Linear Optimization

using

Simplex Method

Jiri Kriz

Version 4.0, November 2009

1 Introduction

In the Version 1 of this document we followed closely the description in [1], in particular also unbounded variables, a special cost function for the Phase I, etc. The implementation was based on swapping of variables. This approach appeared to be rather difficult to implement, error-prone and probably even inefficient, because not only the variables but also the respective columns of the matrix A must be swapped.

In this version we describe only the bounded optimization (i.e. all variables are bounded). In the Phase I, both slack and artificial variables are introduced if it is necessary. The implementation uses auxiliary arrays that specify whether a variable is basic or non-basic. The same approach is employed also in the QSO Optimizer.

2 Basic Concepts

We first explain bounded linear optimization (linear programming LP) for a simple "standard" form (see [1]):

$$\text{minimize} \quad g(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \tag{1}$$

$$\text{subject to} \quad \mathbf{Ax} = \mathbf{b} \tag{2}$$

$$\text{and} \quad \mathbf{0} \leq \mathbf{x} \leq \mathbf{u} \tag{3}$$

x is a n-vector of unknown variables, g is the cost function, **c** is an n-vector of cost coefficients, **A** is an m x n-matrix of constraint coefficients, **b** is a m-vector of right-hand sides. Because the vector **x** is bounded by the lower bound **0** and the upper bound **u**, the optimization is called "bounded optimization". Note that the more general bounds

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \tag{4}$$

are easily transformed to (3) by defining

$$\mathbf{x'} = \mathbf{x} - \mathbf{l} \tag{5}$$

This leads to the problem in the above form (1) - (3):

$$\text{minimize} \quad g(\mathbf{x'}) = \mathbf{c}^T\mathbf{x'} \tag{6}$$

$$\text{subject to} \quad \mathbf{Ax'} = \mathbf{b'} \tag{7}$$

$$\text{and} \quad \mathbf{0} \leq \mathbf{x'} \leq \mathbf{u'} \tag{8}$$

where:

$$\mathbf{b'} = \mathbf{b} - \mathbf{Al} \tag{9}$$

$$\mathbf{u'} = \mathbf{u} - \mathbf{l} \tag{10}$$

The matrix equation (2) represents m linear constraints

$$\mathbf{a}_i^T\mathbf{x} = b_i \tag{11}$$

We assume that:

$$n > m \tag{12}$$

$$\text{rank}\ (\mathbf{A}) = m\ \text{(maximal rank)} \tag{13}$$

We return to the assumptions later. Now, $\mathbf{A}$ can be partitioned into $\mathbf{A} = [\mathbf{A}_B \mid \mathbf{A}_N]$, with a nonsingular m x m "basic" matrix $\mathbf{A}_B$ and a (n-m) x m "nonbasic" matrix $\mathbf{A}_N$. In general, this can be done only if the columns of $\mathbf{A}$ have been swapped appropriately, corresponding to a new numbering of the variables x_i, giving $\mathbf{x}^T = [\mathbf{x}_B \mid \mathbf{x}_N]^T$. We denote by B the set of m basic variables and by N the set of (n-m) nonbasic variables. The equation (2) can be written as

$$\mathbf{A}_B\, \mathbf{x}_B + \mathbf{A}_N\, \mathbf{x}_N = \mathbf{b} \tag{14}$$

and used to compute the basic vector $\mathbf{x}_B$ from the nonbasic vector $\mathbf{x}_N$:

$$\mathbf{x}_B = \mathbf{A}_B^{-1}\, (\mathbf{b} - \mathbf{A}_N\, \mathbf{x}_N) \tag{15}$$

After substituting (15) into equation (1) we get the cost function in the "reduced" form":

$$g(\mathbf{x}) = \mathbf{c}^T\mathbf{x} = \mathbf{c}_B^T\, \mathbf{x}_B + \mathbf{c}_N^T\, \mathbf{x}_N = \mathbf{c}_B^T\mathbf{A}_B^{-1}\, \mathbf{b} + (\mathbf{c}_N^T - \mathbf{c}_B^T\, \mathbf{A}_B^{-1}\, \mathbf{A}_N)\, \mathbf{x}_N \tag{16}$$

With "reduced" costs defined as:

$$\hat{\mathbf{c}}_N = \mathbf{c}_N - \mathbf{A}_N^T\, \boldsymbol{\pi} \tag{17}$$

$$\boldsymbol{\pi} = \mathbf{A}_B^{-T}\, \mathbf{c}_B \tag{18}$$

the cost function g() becomes a function of the nonbasic variables $\mathbf{x}_N$ only:

$$g(\mathbf{x}) = g(\mathbf{x}_N) = \text{constant} + \hat{\mathbf{c}}_N^T\, \mathbf{x}_N \tag{19}$$

Now, suppose that all the nonbasic variables $\mathbf{x}_N$ are at their possible extreme values $\mathbf{0}$ or $\mathbf{u}$, i.e. each nonbasic variable x_i is either 0 (lower bound) or u_i (upper bound). If for all nonbasic x_i:

$$\hat{c}_i \geq 0 \text{ if } x_i = 0 \tag{20}$$

$$\hat{c}_i \leq 0 \text{ if } x_i = u_i \tag{21}$$

then we have an optimal solution. If (20) applies, then x_i can only increase and since $\hat{c}_i \geq 0$, $g(\mathbf{x})$ would increase. If (21) applies, then x_i can only decrease and since $\hat{c}_i \leq 0$, $g(\mathbf{x})$ would also increase. The equations (20) and (21) represent hence the “optimality test”.

On the other hand, it is also true, that if $\mathbf{x}$ is an optimal solution, then $\mathbf{x}$ can be partitioned into m basic and (n-m) nonbasic variables, such that all nonbasic variables are at their extreme values (0 or u) and (20) - (21) hold true. The informative reason is that a linear function g() achieves its minimum always on the boundary of the feasible region(2), (3). The formal proof is more difficult [1]. The LP problem can hence be reduced to the search of the partitioning B, N such that (20) - (21) hold true. The simplex method makes this search systematic as explained below.

3 The Revised Simplex Method

3.1 Phase II

We assume that we have a basic feasible "solution" $\mathbf{x}$ (b.f.s.), such that $\mathbf{x}^T = [\mathbf{x}_B \mid \mathbf{x}_N]^T$, $\mathbf{x}_N$ are at the exreme boundary values (0 or u), $\mathbf{x}_B$ satisfies (3), and (15) is fulfilled. We also assume that we have the mxm matrix $\mathbf{A}_B^{-1}$. Then one iteration step of the Phase II of the revised simplex algorithm is as follows.

We compute $\boldsymbol{\pi}$ using (18), $\hat{\mathbf{c}}_N$ using (18), and make the optimality test (20) - (21). If the test is fulfilled, $\mathbf{x}$ is the optimal solution. If the test is not fulfilled, there is at least one coefficient $\hat{c}_q$ which does not satisfy the optimality test (20) - (21). We usually chose q to be that index such that $\hat{c}_q$ is the coefficient with the greatest absolute value, because along such axis q the goal function g() changes maximally:

$$\hat{c}_q = \max \{\text{abs}(\hat{c}_j) \mid j \in N,\ \hat{c}_j < 0 \text{ and } x_j = 0\} \cup \{\hat{c}_j \mid j \in N,\ \hat{c}_j > 0 \text{ and } x_j = u_j\} \tag{22}$$

Now, the nonbasic x_q will be changed as much as possible to decrease g(). The nonbasic x_q changes to

$$x_q' = x_q + \Delta_q, \tag{23}$$

$$0 < \Delta_q < u_q \quad \text{if } x_q = 0 \quad x_q \text{ will increase to } x_q' > x_q \tag{24}$$

$$-u_q < \Delta_q < 0 \quad \text{if } x_q = u_q \quad x_q \text{ will decrease to } x_q' < x_q \tag{25}$$

All other nonbasic x_j remain unchanged. The basic vector $\mathbf{x}_B$ changes according to (15) to

$$\mathbf{x}_B' = \mathbf{x}_B + \mathbf{d}\,\Delta_q \qquad \mathbf{d} \text{ is an m-vector:} \tag{26}$$

$$\mathbf{d} = -\,\mathbf{A}_B^{-1}\,\mathbf{a}_q, \qquad \mathbf{a}_q = \text{q-th column of A} \tag{27}$$

However, the change in $\mathbf{x}_B$ is restricted by the condition that $\mathbf{x}_B'$ stays within the bounds (3). For all $j \in B$ we perform the test (3) and if it is violated, we reduce the absolute value of Δ_q :

$$\text{If } x_j' = x_j + d_j\,\Delta_q > u_j, \text{ set } \Delta_q = (u_j - x_j) / d_j \tag{28}$$

$$\text{If } x_j' = x_j + d_j\,\Delta_q < 0, \text{ set } \Delta_q = (0 - x_j) / d_j \tag{29}$$

If $d_j = 0$ in the above test, we skip it. Let p be the index in B, where the tightest restriction have been performed, i.e. that index in a loop over B, where the last resetting of Δ_q occurred. Set p = -1, if no such index exists in B.

If p = -1, the basic variables do not restrict Δ_q. In this case the nonbasic variable x_q moves to its other bound, the other nonbasic variables do not change, and the basic variables are computed according to (26) - (27).

If $p \neq -1$, then x_p' achieved its lower or upper bound while x_q' is no longer at a bound but strictly between its bounds. This means that the basic variable x_p' becomes nonbasic, and the nonbasic variables x_q' becomes basic. We must hence swap the variables x_p and x_q.

Swapping of x_p and x_q changes the matrix $\mathbf{A}_B^{-1}$ according to (see [1], and Appendix 4):

$$(\mathbf{A}_B^{-1})' = \mathbf{A}_B^{-1} - (\mathbf{d} + \mathbf{e}_p)\, \mathbf{v}_p^T / d_p \tag{30}$$

where $\mathbf{v}_p^T$ is the p-th row of $\mathbf{A}_B^{-1}$ and $\mathbf{e}_p$ is the unit vector along the p-axis. Now, the iteration step is completed and a new one can start.

Remarks

- In (28) or (29), the found x_p' can exactly equals the bound (0 or u_p). In this case we can either move x_q to its other bound, or swap the variables x_p and x_q.A good choice is perhaps to swap the variables (see below). Another alternative would be to chose randomly (say fifty-fifty) between swapping and moving.

- In (28) or (29), we can get $\Delta_q = 0$ if the basic variable x_p is at one of its bounds. This means that the nonbasic x_q will not change. But since x_p is at its bound we still can swap $p \leftrightarrow q$, although the value of the objective function will not change. This case is called degeneracy. We hope that after the swap we get a new constellation and can decrease g() in the future. However it could happen that the iteration cycles without any progress. [1] describes ways how to break this cycling. They are rather complicated and are not considered in our implementation. The implementation could check whether the goal function does not decrease during several iterations and if so, abort. Unfortunately an often calculation of the goal function takes some time.

- The calculation of $(\mathbf{A}_B^{-1})'$ can be done either using the formula (30) or after making some formal manipulations on (30) as follows. Denote β_{ij} the (i, j)-element of $\mathbf{A}_B^{-1}$. Then:

$$\beta'_{pj} = -\beta_{pj} / d_p \qquad \text{for } i = p \tag{31}$$

$$\beta'_{ij} = \beta_{ij} + d_i\, \beta'_{pj} \qquad \text{for } i \neq p$$

3.2 Artificial Variables (Phase I)

One problem still remains, namely how to obtain a first feasible solution and the corresponding matrix $\mathbf{A}_B^{-1}$. This is done in the "Phase I" of the optimization by introducing the so called artificial variables. However, before doing so, each equation j where $b_j < 0$ is multiplied by -1 to get $\mathbf{b} \geq \mathbf{0}$. The reason will become evident below (see eq. (36)). Then we extend the original problem to a new problem:

$$\text{minimize} \qquad G(\mathbf{r}, \mathbf{x}) = \mathbf{0}^T\mathbf{x} + \mathbf{1}^T\mathbf{r} = \textstyle\sum_{i = 1,..m} r_i \tag{32}$$

$$\text{subject to} \qquad \mathbf{Ax} + \mathbf{Ir} = \mathbf{b} \tag{33}$$

and $\mathbf{0} \le \mathbf{x} \le \mathbf{u},\ \mathbf{0} \le \mathbf{r} \le \mathbf{R}$ (34)

where $\mathbf{1}^T$ is an m-vector of ones, $\mathbf{r}$ is an m-vector of " artificial variables, $\mathbf{R}$ is an m-vector of big positive numbers and $\mathbf{I}$ is an mxm identity matrix. We set the m basic variables to $\mathbf{r}$ and the n nonbasic variables to $\mathbf{x} = \mathbf{0}$. Now

$$\mathbf{A}_B = \mathbf{A}_B^{-1} = \mathbf{I} \tag{35}$$

Since we have achieved $\mathbf{b} \ge \mathbf{0}$, we have a feasible $\mathbf{r}$:

$$\mathbf{r} = \mathbf{b} - \mathbf{Ax} = \mathbf{b} \ge \mathbf{0} \tag{36}$$

and hence an overall feasible vector $(\mathbf{x}, \mathbf{r}) = (\mathbf{0}, \mathbf{b})$ for the extended problem. We solve the extended problem with n + m variables using the Phase II. Notice that the extended problem has in fact the full rank m due to the identity matrix $\mathbf{I}$. The solution of the extended problem can have the following outcomes:

1. $G(\mathbf{x}, \mathbf{r}) = 0$, and all artificial variables became nonbasic. Consequently, the artificial variables are surely at their respective bounds. But because G = 0, all artificial variables are even 0, because otherwise G would not be 0 (see (32)). So, $\mathbf{Ax} = \mathbf{b}$, and $\mathbf{0} \le \mathbf{x} \le \mathbf{u}$ anyhow. Hence, the current $\mathbf{x}$ is a feasible solution of the original problem, it is partitioned into basic and nonbasic variables and the current $\mathbf{A}_B^{-1}$ is the corresponding inverse matrix. Now, we can remove the artificial variables from the problem and should also do so.

2. $G(\mathbf{x}, \mathbf{r}) = 0$, and some artificial variables are still basic. This indicates that the original matrix $\mathbf{A}$ (2) has not the full rank m. Now, we must leave the basic artificial variables in the solution vector. We could remove the nonbasic artificial variables and renumber the remaining artificial variables in a consecutive sequence. Instead of a possibly tedious renumbering, we can keep all artificial variables and just reset the goal function G to g, i.e. set the cost coefficient of all artificial variables to 0, because now the cost function should not depend on artificial variables. Moreover, we can set the column in $\mathbf{A}$ corresponding to a nonbasic articial variable to zero. Then, such a variable will never be selected as a q-variable (for swap or moving), because its reduced cost coefficient is always 0.

3. $G(\mathbf{r}, \mathbf{x}) \ne 0$ and hence positive. This implies $\mathbf{r} \ne \mathbf{0}$ and consequently (2) is inconsistent. The original problem is infeasible. If (2) had a solution, then $\mathbf{r} = \mathbf{0}$ would be a solution of (33) with G = 0 which is a contradiction.

3.3 Slack Variables (Phase I)

The linear constraints on the variables are very often inequalities, e.g.:

$$\mathbf{a}_i^T\mathbf{x} \le b_i \tag{37}$$

They are transformed to equalities by adding so called slack variables:

$$\mathbf{a}_i^T\mathbf{x} + s_i = b_i \tag{38}$$

$$0 \le s_i \le S_i$$

The upper bound S_i is set to a “big number”. It can also be estimated more precisely as follows:

$$s_i \le b_i - \sum_{j \in J} a_{ij} u_j = S_i \qquad J = \{j \mid a_{ij} < 0\} \tag{39}$$

The slack variables are treated as the original variables in the optimization; in fact they are among the original variables. The cost coefficient of the slack variables is set to 0, because the cost function does not depend on it. In addition, the fact should be utilized that the column of **A** corresponding to a slack variable s_i contains zeros except one 1 in the row i, and should be used for the construction of the first $\mathbf{A}_B^{-1}$. If an equation i has a slack variable we could use it instead of an artificial variable in Phase I to reduce the number of variables. However, this is possible only if $b_i >= 0$. If $b_i < 0$, we must multiply the equation I by (-1) and introduce an artificial variable r_i:

$$-\mathbf{a}_i^T\mathbf{x} - s_i + r_i = -b_i \tag{40}$$

Note that the slack variables remain in the solution vector after the Phase I.

4 Implementation Issues

4.1 Basic/Nonbasic Sets

In the first version of this paper we favored the representation of the basic variables by the first m position in the solution vector. This means that permutation arrays for the variables must be maintained and if two variables are swapped also the corresponding columns of **A** must be swapped. Now, we think that it is better to maintain indicator vectors that designate the basic and nonbasic variables. The same approach was already taken in the original QSO optimizer.

We introduce the m-array basic(i), i = 1, …, m, such that basic(i) is the index of the i-th basic variable in the vector **x**. We also introduce an n-array indicator(i) where each element has one of three possible values:

indicator(i) =	BASIC	the variable x_i is basic
	NONBASIC_LOWER	the variable x_i is nonbasic and is at its lower bound
	NONBASIC_UPPER	the variable x_i is nonbasic and is at its upper bound

Setting a variable x_k as the i-th basic variable would be accomplished by:

```
basic(i) = k
indicator(k) = BASIC
```

Swapping the i-th basic variable that achieved e.g. its upper bound, such that basic(i) = p, with a nonbasic variable x_q would be accomplished by:

```
basic(i) = q
indicator(q) = BASIC
indicator(p) = NONBASIC_UPPER
```

4.2 Variable bounds

At the beginning of the Phase I we check first the variable bounds l_i, u_i:

1) $l_i > u_i$
   This is an impossible constraint. We abort.

2) $l_i = u_i$
   The variable x_i must equal l_i = ui. We could eliminate it, but then we would need to renumber other variables. It seems easier to keep x_i but ensure that it will not be considered in the optimization. We set the cost coefficient c_i as well as the whole i-column of **A** to zero.

3) $l_i < u_i$
   Nothing must be done.

Now, we transform the variables **x’ = x – l** to achieve the lower bound equal to zero, see (6) - (10).

4.3 Constraints

We allow m constraints of the types:

$$\mathbf{a}_i^T\mathbf{x} = b_i \tag{EQ}$$

$$\mathbf{a}_i^T\mathbf{x} \le b_i \tag{LE}$$

$$\mathbf{a}_i^T\mathbf{x} \ge b_i \tag{GE}$$

We transform the type GE to LE by multiplying it by (-1), such that all constraints are of the type EQ or LE.

4.4 Variables and Memory Allocation

Let N be the number of original variables of the problem. We can compute the exact number of slack and artificial variables as follows. Let m_{EQ} be the number of equalities, m_{LE+} the number of LE inequalities where bi ≥ 0 and m_{LE-} the number of LE inequalities where bi < 0. Then, we will have $n_S = (m_{LE+} + m_{LE-})$ slack variables and $n_A = (m_{EQ} + m_{LE-})$ artificial variables. The exact number of variables for the Phase I is hence

$$n = N + (m_{LE+} + m_{LE-}) + (m_{EQ} + m_{LE-}) = N + m + m_{LE-} \le N + 2m \tag{41}$$

The total number n of variables is important for allocating storage for the matrix **A**, the vector **x**, and other entities. Since we should not expect that the user will provide the value m_{LE-} in advance, we we shall adopt the upper bound (N + 2m) for the allocation of values with the dimension n (number of variables).

4.5 Slack and Artificial Variables

During the Phase I we compute the exact values of the slack variables n_S and the artificial variables n_A:

$$n_S = m_{LE+} + m_{LE-} \tag{42}$$

$$n_A = m_{EQ} + m_{LE-}$$

Let N be the number of original variables. We iterate over the restrictions (C-indexing: i = 0, 1, .. m-1) and for the restriction i do the following:

1) If $\mathbf{a}_i^T \mathbf{x} \le b_i$ we introduce a slack variable at the position $j = N + N_s$ in the vector **x**, where N_s is the current number of slack variables.
   If b_i>= 0 we make the new slack variable x_j basic.
   If b_i< 0 we multiply the restriction by (-1) (including the coefficient 1 at the slack variable) and introduce also an artificial variable at the position $k = N + n_S + N_A$ in the vector **x**, where N_A is the current number of artificial variables. We make this new variable x_k basic.

2) If $\mathbf{a}_i^T \mathbf{x} = b_i$ and b_i< 0 we multiply the restriction by (-1) and introduce an artificial variable at the position $j = N + n_S + N_A$ in the vector **x**, where N_A is the current number of artificial variables. We make the new variable x_j basic.
   If $\mathbf{a}_i^T \mathbf{x} = b_i$ and b_i>= 0 we do nothing.

When a new variable x_j is introduced while we are at the restriction i, the i-th column of **A** is filled with 0 (this can be done for the whole matrix **A** at the beginning) except at the i-th row where 1 is placed.

4.6 Number of Iterations

The implementation must specify the maximal number of iterations in order to avoid an infinite loop. [2] (p. 26) states that the number of iterations is usually of order 2-4 times m, the number of constraints. However, it is known that the simplex algorithm is exponential (in n) in the worst case, although the LP can be solved in polynomial time. [1] (p. 184) presents an LP problem, where the feasible region has 2^n vertices, which all are visited by the simplex algorithm. Our experiences with the QSO optimization showed that the number of iterations can be much higher than 4m; we set it to 2n + 20m.

4.7 Storage Requirement

The storage requirement depends on n (total number of variables) and m (number of constraints) and the used numerical type for the computation (float or double). The storage need is clearly dominated by the matrix **A** with its m (N + 2m) elements. The experience shows that the computation must be performed in doubles, otherwise the rounding errors destroy the solution. However, the entities that do not change during the computation can be stored in floats; these are: the matrix **A**, the right hand **b**, the bounds **l**, **u**, and the cost **c**. We assume that a double needs 8, and the float 4 bytes.

Item		Space	Bytes/item
constraints	**A**	m * (N + 2m)	4
inverse basic	$\mathbf{A}_B^{-1}$	m * m	8
right hand side	**b**	m	4
solution	**x**	N + 2m	8
lower bound	**l**	N+ 2m	4
upper bound	**u**	N + 2m	4
cost	**c**	N + 2m	4
reduced cost	$\hat{\mathbf{c}}$	N +2m	8
derivative	**d**	m	8
constraint type	**type**	m	1

auxiliary	**aux**	N + 2m	8
basic	basic	m	1
indicator	indicator	N + 2m	1
Total	m(N + 3m) + 7(N + 2m) + 4m		63

Example: m = 30, N = 10'000: 373’240 * 63 = 23’514’120 bytes ~ 24 MB (quite a lot!).

During the calculation the floats are converted to doubles which could slow down the execution time a bit.

In some problems the matrix A is sparse (contains many zero), e.g. it is reported that in some network flow problems there more than 90% zeros in A. For these problems special sparse matrix methods using matrix decomposition are suggested in the literature [1] and offered in some commercial products (e.g. FrontLine Systems). While QuarryMaster probably belongs to this problem category, QSO surely does not. In contrary, the matrix A contains only about 5% zeros in a large QSO problem.

4.8 Computing Time

The most computing time is spent in the iteration process, so we investigate the time of one iteration step:

Operation		Time
1)	$\boldsymbol{\pi} = \mathbf{A}_B^{-T}\,\mathbf{c}_B$	m * m
2)	$\hat{\mathbf{c}}_N = \mathbf{c}_N - \mathbf{A}_N^T\,\boldsymbol{\pi}$	n * m
3)	find nonbasic q	n
4)	find basic p	m
5)	swap $p \leftrightarrow q$	0
6)	update basic $\mathbf{x}_B$	m
7)	update $\mathbf{A}_B^{-1}$	m * m
	Total	n * m + 2 *m * m + n + 2m

Interestingly, in the "classical" situation n >> m, the calculation of the reduced costs (2), i.e. $\mathbf{A}_N^T\,\boldsymbol{\pi}$, is by far the most time consuming operation and should be optimized. Besides the brute force approach like assembler programming of scalar products, the following heuristics appears useful: the vector $\boldsymbol{\pi}$ contains usually some zero components. If $\pi_i = 0$, then the whole i-th column of $\mathbf{A}_N^T$, i.e. the whole i-th row of $\mathbf{A}_N$ can be skipped during the multiplication.

Why there is a good chance that the vector $\boldsymbol{\pi}$ contains zeros? The iteration starts with $\mathbf{c}_B$ containing zeros for all slack variables in their respective bounds and with $\mathbf{A}_B^{-T} = \mathbf{I}$. So, the first $\boldsymbol{\pi}$ will contain many zeros. During the iteration, $\mathbf{A}_B^{-T}$ becomes filled and the slack variables will be replaced by the original variables in the basis, so the zeros in $\boldsymbol{\pi}$ could continuously disappear. In the case of QSO (optimizing tonnage) the experience shows that there many zeros remain in $\boldsymbol{\pi}$ during the whole iteration such that it is worthwhile to perform the test $\pi_i = 0$.

In general, a matrix product $\mathbf{Ax} = \mathbf{b}$ can be written as $\mathbf{b} = \sum x_j\mathbf{a}_j$ where $\mathbf{a}_j$ is the j-th column of $\mathbf{A}$. So if some x_j are 0, the product can be programmed by accumulating the partial sums in the column $\mathbf{b}$:

```
b = 0;
for (j = 0; j < m; ++j)
        // add the j-th column:
```

```
if ($x_j \neq 0$) {
        for (i = 0; i < n; ++i)
                $b_i$ += $x_j a_{ij}$;
}
```

This technique is applied for the calculation of $\boldsymbol{\pi}$ but could also be applied for the calculation of $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$ (when needed) because the many nonbasic variables are at their lower (transformed) bound 0.

4.9 Rounding Errors

We face the rounding errors by various measures. We replace the tests $d > 0$, $d < 0$, by $d > \varepsilon$, $d < -\varepsilon$. We keep always all variables inside their bounds. If a new value $x_q' = x_q + \Delta_q$ is outside of $[0, u_q]$ we set it to the respective bound 0 or u_q. Moreover, the nonbasic variables are exactly equal to one of their bounds.

We could monitor whether the constraints are fulfilled by computing $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$ which should be 0 [2] (p. 34) and if some $\text{abs}(r_i) > 10^{-6}$, excessive rounding error is present. Computing of $\mathbf{r}$, however, is very time consuming and should be done rarely because it needs about n*m multiplications. The number of multiplications can be reduced by utilizing the situation that a variable can be at its lower bound 0; this applies in particular to nonbasic variables.

If an element of $\mathbf{A}_B^{-1}$ is very small, it could be replaced by 0 [2] (p. 34).

The entities that are most critical to rounding errors are the inverse basic matrix $\mathbf{A}_B^{-1}$ and the solution vector $\mathbf{x}$, that are both accumulated during the iteration. In fact only $\mathbf{A}_B^{-1}$ is critical, because $\mathbf{x}$ can be recomputed using (15) (if we know $\mathbf{x}_N$).
The idea of the **reinversion** is to recompute $\mathbf{A}_B^{-1}$ if it gets inaccurate after many iterations. The matrix is recomputed as follows. We start with $\mathbf{A}_B^{-1} = I$ and the initial basic set $B_0 = \{j_1, j_2, .. j_m\}$ with slack and artificial variables (we must keep this set from the Phase I). Notice that all indexes in B_0 are greater than N = number of original variables. Let the current set of basic variables be $B_{curr} = \{i_1, i_2, .. i_m\}$. Suppose that $i_1 = p$ is not in B_0. We swap p and $j_1 = q$ to make p basic. The matrix $\mathbf{A}_B^{-1}$ changes to

$$(\mathbf{A}_B^{-1})' = \mathbf{E}(p, q)\, \mathbf{A}_B^{-1} \tag{43}$$

where $\mathbf{E}(p, q)$ is an identity matrix in which the p-th row is replaced by the vector $\boldsymbol{\eta}$ where

$$\eta_i = -\, d_i / d_p,\ i \neq p; \qquad \eta_p = -\,1 / d_p \tag{44}$$

and $\mathbf{d} = -\,\mathbf{A}_B^{-1}\,\mathbf{a}_q$. This can be proved as follows:

$\mathbf{A}_B = [\mathbf{A}_B e_1, .., \mathbf{A}_B e_{p-1}, \mathbf{A}_B e_p, \mathbf{A}_B e_{p+1}, .., \mathbf{A}_B e_m]$, where $\mathbf{A}_B e_j$ is the j-th column of $\mathbf{A}_B$

$\mathbf{A}_B' = [\mathbf{A}_B e_1, .., \mathbf{A}_B e_{p-1}, \mathbf{a}_q, \mathbf{A}_B e_{p+1}, .., \mathbf{A}_B e_m]$, because the p-th column was replaced by $\mathbf{a}_q$

$\mathbf{A}_B' = \mathbf{A}_B\, [e_1, .., e_{p-1}, -\mathbf{d}, e_{p+1}, .., e_m]$, because $\mathbf{a}_q = -\mathbf{A}_B \mathbf{d}$

$(\mathbf{A}_B')^{-1} = [e_1, .., e_{p-1}, -\mathbf{d}, e_{p+1}, .., e_m]^{-1} \mathbf{A}_B = \mathbf{E}(p, q)\, \mathbf{A}_B^{-1}$

The transformation (43) is equivalent to (30) resp. (31). It requires m^2 operations and is done for all m elements in B_{curr}. Hence, the cost of reinversion is m^3 as expected (inversion of a mxm-matrix takes m^3 operations).

We would proceed in similar way with the variables $i_2, .. i_m$ changing them against $j_2, .. j_m$. However, if i_1 is already in B_0 we should not swap it. A straightforward way is to preprocess the sets B_{curr} and B_0 and mark indexes in both sets that should not be swapped. Then we iterate over B_{curr} and swap the next swappable p in B_{curr} with the next swappable p in B_0 .

After the matrix $\mathbf{A}_B^{-1}$ is recomputed, we compute the basic variables $\mathbf{x}_B$ using (15) and the current nonbasic variables $\mathbf{x}_N$.

A1 The Sherman-Morrison Matrix Identity

If

$$\mathbf{A}' = \mathbf{A} + \mathbf{u}\mathbf{w}^T, \qquad \mathbf{A} : \text{n x n-matrix}, \mathbf{u}, \mathbf{w} : \text{n-vectors} \tag{45}$$

then

$$(\mathbf{A}')^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{u}\mathbf{w}^T\mathbf{A}^{-1} / (1 + \mathbf{w}^T\mathbf{A}^{-1}\mathbf{u}) \tag{46}$$

Note that the expression $(1 + \mathbf{w}^\mathbf{T}\mathbf{A}^{-1}\mathbf{u}) = s$ is a scalar. The proof is straightforward by multiplying the both equations:

$$(45) * (46) = \mathbf{I} + (-1/s + 1)\mathbf{u}\mathbf{w}^\mathbf{T}\mathbf{A}^{-1} - \mathbf{u}(\mathbf{w}^\mathbf{T}\mathbf{A}^{-1}\mathbf{u})\mathbf{w}^\mathbf{T}\mathbf{A}^{-1} = \mathbf{I} + [-1+s - (s-1)]\mathbf{u}\mathbf{w}^\mathbf{T}\mathbf{A}^{-1} = \mathbf{I}$$

In the revised simplex method the basic matrix $\mathbf{A}_B$ is updated by replacing its column $\mathbf{a}_p$ by $\mathbf{a}_q$, i.e.:

$$\mathbf{A}' = \mathbf{A} + (\mathbf{a}_q - \mathbf{a}_p)\mathbf{e}_p^T \tag{47}$$

Using (46) we get

$$\mathbf{A}^{-1}\mathbf{u}\mathbf{w}^T\mathbf{A}^{-1} = \mathbf{A}^{-1}(\mathbf{a}_q - \mathbf{a}_p)\mathbf{e}_p^T\mathbf{A}^{-1} = (\mathbf{A}^{-1}\mathbf{a}_q - \mathbf{A}^{-1}\mathbf{a}_p)\mathbf{v}_p^T = (-\mathbf{d} - \mathbf{A}^{-1}\mathbf{A}\mathbf{e}_p)\mathbf{v}_p^T = -(\mathbf{d} + \mathbf{e}_p)\mathbf{v}_p^T$$

$$1 + \mathbf{w}^T\mathbf{A}^{-1}\mathbf{u} = 1 + \mathbf{e}_p^T\mathbf{A}^{-1}(\mathbf{a}_q - \mathbf{a}_p) = 1 + \mathbf{e}_p^T(-\mathbf{d} - \mathbf{A}^{-1}\mathbf{a}_p) = 1 - \mathbf{d}_p - \mathbf{e}_p^T\mathbf{A}^{-1}\mathbf{A}\mathbf{e}_p = -\mathbf{d}_p$$

and (30) is proved. We have heavily used the identity $\mathbf{a}_p = \mathbf{A}\mathbf{e}_p$, because $\mathbf{a}_p$ is the current p-th column of $\mathbf{A}$.

References

[1] Fletcher, R.: Practical Methods of Optimization, 2nd ed., J. Wiley, 1987 (reprinted 1997) {excellent}

[2] Murtagh, B.A.: Advanced Linear Programming: Computation and Practice, McGraw-Hill, 1981