex table (13) in the unwrapped form:

$$\begin{aligned}
(x_B)_1 &= \alpha_{11}(x_N)_1 + \ldots + \alpha_{1q}(x_N)_q + \ldots + \alpha_{1n}(x_N)_n \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
(x_B)_p &= \alpha_{p1}(x_N)_1 + \ldots + \alpha_{pq}(x_N)_q + \ldots + \alpha_{pn}(x_N)_n \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
(x_B)_m &= \alpha_{m1}(x_N)_1 + \ldots + \alpha_{mq}(x_N)_q + \ldots + \alpha_{mn}(x_N)_n
\end{aligned} \tag{44}$$

Let the basic variable $(x_B)_p$ has been chosen to leave the basis and the non-basic variable $(x_N)_q$ has been chosen to enter the basis.

In order to express the new basic variable $(x_N)_q$ through the new set of non-basic variables (which now includes the old basic variable $(x_B)_p$) we should transform p-th (pivot) row of the simplex table (44):

$$(x_B)_p = \alpha_{pq}(x_N)_q + \sum_{j\neq q} \alpha_{pj}(x_N)_j,$$

whence it follows

$$(x_N)_q = \frac{1}{\alpha_{pq}}(x_B)_p - \sum_{j\neq q} \frac{\alpha_{pj}}{\alpha_{pq}}(x_N)_j. \tag{45}$$

Now we can use the expression (45) in order to eliminate the variable $(x_N)_q$ from other rows of the simplex table. Substitution of $(x_N)_q$ into i-th row of the simplex table ($i \neq p$) gives

$$\begin{aligned}
(x_B)_i &= \alpha_{iq}(x_N)_q + \sum_{j\neq q} \alpha_{ij}(x_N)_j = \\
&= \alpha_{iq}\left[\frac{1}{\alpha_{pq}}(x_B)_p - \sum_{j\neq q} \frac{\alpha_{pj}}{\alpha_{pq}}(x_N)_j\right] + \sum_{j\neq q} \alpha_{ij}(x_N)_j = \\
&= \frac{\alpha_{iq}}{\alpha_{pq}}(x_B)_p - \sum_{j\neq q} \frac{\alpha_{iq}\alpha_{pj}}{\alpha_{pq}}(x_N)_j + \sum_{j\neq q} \alpha_{ij}(x_N)_j = \\
&= \frac{\alpha_{iq}}{\alpha_{pq}}(x_B)_p + \sum_{j\neq q} \left[\alpha_{ij} - \frac{\alpha_{iq}\alpha_{pj}}{\alpha_{pq}}\right](x_N)_j,
\end{aligned} \tag{46}$$

Using (45) and (46) and assuming that in the new simplex table (which corresponds to the adjacent basis) its p-th row corresponds to the new basic variable $(x_N)_q$ and its q-th column corresponds to the new non-basic variable $(x_B)_p$ we can write the necessary formulae for updating the simplex table (44):

a) for p-th (pivot) row

$$\overline{\alpha}_{pq} = \frac{1}{\alpha_{pq}} \tag{47}$$

$$\overline{\alpha}_{pj} = -\frac{\alpha_{pj}}{\alpha_{pq}} = -\overline{\alpha}_{pq}\alpha_{pj},\ j \neq q \tag{48}$$

b) for other rows ($i \neq p$)

$$\overline{\alpha}_{iq} = \frac{\alpha_{iq}}{\alpha_{pq}} = \overline{\alpha}_{pq}\alpha_{iq} \tag{49}$$

$$\overline{\alpha}_{ij} = \alpha_{ij} - \frac{\alpha_{iq}\alpha_{pj}}{\alpha_{pq}} = \alpha_{ij} - \overline{\alpha}_{iq}\alpha_{pj} = \alpha_{ij} + \alpha_{iq}\overline{\alpha}_{pj},\ j \neq q \tag{50}$$

(Note that the considered transformation of the simplex table is one step of Gauss—Jordan elimination procedure which is used for inverting non-singular matrices.)

1.6 Steepest edge technique

The general scheme of the primal simplex method (see Section 1.2) doesn't answer to the question: what non-basic variable $(x_N)_q$ should be chosen on each iteration? In fact, the theory requires only that the chosen non-basic variable should have reduced cost d_q which satisfies to the conditions (23).

The obvious way (called *textbook technique*) is to choose that variable which has the largest (in absolute value) reduced cost, because changing $(x_N)_j$ by one in the feasible direction involves improving (decreasing) the objective function by $|d_j|$. However, should note that actual change of $(x_N)_j$ depends on bound constraints for basic variables and therefore it is unknown until the basic variable has been chosen.

To-day it is well known that the textbook technique is not efficient, because in most practical cases it involves very slow convergence. Geometrically this technique means that we choose an edge of the convex polyhedron (defined by constraints of the LP problem) which has sharpest angle with the antigradient $(-\nabla Z)$ of the objective function in *the space of non-basic variables*. However, this space is changed on each iteration of the simplex method. Transformation applied to this space is not orthogonal, therefore the angle which is sharpest in the current space may be not sharpest in the space that corresponds to the adjacent set of non-basic variables.

In order to get rid of the asymmetry inherent to the textbook technique, D. Goldfarb and J. K. Reid proposed to choose the edge of the polyhedron which has the sharpest angle with the antigradient of the objective function in *the space of all variables*, because this space doesn't depend on the current basis. The corresponding heuristic is called *steepest edge technique* [2]. Numerous experiments show that the steepest edge technuqe allows significantly improving the efficiency of the simplex method[3].

Let $x^0 = (x_B, x_N) = [(x_B)_1, \ldots, (x_B)_m; (x_N)_1, \ldots, (x_N)_n]$ is a vertex of the polyhedron (in the space of all variables) which corresponds to the current basis solution. Let also the non-basic variable $(x_N)_j$ has increased by one, but all other non-basic variables have not been changed. Then the displaced point is $x^j = [(x_B)_1 + \alpha_{1j}, \ldots, (x_B)_m + \alpha_{mj}; (x_N)_1, \ldots, (x_N)_{j-1}, (x_N)_j + 1, (x_N)_{j+1}, \ldots, (x_N)_n]$, where $\alpha_{1j}, \ldots, \alpha_{mj}$ are influence coefficients of j-th column of the matrix $\widetilde{A}$ (see (17)).

It is understood that the vector

$$\Delta x^j = x^j - x^0 = (\alpha_{1j}, \ldots, \alpha_{mj}; 0, \ldots, 0, 1, 0, \ldots, 0) \tag{51}$$

[3] The efficiency of the steepest edge technique is conditioned by that it prevents spiral-like movement through vertices of the polyhedron "stretching out" the trajectory along the antigradient of the objective function.

defines the direction along the edge of the polyhedron which corresponds to the non-basic variable $(x_N)_j$.

We are interested in the angle between the direction Δx_j and the gradient ∇Z which is

$$\cos(\Delta x_j \hat{} \nabla Z) = \cos\varphi^j = \frac{(\Delta x^j)^T \nabla Z}{||\Delta x^j|| ||\nabla Z||}. \tag{52}$$

We know that $Z = c^T x = (c_B)^T x_B + (c_N)^T x_N$ and $\nabla Z = c = (c_B, c_N)$. Therefore

$$(\Delta x^j)^T = \sum_{i=1}^{m} \alpha_{ij}(c_B)_i + (c_N)_j = \widetilde{A}_j^T c_B = (c_N)_j, \tag{53}$$

where $\widetilde{A}_j = (\alpha_{1j}, \ldots, \alpha_{mj})$ is j-th column of the matrix $\widetilde{A}$. However, from (17) it follows that $\widetilde{A}_j = -B^{-1}N$, hence

$$(\Delta x^j)^T \nabla Z = (c_N)_j - N_j^T B^{-T} c_B = d_j, \tag{54}$$

where d_j is the reduced cost of non-basic variable $(x_N)_j$ (cf. (16)).

Substituting (54) into (52) we finally have

$$\cos\varphi^j = \frac{d_j}{||\Delta x^j|| ||\nabla Z||} \tag{55}$$

$$d_j = (c_N)_j - N_j^T B^{-T} c_B \tag{56}$$

$$||\Delta x^j|| = \left(1 + \sum_{i=1}^{m} \alpha_{ij}^2\right)^{1/2} = \left(1 + \widetilde{A}_j^T \widetilde{A}_j\right)^{1/2} \tag{57}$$

$$||\nabla Z|| = \left(1 + c^T c\right)^{1/2} \tag{58}$$

Thus, in the steepest edge technique it is proposed to choose the non-basic variable $(x_N)_q$ with largest (in absolute value) $\cos\varphi^j$ (instead d^j as in case of the textbook technique), because the corresponding movement direction has sharpest angle with the antigradient of the objective function. Should note that in any case the sign of d_j should satisfy to the conditions (23).

On practice we may disregard the term $||\nabla Z||$ in (55), because it is constant and doesn't affect on the choice of non-basic variable.

For the sake of convenience let's introduce the quantities

$$\gamma_j = ||\Delta x^j||^2 = 1 + \sum_i \alpha_{ij}^2 = 1 + \widetilde{A}_j^T \widetilde{A}_j, \ j = 1, \ldots, n. \tag{59}$$

Then in order to exploit the steepest edge technique on the step 3 of the primal simplex method (see Section 1.2) we should choose non-basic variable using *scaled reduced costs*

$$d'_j = \frac{1}{\sqrt{\gamma_j}} d_j, \ j = 1, \ldots, n. \tag{60}$$

It is important to note that the matrix $\widetilde{A}$ is not available explicitly in the case of the revised simplex method that hinders from computing the quantities γ_j directly using the

formula (59). Although theoretically it is possible to compute all columns $\widetilde{A}_j$, practically it would be extermely inefficient, because computation of one column requires one FTRAN. That's why D. Goldfarb and J. K. Reid proposed efficient formulae for *updating* the quantities γ_j for the adjacent basis assuming that they are known for the current basis. These formulae can be easily derived using the explicit formulae for updating the simplex table (see Section 1.5).

At first let's consider updating the quantity γ_q for the pivot column ($j = q$):

$$\begin{aligned}\overline{\gamma}_q &= 1 + \sum_i \overline{\alpha}_{iq}^2 = 1 + \sum_{i \neq p} \overline{\alpha}_{iq}^2 + \overline{\alpha}_{pq}^2 = 1 + \sum_{i \neq p} \frac{\alpha_{iq}^2}{\alpha_{pq}^2} + \frac{1}{\alpha_{pq}^2} = \\ &= 1 + \left(\sum_i \frac{\alpha_{iq}^2}{\alpha_{pq}^2} - 1 \right) + \frac{1}{\alpha_{pq}^2} = \frac{1}{\alpha_{pq}^2} \left(1 + \sum_i \alpha_{iq}^2 \right) = \frac{\gamma_q}{\alpha_{pq}^2}.\end{aligned}$$

So we have:

$$\overline{\gamma}_q = \frac{\gamma_q}{\alpha_{pq}^2} = \frac{1}{\alpha_{pq}^2} \left(1 + \sum_i \alpha_{iq}^2 \right) = \frac{1}{\alpha_{pq}^2} (1 + \widetilde{A}_q^T \widetilde{A}_q). \tag{61}$$

Now let's consider updating the quantities γ_j for other columns ($j \neq q$):

$$\begin{aligned}\overline{\gamma}_j &= 1 + \sum_i \overline{\alpha}_{ij}^2 = 1 + \sum_{i \neq p} \overline{\alpha}_{ij}^2 + \overline{\alpha}_{pj}^2 = 1 + \sum_{i \neq p} \left(\alpha_{ij} - \frac{\alpha_{iq} \alpha_{pj}}{\alpha_{pq}} \right)^2 + \frac{\alpha_{pj}^2}{\alpha_{pq}^2} = \\ &= 1 + \left[\sum_i \left(\alpha_{ij} - \frac{\alpha_{iq} \alpha_{pj}}{\alpha_{pq}} \right)^2 - \left(\alpha_{pj} - \frac{\alpha_{pq} \alpha_{pj}}{\alpha_{pq}} \right)^2 \right] + \frac{\alpha_{pj}^2}{\alpha_{pq}^2} = \\ &= 1 + \sum_i \left(\alpha_{ij} - \frac{\alpha_{iq} \alpha_{pj}}{\alpha_{pq}} \right)^2 + \frac{\alpha_{pj}^2}{\alpha_{pq}^2} = 1 + \sum_i \alpha_{ij}^2 - 2 \sum_i \frac{\alpha_{ij} \alpha_{iq} \alpha_{pj}}{\alpha_{pq}} + \sum_i \frac{\alpha_{iq}^2 \alpha_{pj}^2}{\alpha_{pq}^2} + \frac{\alpha_{pj}^2}{\alpha_{pq}^2} = \\ &= \gamma_j + \frac{\alpha_{pj}^2}{\alpha_{pq}^2} \left(1 + \sum_i \alpha_{iq}^2 \right) - 2 \frac{\alpha_{pj}}{\alpha_{pq}} \sum_i \alpha_{ij} \alpha_{iq} = \gamma_j + \alpha_{pj}^2 \overline{\gamma}_q - 2 \frac{\alpha_{pj}}{\alpha_{pq}} \sum_i \alpha_{ij} \alpha_{iq}.\end{aligned}$$

Since $\sum_i \alpha_{ij} \alpha_{iq} = \widetilde{A}_j^T \widetilde{A}_q$ and $\widetilde{A}_j = -B^{-1} N_j$, we have:

$$\overline{\gamma}_j = \gamma_j + \alpha_{pj}^2 \overline{\gamma}_q + 2 \frac{\alpha_{pj}}{\alpha_{pq}} N_j^T B^{-T} \widetilde{A}_q, \ j \neq q. \tag{62}$$

It is seen that the updating formula (62) requires two additional BTRANs: the first is needed for computing elements α_{pj} of p-th (pivot) row of the matrix $\widetilde{A}$, and the second is needed for computing the vector $B^{-T} \widetilde{A}_q$.

The steepest edge technique can be aslo applied to the dual simplex method.

First of all we may note that the choice of basic variable in the dual simplex method is based on *residuals*

$$r_i = \begin{cases} (l_B)_i - \beta_i, & if \ \beta_i < (l_B)_i \\ \beta_i - (u_B)_i, & if \ \beta_i > (u_B)_i \end{cases} \tag{63}$$

where β_i is the current value of basic variable $(x_B)_i$, $(l_B)_i$ and $(u_B)_i$ are, respectively, lower and upper bounds of $(x_B)_i$.

From the linear programming theory it is known that in the dual simplex method the residuals r_i play the same role as the reduced costs d_j in the primal simplex method. And so far as the dual simplex method uses the transposed simplex table (33), we may expect that for choosing basic variable the following *scaled residuals*

$$r'_i = \frac{1}{\sqrt{\delta_i}} r_i \tag{64}$$

should be used instead the original residuals (63), where quantities

$$\delta_i = 1 + \sum_j \alpha_{ij}^2, \ i = 1, \ldots, m \tag{65}$$

are similar to γ_j (cf. (59)).

Again the formulae for updating the quantities δ_i can be derived from the explicit formulae for updating the simplex table (see Section 1.5).

So for the pivot row ($i = p$) we have

$$\overline{\delta}_p = 1 + \sum_j \overline{\alpha}_{pj}^2 = 1 + \sum_{j \neq q} \overline{\alpha}_{pj}^2 + \overline{\alpha}_{pq}^2 = 1 + \sum_{j \neq q} \frac{\alpha_{pj}^2}{\alpha_{pq}^2} + \frac{1}{\alpha_{pq}^2} =$$

$$= 1 + \left(\sum_j \frac{\alpha_{pj}^2}{\alpha_{pq}^2} - 1 \right) + \frac{1}{\alpha_{pq}^2} = \frac{1}{\alpha_{pq}^2} \left(1 + \sum_j \alpha_{pj}^2 \right) = \frac{\delta_p}{\alpha_{pq}^2}.$$

Therefore

$$\overline{\delta}_p = \frac{\delta_p}{\alpha_{pq}^2} = \frac{1}{\alpha_{pq}^2} \left(1 + \sum_j \alpha_{pj}^2 \right) = \frac{1}{\alpha_{pq}^2} (1 + \alpha_p^T \alpha_p). \tag{66}$$

where α_p is p-th row of the matrix $\widetilde{A}$.

Now let's consider updating quantities δ_i for other rows ($i \neq p$):

$$\overline{\delta}_i = 1 + \sum_j \overline{\alpha}_{ij}^2 = 1 + \sum_{j \neq q} \overline{\alpha}_{ij}^2 + \overline{\alpha}_{iq}^2 = 1 + \sum_{j \neq q} \left(\alpha_{ij} - \frac{\alpha_{iq} \alpha_{pj}}{\alpha_{pq}} \right)^2 + \frac{\alpha_{iq}^2}{\alpha_{pq}^2} =$$

$$= 1 + \left[\sum_j \left(\alpha_{ij} - \frac{\alpha_{iq} \alpha_{pj}}{\alpha_{pq}} \right)^2 - \left(\alpha_{iq} - \frac{\alpha_{iq} \alpha_{pq}}{\alpha_{pq}} \right)^2 \right] + \frac{\alpha_{iq}^2}{\alpha_{pq}^2} =$$

$$= 1 + \sum_j \left(\alpha_{ij} - \frac{\alpha_{iq} \alpha_{pj}}{\alpha_{pq}} \right)^2 + \frac{\alpha_{iq}^2}{\alpha_{pq}^2} = 1 + \sum_j \alpha_{ij}^2 - 2 \sum_j \frac{\alpha_{ij} \alpha_{iq} \alpha_{pj}}{\alpha_{pq}} + \sum_j \frac{\alpha_{iq}^2 \alpha_{pj}^2}{\alpha_{pq}^2} + \frac{\alpha_{iq}^2}{\alpha_{pq}^2} =$$

$$= \delta_i + \frac{\alpha_{iq}^2}{\alpha_{pq}^2} \left(1 + \sum_j \alpha_{pj}^2 \right) - 2 \frac{\alpha_{iq}}{\alpha_{pq}} \sum_j \alpha_{ij} \alpha_{pj} = \delta_i + \alpha_{iq}^2 \overline{\delta}_p - 2 \frac{\alpha_{iq}}{\alpha_{pq}} \sum_j \alpha_{ij} \alpha_{pj}.$$

Since $\sum_j \alpha_{ij} \alpha_{pj} = \alpha_i \alpha_p^T$, where α_i and α_p are, respectively, i-th and p-th (pivot) rows of the matrix $\widetilde{A}$, and $\alpha_i = e_i^T \widetilde{A} = -e_i^T B^{-1} N$, where e_i is the unity vector (see (41)), therefore $\alpha_i \alpha_p^T = -e_i^T (B^{-1} N \alpha_p^T) = -(B^{-1} N \alpha_p^T)_i$. Thus, finally we have:

$$\overline{\delta}_i = \delta_i + \alpha_{iq}^2 \overline{\delta}_p + 2 \frac{\alpha_{iq}}{\alpha_{pq}} (B^{-1} N \alpha_p^T)_i. \tag{67}$$

It is seen that the updating formula (67) requires two additional FTRANs: the first is needed for computing elements α_{iq} of q-th (pivot) column of the matrix $\widetilde{A}$, and the second is needed for computing the vector $B^{-1} N \alpha_p^T$.

1.7 Advanced ratio test based on Harris technique

Considering the step 5 of the primal simplex method (see Section 1.2) we may note that if *several* basic variables reach their bounds *at the same time*, the choice of basic variable becomes ambiguous. This phenomenon happens mainly in the case of *degenerative* basis solution, i.e. when current values of some basic variables are equal to their bounds. This ambiguity could be used in order to choose that non-basic variable $(x_B)_p$ which has more appropriate coefficient α_{pq} at the chosen non-basic variable $(x_N)_q$, because as it seen from the explicit formulae for updating the simplex table (see Section 1.5) this coefficient is used further as the pivot and therefore it affects on numerical stability. However, on practice in the case of degenerative basis solution current values of basic variables may slightly differ from thir bounds due to round-off errors. This may cause choosing basic variable which has unacceptably small (in absolute value) coefficint α_{pq}. For example, let there are two basic variables with lower bounds:

$$(x_B)_1 \geq 2$$
$$(x_B)_2 \geq 3$$

whose current values (computed with finite precision) are

$$\beta_1 = 2.00000$$
$$\beta_2 = 3.00001$$

and the fragment of the current simplex table that corresponds to the chosen non-basic variable is the following:

$$(x_B)_1 = \ldots - 0.00001(x_N)_q + \ldots$$
$$(x_B)_2 = \ldots - 5.00000(x_N)_q + \ldots$$

Let $(x_N)_q$ is increasing. Following the textbook ratio test we should choose the basic variable $(x_B)_1$. On the other hand it is seen that the basic variable $(x_B)_2$ is also on its lower bound, where small error is caused by possible round-off errors. Therefore if we would consider β_1 and β_2 as approximate quantities, but not an exact ones, we could choose the basic variable $(x_B)_2$ that is more desirable from the point of view of numerical stability.

P. Harris noted in her paper [3] that since all computed quantities are approximate and therefore in the case of degenerative basis solution current values of basic variables can slightly violate their bounds, we would exploit this in order to make more acceptable choice of basic variable. In order to achieve this purpose she proposed to choose basic variable using two-pass technique in the following way.

On the first pass we relax bounds of basic variables by small tolerance $\varepsilon > 0$ and determinte the following quantity (cf. the step 5 of the primal simplex method):

$$\widetilde{\theta} = \min[\min_i \widetilde{\theta}_i, (u_N)_q - (l_N)_q], \tag{68}$$

where $\widetilde{\theta}_i$ are computed using relaxed bounds:

a) if $(x_N)_q$ is increasing, i.e. if $d_q < 0$

$$\widetilde{\theta}_i = \begin{cases} \infty, & \alpha_{iq} = 0 \\ \{[(u_B)_i + \varepsilon] - \beta_i\}/\alpha_{iq}, & \alpha_{iq} > 0 \\ \{[(l_B)_i - \varepsilon] - \beta_i\}/\alpha_{iq}, & \alpha_{iq} < 0 \end{cases} \tag{69}$$

b) if $(x_N)_q$ is decreasing, i.e. if $d_q > 0$

$$\widetilde{\theta}_i = \begin{cases} \infty, & \alpha_{iq} = 0 \\ \{\beta_i - [(l_B)_i - \varepsilon]\}/\alpha_{iq}, & \alpha_{iq} > 0 \\ \{\beta_i - [(u_B)_i + \varepsilon]\}/\alpha_{iq}, & \alpha_{iq} < 0 \end{cases} \tag{70}$$

It is clear that the quantity $\widetilde{\theta}$ is the largest change of the non-basic variable $(x_N)_q$ on which all basic variables still remain within their relaxed bounds.

On the second pass we compute quantities θ_i using the formulae (25) and (26) for the original (not relaxed) bounds of basic variables and choose the basic variable $(x_B)_p$ for which

$$\theta_p \le \widetilde{\theta} \tag{71}$$

$$|\alpha_{pq}| = \max_i |\alpha_{iq}| \tag{72}$$

The condition (71) guarantees that all basic variables remain within their relaxed bounds and the condition (72) allows providing better numerical stability.

So if we apply this two-pass technique to the example considered above with, say, $\varepsilon = 10^{-4}$, we choose more acceptable basic variable $(x_B)_2$, and in the adjacent basis the value of other basic variable $(x_B)_1$ is equal to $2.00000 - 0.00001/5.00000 = 1.999998$. Although the basic variable $(x_B)_1$ violates its lower bound, this violation is so small that it vanishes after rounding the computed value to 6 decimal places.

It is understood that Harris two-pass technique can be also applied to the dual simplex method in the same way for choosing non-basic variable which has to enter the basis (see the step 5 in Section 1.3).

In her paper P. Harris points out that if her two-pass ratio test chooses a basic variable that differs from that one which would be chosen using the textbook ratio test, it provides larger value of θ_p (that involves faster convergence and sometimes helps to prevent cycling in cases of degenerative problems) and better value of the pivot coefficient α_{pq} (that involves better numerical stability).

1.8 Scaling LP problem

In the linear programming scaling means that we replace the original LP problem (1) by some equivalent *scaled* LP problem.

For scaling LP problem it is necessary to find some scaling matrices R and S (which are diagonal matrices with positive elements) in order that the scaled matrix of constraint coefficients

$$\widehat{A}' = R\widehat{A}S \tag{73}$$

would have better numerical properties than the original matrix $\widehat{A}$.

Let we already have some appropriate scaling matrices R and S. Using (73) we can transform the original LP problem (1) to the following equivalent scaled LP problem:

$$\begin{aligned} Z = (S\widehat{c})^T(S^{-1}x_S) + c_0 &\to \min\ (\max) \\ (Rx_R) = (R\widehat{A}S)(S^{-1}x_S) & \\ (Rl_R) \le (Rx_R) &\le (Ru_R) \\ (S^{-1}l_S) \le (S^{-1}x_S) &\le (S^{-1}u_S) \end{aligned} \tag{74}$$

Introducing *scaled* components we can express the scaled LP problem (74) in the same formulation as the original non-scaled problem (1):

$$\begin{aligned} Z &= (\widehat{c}')^T x'_S + c_0 \rightarrow \min(\max) \\ x'_R &= \widehat{A}' x'_S \\ l'_R &\leq x'_R \leq u'_R \\ l'_S &\leq x'_S \leq u'_S \end{aligned} \tag{75}$$

where

$$\begin{aligned} \widehat{A}' &= R\widehat{A}S \\ \widehat{c}' &= S\widehat{c} \\ x'_R &= Rx_R,\ l'_R = Rl_R,\ u'_R = Ru_R \\ x'_S &= S^{-1}x_S,\ l'_S = S^{-1}l_S,\ u'_S = S^{-1}u_S \end{aligned} \tag{76}$$

Then we can express the scaled LP problem (75) in the practical form (5)—(7) and solve it using the simplex method. After the scaled solution has been obtained we can recover the original non-scaled solution using the formulae (76).

2 Simplex Method Routines

2.1 Structure of common block

Most of GLPK routines implementing components of the revised simplex method use some common information which is organized in the form of the following structure:

```
typedef struct RSM RSM;
struct RSM
{     int m;
      int n;
      int type[1+m+n];
      double lb[1+m+n];
      double ub[1+m+n];
      MAT A[m:m+n];
      int posx[1+m+n];
      int indb[1+m];
      int indn[1+n];
      int tagn[1+n];
      EFI *efi;
      RFI *rfi;
      int iter;
};
```

where:

`m` number of auxiliary variables = number of basic variables = number of rows in the simplex table (except row of the objective function);

`n` number of structural variables = number of non-basic variables = number of columns in the simplex table;

`type` array of types of variables. Location `type[0]` is not used. Location `type[k]` specifies the type of variable $x_k, 1 \le k \le m + n$:

- `'F'` free (unbounded) variable ($-\infty < x_k < +\infty$);
- `'L'` variable with lower bound ($l_k \le x_k < +\infty$);
- `'U'` variable with upper bound ($-\infty < x_k \le u_k$);
- `'D'` double-bounded variable ($l_k \le x_k \le u_k$);
- `'S'` fixed variable ($l_k = x_k = u_k$);

`lb` array of lower bounds of variables. Location `lb[0]` is not used. Location `lb[k]` specifies the lower bound (or the fixed value) of variable $x_k, 1 \le k \le m + n$. If the variable x_k has no lower bound, `lb[k]` is equal to zero;

`ub` array of upper bounds of variables. Location `ub[0]` is not used. Location `ub[k]` specifies the upper bound of variable $x_k, 1 \le k \le m + n$. If the variable x_k has no upper bound, `ub[k]` is equal to zero. If the variable x_k is of fixed type, `ub[k]` is equal to `lb[k]`.

`A` expanded matrix of constraint coefficients (3). This matrix has m rows and $m + n$ columns. The first m columns correspond to auxiliary variables x_R and form the unity matrix. The last n columns correspond to structural variables x_S and form the original matrix of constraint coefficients $\widehat{A}$ with opposite sign;

`posx` array of positions of variables in the current basis solution. Location `posx[0]` is not used. If `posx[k]` $= +i > 0$, the variable x_k $(1 \leq k \leq m+n)$ is basic variable $(x_B)_i$. If `posx[k]` $= -j < 0$, the variable x_k is non-basic variable $(x_N)_j$.

`indb` array of original positions of basic variables. Location `indb[0]` is not used. The case `indb[i]` $= k$ means that the basic variable $(x_B)_i$ $(1 \leq i \leq m)$ is the original variable x_k, i.e. `posx[k]` $= +i$;

`indn` array of original positions of non-basic variables. Location `indn[0]` is not used. The case `indn[j]` $= k$ means that the non-basic variable $(x_N)_j$ $(1 \leq j \leq n)$ is the original variable x_k, i.e. `posx[k]` $= -j$;

`tagn` array of tags of non-basic variables. Location `tagn[0]` is not used. Location `tagn[j]` specifies which subset (18) the non-basic variable $(x_N)_j$ $(1 \leq j \leq n)$ belongs to:

- `'L'` non-basic variable on its lower bound, $(x_N)_j \in x_L$;
- `'U'` non-basic variable on its upper bound, $(x_N)_j \in x_U$;
- `'F'` non-basic free variable, $(x_N)_j \in x_F$;
- `'S'` non-basic fixed variable, $(x_N)_j \in x_S$;

`efi` if EFI (Elimination form of the Inverse) is used to represent the current basis matrix B, `efi` points to EFI (see [1]). Otherwise the `efi` field contains `NULL`;

`rfi` if RFI (Reid's form of the Inverse) is used to represent the current basis matrix B, `rfi` points to RFI (see[1]). Otherwise the `rfi` field contains `NULL`;

`iter` iteration count. This value is increased by one each time when the current basis is replaced by the adjacent one.

Remarks.

1. It is implicitly assumed that the united vector (2) contains variables of the LP problem (1) in the following order $x = [(x_R)_1, \ldots (x_R)_m, (x_S)_1, \ldots, (x_S)_n]$, where $(x_R)_i$ are auxiliary variables (that correspond to the equality constraints) and $(x_S)_j$ are structural variables. The same ordering is assumed for columns of the expanded matrix A.
2. It is implicitly assumed that the order of columns in the current basis matrix B is determined by the order of variables in the vector x_B, i.e. by the array `indb`.
3. It is implicitly assumed that all other arrays used by GLPK routines and which contain additional information related to basic or non-basic variables have the same order of elements as defined by the array `posx`, `indb`, and `indn`.

2.2 Operations on basis matrix

This section describes routines which are intended for performing operations on the basis matrix (see also Sections 1.1 and 1.4).

These routines are an interface to partucular routines which actually perform the corresponding operations (see chapters GLPEFI and GLPRFI in [1]).

2.2.1 `invert_b` — rebuild representation of the basis matrix

Synopsis

```
#include "glprsm.h"
int invert_b(RSM *rsm);
```

Description

The `invert_b` routine reinverts the basis matrix B, i.e. it rebuilds anew some representation of the basis matrix.

This operation can be used in the following three cases:

a) if it is necessary to rebuild the representation for some initial basis matrix;

b) if the representation of the current basis matrix has become too large or inaccurate;

c) if the basis matrix has been completely changed.

The `invert_b` routine assumes that the array `rsm.indb` specifies what columns of the expanded matrix A belong to the basis matrix B.

Returns

If the operation was successful, the `invert_b` routine returns zero. Otherwise the routine returns non-zero. The latter case means that the basis matrix is numerically singular or ill-conditioned. For further information see descriptions of particular routines that perform this operation [1].

2.2.2 `ftran` — perform forward transformation

Synopsis

```
#include "glprsm.h"
void ftran(RSM *rsm, double u[], int save);
```

Description

The `ftran` routine performs forward transformation (FTRAN) of the vector u using some representation of the current basis matrix.

This operation means solving the system $Bu' = u$, where B is the current basis matrix, u is the given vector which should be transformed, u' is the resultant vector.

On entry the array `u` should contain elements of the vector u in locations `u[1]`, ..., `u[m]`, where m is order of the matrix B. On exit this array contains elements of the vector u' in the same locations.

The parameter `save` is a flag. If this flag is set, it means that the vector u is the column of non-basic variable $(x_N)_q$ which has been chosen to enter the basis (i.e. $u = N_q$; see the step 6 of primal and dual simplex methods in Sections 1.2 and 1.3). In this case `ftran` routine saves some internal information which will be used further by the `update_b` routine (see Subsection 2.2.4) in order to update the representation of the basis matrix for the adjacent basis. It is assumed that the calling program should perform at least one call to the `ftran` routine with the `save` flag set before subsequent call to the `update_b` routine.

2.2.3 `btran` — perform backward transformation

Synopsis

```
#include "glprsm.h"
void btran(RSM *rsm, double u[]);
```

Description

The `btran` routine performs backward transformation (BTRAN) of the vector u using some representation of the current basis matrix.

This operation means solving the system $B^T u' = u$, where B is the current basis matrix, u is the given vector which should be transformed, u' is the resultant vector.

On entry the array `u` should contain elements of the vector u in locations `u[1]`, ..., `u[m]`, where m is order of the matrix B. On exit this array contains elements of the vector u' in the same locations.

2.2.4 `update_b` — update representation of the basis matrix

Synopsis

```
#include "glprsm.h"
int update_b(RSM *rsm, int p);
```

Description

Let p-th column of the current basis matrix B has been replaced by some other column that gives the new (adjacent) basis matrix $\overline{B}$ (see the step 6 of primal and dual simplex methods in Sections 1.2 and 1.3). The `update_b` routine updates some representation of the basis matrix B in order that the updated representation corresponds to the matrix $\overline{B}$.

The new p-th column of the basis matrix is passed implcitly to the `update_b` routine. It is assumed that this column has been saved before by the `ftran` routine (see Subsection 2.2.2).

Returns

The `update_b` routine returns one of the following error codes:

0 The representation has been successfully updated;

1 The representation has become inaccurate;

2 The representation has become too large.

If the returned code is non-zero, the calling program should rebuild the representation anewby means of the `invert_b` routine (see Subsection 2.2.1).

2.3 Operations on simplex table

This section describes routines which are intended for performing operations on the simplex table (see also Sections 1.1 and 1.4).

Implicit representation of the simplex table is used.

2.3.1 `check_rsm` — check common block for correctness

Synopsis

```
#include "glprsm.h"
void check_rsm(RSM *rsm)
```

Description

The `check_rsm` routine checks the revised simplex method common block, which `rsm` points to, for correctness. In case of error the routine displays an appropriate error message and abnormally terminates the program.

Note that the `check_rsm` routine doesn't check the expanded constraint matrix A, because the corresponding operation is extremely inefficient. This matrix can be checked additionally by means of the `check_mat` routine (see [1]).

The `check_rsm` routine is intended for debugging purposes.

2.3.2 `eval_xn` — determine value of non-basic variable

Synopsis

```
#include "glprsm.h"
double eval_xn(RSM *rsm, int j);
```

Returns

The `eval_xn` routine returns the value of non-basic variable $(x_N)_j$, $1 \leq j \leq n$, that corresponds to the current basis solution.

The returned value is determined by the formula (20) (see Section 1.1).

2.3.3 `eval_bbar` — compute values of basic variables

Synopsis

```
#include "glprsm.h"
void eval_bbar(RSM *rsm, double bbar[]);
```

Description

The `eval_bbar` routine computes values of basic variables $x_B = \beta = (\beta_1, \ldots, \beta_m)$ that correspond to the current basis solution and stores $\beta_1, \ldots, \beta_m$ into locations `bbar[1]`, ..., `bbar[m]` respectively.

The vector β is computed using the formula (38) (see Section 1.4).

2.3.4 `eval_pi` — compute simplex multipliers

Synopsis

```
#include "glprsm.h"
void eval_pi(RSM *rsm, double c[], double pi[]);
```

Description

The `eval_pi` routine computes simplex multipliers $\pi = (\pi_1, \ldots, \pi_m)$, i.e. Lagrange multipliers for the equality constraints (6), that correspond to the current basis and stores $\pi_1, \ldots, \pi_m$ into locations `pi[1]`, ..., `pi[m]` respectively.

On entry the array `c` should contain the expanded vector of coefficients of the objective function c (see (4)) in locations `c[1]`, ..., `c[m+n]`. The array `c` is not changed on exit.

The vector π is computed using the formula (39) (see Section 1.4).

2.3.5 `eval_cbar` — compute reduced costs of non-basic variables

Synopsis

```
#include "glprsm.h"
void eval_cbar(RSM *rsm, double c[], double pi[], double cbar[]);
```

Description

The `eval_cbar` routine computes reduced costs $d = (d_1, \ldots, d_n)$ of non-basic variables that correspond to the current basis solution and stores $d_1, \ldots, d_n$ into locations `cbar[1]`, ..., `cbar[n]` respectively.

On entry the array `c` should contain the expanded vector of coefficients of the objective function c (see (4)) in locations `c[1]`, ..., `c[m+n]`. The array `c` is not changed on exit.

On entry the array `pi` should contain the vector of simplex multipliers π computed by means of the `eval_pi` routine (see Subsection 2.3.4) for the same vector c. The array `pi` is not changed on exit.

The vector d is computed using the formula (40) (see Section 1.4).

2.3.6 `check_rr` — check relative residual

Synopsis

```
#include "glprsm.h"
int check_rr(double x, double x0, double tol);
```

Description

The `check_rr` routine checks relative residual between the computed quantity x and the given quantity x_0 using the relative tolerance $tol > 0$.

Returns

The `check_rr` routine returns one of the following codes:

–2 if $x < x_0 - \varepsilon$;

–1 if $x_0 - \varepsilon \leq x < x0$;

0 if $x = x_0$;

+1 if $x_0 < x \leq x_0 + \varepsilon$;

+2 if $x > x_0 + \varepsilon$;

where $\varepsilon = tol \cdot \max(1, |x_0|)$.

2.3.7 `check_bbar` — check basis solution for primal feasibility

Synopsis

```
#include "glprsm.h"
int check_bbar(RSM *rsm, double bbar[], double tol);
```

Description

The `check_bbar` routine checks the given basis solution for primal feasibility.

The array `bbar` should contain the given values of basic variables $\beta = (\beta_1, \ldots, \beta_m)$ in locations `bbar[1]`, ..., `bbar[m]` respectively.

The parameter tol > 0 is a relative tolerance.

In order to see if the given basis solution is primal feasible, the routine checks the primal feasibility conditions (21) (see Section 1.1) which using the relative tolerance are the following:

$$(l_B)_i - \varepsilon \leq \beta_i \leq (u_B)_i + \varepsilon, \ i = 1, \ldots, m,$$

where

$$\varepsilon = \begin{cases} tol \cdot \max(1, |(l_B)_i|) & \textit{in case of lower bound} \\ tol \cdot \max(1, |(u_B)_i|) & \textit{in case of upper bound} \end{cases}$$

Returns

If the given basis solution is primal feasible (i.e. it satisfies to the conditions above), the `check_bbar` routine returns zero. Otherwise the routine returns non-zero.

2.3.8 `check_cbar` — check basis solution for dual feasibility

Synopsis

```
#include "glprsm.h"
int check_cbar(RSM *rsm, double c[], double cbar[], double tol);
```

Description

The `check_cbar` routine checks the given basis solution for dual feasibility.

The array `c` should contain the expanded vector c of coefficients of the objective function (4) in locations `c[1]`, ..., `c[m+n]`.

The array `cbar` should contain the given reduced costs $d = (d_1, \ldots, d_n)$ of non-basic variables in locations `cbar[1]`, ..., `cbar[n]` respectively. It is assumed that the reduced costs are computed using the vector c passed to this routine.

The parameter $tol > 0$ is a relative tolerance.

In order to see if the given basis solution is dual feasible, the routine checks the dual feasibility conditions (22) (see Section 1.1) which using the relative tolerance are the following:

$$\begin{aligned}
&if\ (x_N)_j \in x_F\ then\ \ -\varepsilon \le d_j \le +\varepsilon\\
&if\ (x_N)_j \in x_L\ then\ d_j \ge -\varepsilon\\
&if\ (x_N)_j \in x_U\ then\ d_j \le +\varepsilon\\
&if\ (x_N)_j \in x_S\ then\ \ -\infty < d_j < +\infty
\end{aligned}$$

where

$$\varepsilon = tol \cdot \max(1, |(c_N)_j|)$$

(the absolute tolerance ε reflects that the reduced cost of non-basic variable $(x_N)_j$ is the difference $d_j = (c_N)_j - N_j^T \pi$ as it follows from the formula (40) in Section 1.4).

Returns

If the given basis solution is dual feasible (i.e. it satisfies to the conditions above), the `check_cbar` routine returns zero. Otherwise the routine returns non-zero.

2.3.9 `eval_col` — compute column of the simplex table

Synopsis

```
#include "glprsm.h"
void eval_col(RSM *rsm, double j, double aj[], int save);
```

Description

The `eval_col` routine computes j-th column of the simplex table $\widetilde{A}_j = (\alpha_{1j}, \ldots, \alpha_{mj})$, i.e. j-th column of the matrix $\widetilde{A}$ (see (17), Section 1.1) and stores its elements into locations `aj[1]`, ..., `aj[m]` respectively.

The parameter `save` is a flag. If this flag is set, it means that the column $\widetilde{A}_j$ is the column of non-basic variable $(x_N)_q$ which has been chosen to enter the basis (i.e. $j = q$). This flag is passed to the `ftran` routine (see Section 2.2.2) which is called by the `eval_col` routine in order to perform forward transformation.

The column $\widetilde{A}_j$ is computed using the formula (42) (see Section 1.4).

2.3.10 `eval_zeta` — compute row of the inverse

Synopsis

```
#include "glprsm.h"
void eval_zeta(RSM *rsm, int i, double zeta[]);
```

Description

The `eval_zeta` routine computes i-th row of the inverse $\zeta_i = (\zeta_{i1}, \ldots, \zeta_{im})$, i.e. i-th row of the matrix B^{-1}, where B is the current basis matrix, and stores its elements into locations `zeta[1]`, ..., `zeta[m]`.

The vector ζ_i is computed using the formula:

$$\zeta_i = B^{-T} e_i,$$

where e_i is the unity vector (41).

2.3.11 `eval_row` — compute row of the simplex table

Synopsis

```
#include "glprsm.h"
void eval_row(RSM *rsm, double zeta[], double ai[]);
```

Description

The `eval_row` routine computes i-th row of the simplex table $\alpha_i = (\alpha_i 1, \ldots, \alpha_i n)$, i.e. i-th row of the matrix $\widetilde{A}$ (see (17), Section 1.1), and stores its elements into locations `ai[1]`, ..., `ai[n]` respectively.

The array `zeta` should contain i-th row of the inverse $\zeta_i = (\zeta_{i1}, \ldots, \zeta_{im})$, i.e. i-th row of the matrix B^{-1}, where B is the current basis matrix, computed by means of the `eval_zeta` routine (Subsection 2.3.11). This array is not changed on exit.

The row α_i is computed using the formula (43) (see Section 1.4).

2.3.12 `change_b` — change basis

Synopsis

```
#include "glprsm.h"
int change_b(RSM *rsm, int p, int tagp, int q);
```

Description

The `change_b` routine changes the current basis replacing it by the adjacent one. The routine takes the basic variable $(x_B)_p$ $(1 \le p \le m)$ out the set of basic variables x_B and brings instead it the non-basic variable $(x_N)_q$ $(1 \le q \le n)$ into x_B. Correspondingly, the basic variable $(x_B)_p$ replaces the non-basic variable $(x_N)_q$ in the set of non-basic variables x_N.

The parameter `tagp` specifies to what subset x_F, x_L, x_U, or x_S (see (18), Section 1.1) the basic variable $(x_B)_p$ should be attributed after it has left the basis. The value of `tagp` has the same meaning as the field `rsm.tagn` (see Section 2.1) and should be compatible with the type of variable $(x_B)_p$.

The special case $p < 0$ means that the current basis is not changed, but the non-basic variable $(x_N)_q$ (which should be double-bounded variable) just goes from its current bound to the opposite one. The parameter `tagp` is ignored in this special case.

The `change_b` routine also replaces p-th column of the basis matrix B by q-th column of the matrix N and updates the representation of B by means of `update_b` routine (Section 2.2.4). Note that new p-th column of B is passed implcitly; it is assumed that this column was saved before by the `eval_col` routine (Section 2.3.9). If the representation becomes too long or inaccurate, the `change_b` routine automatically rebuilds it by means of the `invert_b` routine (Section 2.2.1).

One call to the `change_b` routine is considered as one iteration of the simplex method.

Returns

If the representation of new basis matrix was not rebuilt or if it was rebuilt successfully, the `change_b` routine returns zero. Otherwise the routine returns non-zero. In the latter case the calling program should not use the representation until the basis will be corrected and the representation will be rebuilt anew.

2.4 Primal steepest edge routines

This section describes routines which implement primal steepest edge technique (see Section 1.6) and which are intended for using in conjunction with primal simplex method routines (see Section 2.6).

2.4.1 `init_gvec` — initialize the vector γ

Synopsis

```
#include "glprsm.h"
void init_gvec(RSM *rsm, double gvec[]);
```

Description

The `init_gvec` routine computes the vector γ (see (59), Section 1.6) for the initial basis and stores its elements into locations `gvec[1]`, ..., `gvec[n]`.

Initial basis is a basis, where all auxiliary variables are basic and all structural variables are non-basic. In this special case $B = I$ and $N = \widehat{A}$, therefore $\alpha_{ij} = \widehat{a}_{ij}$, where $\widehat{a}_{ij}$ are elements of the original constraint matrix $\widehat{A}$, that allows using the formula (59) directly for computing the initial vector γ.

2.4.2 `update_gvec` — update the vector γ

Synopsis

```
#include "glprsm.h"
void update_gvec(RSM *rsm, double gvec[], int p, int q, double ap[],
                 double aq[], double w[]);
```

Description

The `update_gvec` routine replaces the vector γ (see (59), Section 1.6) which corresponds to the current basis by the updated vector $\overline{\gamma}$ which corresponds to the adjacent basis.

On entry the array `gvec` should contain elements of the vector $\gamma = (\gamma_1, \ldots, \gamma_n)$ for the current basis in locations `gvec[1]`, ..., `gvec[n]` respectively. On exit this array will contain elements of the vector $\overline{\gamma} = (\overline{\gamma}_1, \ldots, \overline{\gamma}_n)$ for the adjacent basis in the same locations.

The parameter `p` specifies basic variable $(x_B)_p$ which has been chosen to leave the basis.

The parameter `q` specifies non-basic variable $(x_N)_q$ which has been chosen to enter the basis.

On entry the array `ap` should contain elements of p-th row of the current simplex table α_p that corresponds to basic variable $(x_B)_p$ in locations `ap[1]`, ..., `ap[n]`. This array can be computed by means of the `eval_row` routine (Section 2.3.11). It is not changed on exit.

On entry the array `aq` should contain elements of q-th column of the current simplex table $\widetilde{A}_q$ that corresponds to non-basic variable $(x_N)_q$ in locations `aq[1]`, ..., `aq[m]`. This array can be computed by means of the `eval_col` routine (Section 2.3.9). It is not changed on exit.

The working array `w` should have at least $1+m$ locations, where m is order of the basis matrix B.

The `update_gvec` routine assumes that the representation of the basis matrix corresponds to the current basis, not to the adjacent one. Therefore this routine should be called before changing the basis.

For updating the vector γ the `update_gvec` routine uses the following formulae which can be easily derived from the formulae (61) and (62) given in Section 1.6 (dashed symbols denote quantities related to the adjacent basis):

$$\gamma_q = 1 + \sum_{i=1}^{m} \alpha_{iq}^2 \qquad (a)$$

$$w = B^{-T} \widetilde{A}_q \qquad (b)$$

$$\overline{\alpha}_{pj} = -\frac{\alpha_{pj}}{\alpha_{pq}} \qquad (c)$$

$$t_j = N_j^T w \qquad (d)$$

$$\overline{\gamma}_j = \max(\gamma_j + \overline{\alpha}_{pj}^2 - 2\overline{\alpha}_{pj} t_j, 1 + \overline{\alpha}_{pj}^2) \qquad (e)$$

$$\overline{\gamma}_q = \frac{1}{\alpha_{pq}^2} \gamma_q \qquad (f)$$

The formula (a) is used directly to compute exact value of γ_q, because q-th (pivot) column of the simplex table is available in the case of primal simplex method.

The formula (b) is used once for computing auxiliary vector w.

Formulae (c), (d), and (e) are used for computing all $\overline{\gamma}_j$ (except $\overline{\gamma}_q$). If due to round-off errors the computed value of $\overline{\gamma}_j$ becomes too small, it is replaced by $1 + \overline{\alpha}_{pj}^2$, because from the definition (59) it is seen that $\overline{\gamma}_j \geq 1 + \overline{\alpha}_{pj}^2$.

Finally, the formula (g) is used to compute exact value of $\overline{\gamma}_q$.

2.4.3 `exact_gvec` — compute exact value of γ

Synopsis

```
#include "glprsm.h"
double exact_gvec(RSM *rsm, int j);
```

Description

The `exact_gvec` routine computes exact value of γ_j using the formula (59) directly.

This operation is extremely inefficient and may be used only for debugging purposes.

Returns

The `exact_gvec` routine returns the computed value of γ_j.

2.4.4 `check_gvec` — check accuracy of the vector γ

Synopsis

```
#include "glprsm.h"
double check_gvec(RSM *rsm, double gvec[]);
```

Description

The `check_gvec` routine is intended for checking accuracy of the vector γ. It computes the absolute error

$$e = \max_j |\gamma_j^* - \gamma_j|,$$

where γ^* is exact vector computed by means of the `exact_gvec` routine (Section 2.4.3), γ is approximate vector given in the array `gvec`.

This operation is extremely inefficient and may be used only for debugging purposes.

Returns

The `check_gvec` routine returns the absolute error e (see above).

2.5 Dual steepest edge routines

This section describes routines which implement dual steepest edge technique (see Section 1.6) in the same way as primal steepest edge routines (see Section 2.4) and which are intended for using in conjunction with dual simplex method routines (see Section 2.7).

2.5.1 `init_dvec` — initialize the vector δ

Synopsis

```
#include "glprsm.h"
void init_dvec(RSM *rsm, double dvec[]);
```

Description

The `init_dvec` routine computes the vector δ (see (65), Section 1.6) for the initial basis and stores its elements into locations `dvec[1]`, ..., `dvec[m]`.

Initial basis is a basis, where all auxiliary variables are basic and all structural variables are non-basic. In this special case $B = I$ and $N = \widehat{A}$, therefore $\alpha_{ij} = \widehat{a}_{ij}$, where $\widehat{a}_{ij}$ are elements of the original constraint matrix $\widehat{A}$, that allows using the formula (65) directly for computing the initial vector δ.

2.5.2 `update_dvec` — update the vector δ

Synopsis

```
#include "glprsm.h"
void update_dvec(RSM *rsm, double dvec[], int p, int q, double ap[],
                 double aq[], double w[]);
```

Description

The `update_dvec` routine replaces the vector δ (see (65), Section 1.6) which corresponds to the current basis by the updated vector $\overline{\delta}$ which corresponds to the adjacent basis.

On entry the array `dvec` should contain elements of the vector $\delta = (\delta_1, \ldots, \delta_m)$ for the current basis in locations `dvec[1]`, ..., `dvec[m]` respectively. On exit this array will contain elements of the vector $\overline{\delta} = (\overline{\delta}_1, \ldots, \overline{\delta}_m)$ for the adjacent basis in the same locations.

The parameter `p` specifies basic variable $(x_B)_p$ which has been chosen to leave the basis.

The parameter `q` specifies non-basic variable $(x_N)_q$ which has been chosen to enter the basis.

On entry the array `ap` should contain elements of p-th row of the current simplex table α_p that corresponds to basic variable $(x_B)_p$ in locations `ap[1]`, ..., `ap[n]`. This array can be computed by means of the `eval_row` routine (Section 2.3.11). It is not changed on exit.

On entry the array `aq` should contain elements of q-th column of the current simplex table $\widetilde{A}_q$ that corresponds to non-basic variable $(x_N)_q$ in locations `aq[1]`, ..., `aq[m]`. This array can be computed by means of the `eval_col` routine (Section 2.3.9). It is not changed on exit.

The working array `w` should have at least $1+m$ locations, where m is order of the basis matrix B.

The `update_dvec` routine assumes that the representation of the basis matrix corresponds to the current basis, not to the adjacent one. Therefore this routine should be called before changing the basis.

For updating the vector δ the `update_dvec` routine uses the following formulae which can be easily derived from the formulae (66) and (67) given in Section 1.6 (dashed symbols denote quantities related to the adjacent basis):

$$\delta_p = 1 + \sum_{j=1}^{n} \alpha_{pj}^2 \qquad (a)$$

$$w = B^{-1} N \alpha_p^T \qquad (b)$$

$$\overline{\alpha}_{iq} = +\frac{\alpha_{iq}}{\alpha_{pq}} \qquad (c)$$

$$\overline{\delta}_i = \max(\delta_i + \overline{\alpha}_{iq}^2 \delta_p + 2\overline{\alpha}_{iq} w_j, 1 + \overline{\alpha}_{iq}^2) \qquad (d)$$

$$\overline{\delta}_p = \frac{1}{\alpha_{pq}^2} \delta_p \qquad (e)$$

The formula (a) is used directly to compute exact value of δ_p, because p-th (pivot) row of the simplex table is available in the case of dual simplex method.

The formula (b) is used once for computing auxiliary vector w.

Formulae (c) and (d) are used for computing all $\overline{\delta}_i$ (except $\overline{\delta}_p$). If due to round-off errors the computed value of $\overline{\delta}_i$ becomes too small, it is replaced by $1 + \overline{\alpha}_{iq}^2$, because from the definition (65) it is seen that $\overline{\delta}_i \geq 1 + \overline{\alpha}_{iq}^2$.

Finally, the formula (e) is used to compute exact value of $\overline{\delta}_p$.

2.5.3 `exact_dvec` — compute exact value of δ

Synopsis

```
#include "glprsm.h"
double exact_dvec(RSM *rsm, int i);
```

Description

The `exact_dvec` routine computes exact value of δ_i using the formula (65) directly.

This operation is extremely inefficient and may be used only for debugging purposes.

Returns

The `exact_dvec` routine returns the computed value of δ_i.

2.5.4 `check_dvec` — check accuracy of the vector δ

Synopsis

```
#include "glprsm.h"
double check_dvec(RSM *rsm, double dvec[]);
```

Description

The `check_dvec` routine is intended for checking accuracy of the vector δ. It computes the absolute error

$$e = \max_i |\delta_i^* - \delta_i|,$$

where δ^* is exact vector computed by means of the `exact_dvec` routine (Section 2.5.3), δ is approximate vector given in the array `dvec`.

This operation is extremely inefficient and may be used only for debugging purposes.

Returns

The `check_dvec` routine returns the absolute error e (see above).

2.6 Primal simplex method routines

2.6.1 `pivot_col` — choose non-basic variable (primal)

Synopsis

```
#include "glprsm.h"
int pivot_col(RSM *rsm, double c[], double cbar[], double gvec[],
              double tol);
```

Description

The `pivot_col` routine chooses non-basic variable $(x_N)_q$ (i.e. pivot column of the simplex table) which should enter the basis on the next iteration of the primal simplex method. Note that the routine assumes that the objective function should be *minimized*.

The array `c` should contain the expanded vector $c = (c_1, \ldots, c_{m+n})$ of coefficients of the objective function in locations `c[1]`, ..., `c[m+n]`. This array is not changed on exit.

The array `cbar` should contain reduced costs $d = (d_1, \ldots, d_n)$ of non-basic variables in locations `cbar[1]`, ..., `cbar[n]`. This array is not changed on exit.

The array `gvec` should contain the vector $\gamma = (\gamma_1, \ldots, \gamma_n)$ (see (59), Section 1.6) in locations `gvec[1]`, ..., `gvec[n]`. This array is not changed on exit. It is allowed to specify `NULL` instead the array `gvec`; in this case the routine assumes that all elements of the vector γ are equal to one.

The parameter `tol` is a relative tolerance (see below).

The `pivot_col` routine considers only those non-basic variables which changing in feasible direction can improve (decrease) the objective function, i.e. for which

$$\begin{aligned}
&(x_N)_j \in x_F \ and \ |d_j| > \varepsilon, \ or \\
&(x_N)_j \in x_L \ and \ d_j < -\varepsilon, \ or \\
&(x_N)_j \in x_U \ and \ d_j > +\varepsilon
\end{aligned}$$

where

$$\varepsilon = tol \cdot \max(1, |(c_N)_j|)$$

(the absolute tolerance ε reflects that the reduced cost of non-basic variable $(x_N)_j$ is the difference $d_j = (c_N)_j - N_j^T \pi$ as it follows from (40) in Section 1.4).

The routine chooses the non-basic variable $(x_N)_q$ which has the largest (in absolute value) scaled reduced cost (60). Thus, if the vector γ is not used, the choice made by the routine corresponds to the textbook technique. Otherwise the choice corresponds to the steepest edge technique.

Returns

If the choice has been made, the `pivot_col` routine returns q which is a number of the chosen non-basic variable $(x_N)_q, 1 \le q \le n$. Otherwise, if the current basis solution is dual feasible and the choice is impossible, the routine returns zero.

2.6.2 `pivot_row` — choose basic variable (primal, standard technique)

Synopsis

```
#include "glprsm.h"
int pivot_row(RSM *rsm, int q, int dir, double aq[], double bbar[],
              int *tagp, double tol);
```

Description

The `pivot_row` routine chooses basic variable $(x_B)_p$ (i.e. pivot row of the simplex table) which should leave the basis on the next iteration of the primal simplex method. The routine is based on the standard (textbook) ratio test.

The parameter `q` specifies a number of the non-basic variable $(x_N)_q$ which has been chosen to enter the basis, $1 \le q \le n$.

The parameter `dir` specifies in what direction the non-basic variable $(x_N)_q$ is changing:

0, if $(x_N)_q$ is increasing;

1, if $(x_N)_q$ is decreasing.

The array `aq` should contain q-th (pivot) column $\widetilde{A}_q = (\alpha_{1q}, \ldots, \alpha_{mq})$ of the simplex table, i.e. q-th column of the matrix $\widetilde{A}$ (see (17), Section 1.1), in locations `aq[1]`, ..., `aq[m]`. This array is not changed on exit.

The array `bbar` should contain current values $\beta = (\beta_1, \ldots, \beta_m)$ of basic variables x_B in locations `bbar[1]`, ..., `bbar[m]`. This array is not changed on exit.

If the routine chooses some basic variable $(x_B)_p$, it stores into location `tagp` a tag that specifies to what subset x_L, x_U, or x_S (see (18), Section 1.1) the basic variable should be attributed after it has left the basis. This tag has the same meaning as the field `rsm.tagn` (see Section 2.1).

The parameter `tol` is a relative tolerance (see below).

The `pivot_row` routine implements the standard (textbook) ratio test for choosing basic variable as described in Section 1.2 (see step 5). i.e. the routine determines that basic variable which reaches its (lower or upper) bound first when the non-basic variable $(x_N)_q$ is changing in the feasible direction. Besides, the following additional rules are used:

if $|\alpha_{iq}| < tol \cdot \max_i |\alpha_{iq}|$, i.e. if the influence coefficient α_{iq} is relatively close to zero, it is assumed that the corresponding basic variable $(x_B)_i$ doesn't depend on the non-basic variable $(x_N)_q$ and therefore such basic variable is not considered to be chosen;

if current value β_i of some basic variable $(x_B)_i$ violates its bound, it is assumed that this happens due to round-off errors and actually the basic variable is exactly on its bound (because the current basis solution should be primal feasible);

if several basic variables reach their bounds first at the same time, the routine prefers that variable which has the largest (in absolute value) influence coefficient.

Returns

If the choice has been made, the `pivot_row` routine returns p which is a number of the chosen basic variable $(x_B)_p, 1 \le p \le m$. In the special case, if the non-basic variable $(x_N)_q$ being double-bounded variable reaches its opposite bound before any basic variable, the routine returns a negative value. Otherwise, if the chosen non-basic variable $(x_N)_q$ can unlimitedly change in the feasible direction and therefore the choice is impossible, the routine returns zero.

2.6.3 `harris_row` — choose basic variable (primal, Harris technique)

Synopsis

```
#include "glprsm.h"
int harris_row(RSM *rsm, int q, int dir, double aq[], double bbar[],
              int *tagp, double tol, double tol1);
```

Description

The `harris_row` routine chooses basic variable $(x_B)_p$ (i.e. pivot row of the simplex table) which should leave the basis on the next iteration of the primal simplex method. The routine is based on the two-pass ratio test proposed by P. Harris.

The `harris_row` routine has the same program specification as the `pivot_row` routine (see Subsection 2.6.2).

The only difference is the additional parameter `tol1` which is a relative tolerance used for relaxing bounds of basic variables on the first pass. The routine replaces original bounds of basic variables by their relaxed bounds:

$$(l_B)_i - \varepsilon \le (x_B)_i \le (u_B)_i + \varepsilon,$$

where

$$\varepsilon = \begin{cases} tol1 \cdot \max(1, |(l_B)_i|) & \textit{in case of lower bound} \\ tol1 \cdot \max(1, |(u_B)_i|) & \textit{in case of upper bound} \end{cases}$$

For futher details see Section 1.7.

2.7 Dual simplex method routines

2.7.1 `dual_row` — choose basic variable (dual)

Synopsis

```
#include "glprsm.h"
int dual_row(RSM *rsm, double bbar[], double dvec[], int *tagp,
             double tol);
```

Description

The `dual_row` routine chooses basic variable $(x_B)_p$ (i.e. pivot row of the simplex table) which should leave the basis on the next iteration of the dual simplex method.

The array `bbar` should contain current values $\beta = (\beta_1, \ldots, \beta_m)$ of basic variables x_B in locations `bbar[1]`, ..., `bbar[m]`. This array is not changed on exit.

The array `dvec` should contain the vector $\delta = (\delta_1, \ldots, \delta_m)$ (see (65), Section 1.6) in locations `dvec[1]`, ..., `dvec[m]`. This array is not changed on exit. It is allowed to specify `NULL` instead the array `dvec`; in this case the routine assumes that all elements of the vector δ are equal to one.

If the routine chooses some basic variable $(x_B)_p$, it stores into location `tagp` a tag that specifies to what subset x_L or x_U (see (18), Section 1.1) the basic variable should be

attributed after it has left the basis. This tag has the same meaning as the field `rsm.tagn` (see Section 2.1). Thus, the subset x_L means that the basic variable $(x_B)_p$ violates its lower bound and therefore it should be set on its lower bound. Analogously, the subset x_U means that the basic variable $(x_B)_p$ violates its upper bound and therefore it should be set on its upper bound. Note that if the basic variable $(x_B)_p$ is of fixed type, it is considered as double-bounded variable (with lower bound equal to upper bound), therefore its tag should be corrected before changing the basis by means of the `change_b` routine (see Subsection 2.3.12). This exception is used in order to let other routines know what to do with such fixed basic variable: increase (in case of x_L) or decrease (in case of x_U).

The parameter `tol` is a relative tolerance (see below).

The `dual_row` routine considers only those basic variables which violates their bounds:

$$\beta_i < (l_B)_i - \varepsilon \quad or$$
$$\beta_i > (u_B)_i + \varepsilon$$

where

$$\varepsilon = \begin{cases} tol \cdot \max(1, |(l_B)_i|) & in\ case\ of\ lower\ bound \\ tol \cdot \max(1, |(u_B)_i|) & in\ case\ of\ upper\ bound \end{cases}$$

The routine chooses the basic variable $(x_B)_p$ which has the largest (in absolute value) scaled residual (64). Thus, if the vector δ is not used, the choice made by the routine corresponds to the textbook technique. Otherwise the choice corresponds to the steepest edge technique.

Returns

If the choice has been made, the `dual_row` routine returns p which is a number of the chosen basic variable $(x_B)_p, 1 \le p \le m$. Otherwise, if the current basis solution is primal feasible and the choice is impossible, the routine returns zero.

2.7.2 `dual_col` — choose non-basic variable (dual, standard technique)

Synopsis

```
#include "glprsm.h"
int dual_col(RSM *rsm, int tagp, double ap[], double cbar[], double tol);
```

Description

The `dual_col` routine chooses non-basic variable $(x_N)_q$ (i.e. pivot column of the simplex table) which should enter the basis on the next iteration of the dual simplex method. The routine is based on the standard (textbook) ratio test.

The parameter `tagp` is a tag that specifies to what subset x_L or x_U (see (18), Section 1.1) the chosen basic variable $(x_B)_p$ should be attributed after it has left the basis. This tag is set by the `dual_row` routine (see Subsection 2.7.1).

The array `ap` should contain p-th (pivot) row $\alpha_p = (\alpha_1, \ldots, \alpha_n)$ of the simplex table, i.e. p-th row of the matrix $\widetilde{A}$ (see (17), Section 1.1) in locations `ap[1]`, ..., `ap[n]`. This array is not changed on exit.

The array `cbar` should contain reduced costs $d = (d_1, \ldots, d_n)$ of non-basic variables in locations `cbar[1]`, ..., `cbar[n]`. This array is not changed on exit.

The parameter `tol` is a relative tolerance (see below).

The `dual_col` routine implements the standard (textbook) ratio test for choosing non-basic variable as described in Section 1.3 (see step 5), i.e. the routine determines that non-basoc variable whose dual variable reaches its (lower or upper) zero bound first when the dual variable $(\lambda_B)_p$ that corresponds to the chosen basic variable $(x_B)_p$ is changing in feasible direction (increasing if $(x_B)_p$ goes on its lower bound or decreasing if $(x_B)_p$ goes on its upper bound). Besides, the following additional rules are used:

if $|\alpha_{pj}| < tol \cdot \max_j |\alpha_{pj}|$, i.e. if the influence coefficient α_{pj} is relatively close to zero, it is assumed that the corresponding non-basic variable $(x_N)_j$ doesn't affect on the basic variable $(x_B)_p$ and therefore such non-basic variable is not considered to be chosen;

if the reduced cost d_j of some non-basic variable $(x_N)_j$ violates its (zero) bound, it is assumed that this happens due to round-off errors and actually the reduced cost is exactly zero (because the current basis solution should be dual feasible);

if several dual variables that correspond to non-basic primal variables reach their (zero) bounds first at the same time, the routine prefers that variable which has the largest (in absolute value) influence coefficient.

Returns

If the choice has been made, the `dual_col` routine returns q which is a number of the chosen non-basic variable $(x_N)_q, 1 \leq q \leq n$. Otherwise, if the dual variable $(\lambda_B)_p$ that corresponds to the chosen basic variable $(x_B)_p$ can unlimitedly change in the feasible direction and therefore the choice is impossible, the routine returns zero.

2.7.3 `harris_col` — choose non-basic variable (dual, Harris technique)

Synopsis

```
#include "glprsm.h"
int harris_col(RSM *rsm, int tagp, double ap[], double c[], double cbar[],
               double tol, double tol1);
```

Description

The `harris_col` routine chooses non-basic variable $(x_N)_q$ (i.e. pivot column of the simplex table) which should enter the basis on the next iteration of the dual simplex method. The routine is based on the two-pass ratio test proposed by P. Harris.

The `harris_col` routine has analogous program specification as the `dual_col` routine (see Subsection 2.7.2).

The first additional parameter is the array `c` which should contain the expanded vector c of coefficients of the objective function (4) in locations `c[1]`, ..., `c[m+n]`. It is assumed that the reduced costs are computed using the vector c passed to this routine.

The second additional parameter is `tol1` which is a relative tolerance used for relaxing zero bounds of dual variables λ_N (that correspond to primal non-basic variables x_N) on the first pass. The routine replaces original zero bounds of dual variables by their relaxed bounds:

$$(\lambda_N)_j \geq -\varepsilon \ or \ (\lambda_N)_j \leq +\varepsilon$$

where

$$\varepsilon = tol1 \cdot \max(1, |(c_N)_j|)$$

(the absolute tolerance ε reflects that the reduced cost of non-basic variable $(x_N)_j$ is the difference $d_j = (c_N)_j - N_j^T \pi$ as it follows from the formula (40) in Section 1.4).

For futher details see Section 1.7.

References

1. GLPK Library Reference.
2. *D. Goldfarb, J. K. Reid.* A practicable steepest-edge simplex algorithm. Mathematical Programming, 12, 1977, 361—71.
3. *P. M. J. Harris.* Pivot selection methods of the Devex LP code. Mathematical Programming, 5, 1973, 1—28.

Index

B

btran 22

C

change_b 28
check_bbar 25
check_cbar 26
check_dvec 33
check_gvec 30
check_rr 25
check_rsm 23

D

dual_col 37
dual_row 36

E

eval_bbar 24
eval_cbar 24
eval_col 26
eval_pi 24
eval_row 27
eval_xn 23
eval_zeta 27
exact_dvec 33
exact_gvec 30

F

ftran 21

H

harris_col 38
harris_row 35

I

init_dvec 31
init_gvec 28
invert_b 21

P

pivot_col 33
pivot_row 34

R

RSM 19

U

update_b 22
update_dvec 31
update_gvec 29