Applying Schur Complements for Handling General Updates of a Large, Sparse, Unsymmetric Matrix*

Jacek Gondzio
Systems Research Institute, Polish Academy of Sciences,
Newelska 6, 01-447 Warsaw, Poland
e-mail: gondzio@ibspan.waw.pl

Technical Report ZTSW-2-G244/93
September 1993, revised April 30, 1995

Abstract

We describe a set of procedures for computing and updating an inverse representation of a large and sparse unsymmetric matrix A. The representation is built of two matrices: an easily invertible, large and sparse matrix A_0 and a dense Schur complement matrix S. An efficient heuristic is given that finds this representation for any matrix A and keeps the size of S as small as possible.

Equations with A are replaced with a sequence of equations that involve matrices A_0 and S. The former take full advantage of the sparsity of A_0; the latter benefit from applying dense linear algebra techniques to a dense representation of the inverse of S.

We show how to manage five general updates of A: row or column replacement, row and column addition or deletion and a rank one correction. They all maintain an easily invertible form of A_0 and reduce to some low rank update of matrix S.

An experimental implementation of the approach is described and the preliminary computational results of applying it to handling working basis updates that arise in linear programming are given.

Key words: large sparse unsymmetric matrix, inverse representation, Schur complement, general updates.

*Supported by the Committe for Scientific Research of Poland, grant No PB 8 S505 015 05.

1 Introduction

In this paper we describe a method for the inverse representation of a large and sparse unsymmetric matrix A. The representation is expected to be used to solve equations

$$Ax = b \qquad \text{and} \qquad A^T y = d, \tag{1}$$

where x, b, y and d are vectors from $\mathcal{R}^n$.

We present a method for updating this representation after the following modifications of A:
– row replacement,
– column replacement,
– row and column addition,
– row and column deletion,
– rank one change.

LU decomposition (cf. Duff et al. [5], Chapter 8) is probably the most widely used method for representing an inverse of a sparse unsymmetric matrix A. Simplicity and efficiency of its update after column replacement is one of its main advantages. The updates by the method of Bartels and Golub [1] preserve sparsity of LU factors and ensure accuracy of the representation that is sufficient for most practical applications. There exist many efficient implementations of this method [4, 29, 32], including an extension of Gill et al. [13] for different updates of a general (rectangular) A.

The Bartels-Golub updates of sparse LU factors have one basic disadvantage: they are achieved by a purely sequential algorithm. Hence, relatively little can be done to specialize them for new computer architectures.

We feel motivated to look for a *parallelisable* method for the inverse representation of a sparse unsymmetric matrix A. Additionally, in order to cover a wider field of applications we would like to be able to update this representation inexpensively and in a stable way after simple modifications of A.

The method presented in this paper satisfies these requirements.

We propose to represent the inverse of A with two matrices: an easily invertible *fundamental basis* A_0 and a Schur complement S (hopefuly, of small size). The fundamental basis is an easily invertible matrix close to A (subject to the row and column permutations). The "closeness" is measured with the number of columns of A that have to be replaced with appropriate unit vectors in order to get a nonsingular matrix permutable to A_0. The number of columns that do not fit the required structure determines the size of the Schur complement. It is in our interest to keep this number as small as possible.

Many different forms of A_0 might be chosen. The most suitable one would clearly depend on the particular structure of A. The first suggestion is to let A_0 be triangular. Although it seems very restrictive, this simple form already covers a large class of matrices, as e.g., linear programming bases arising in network or stochastic optimization [20]. (The structure inherent to these problems implies that all bases are "nearly triangular".)

As shown in [17], the general linear programming bases can always be permuted to a form close to a block triangular matrix with at most 2x2 irreducible blocks. This particular form of A_0 ensures that equations with it can be solved very efficiently since the sparsity of A_0 as well as that of the right hand side vectors can be fully exploited.

It was also shown in the abovementioned paper that the Schur complement inverse representation of A^{-1} can be updated in a stable and efficient way after column replacement in A (a usual modification of the linear programming bases).

In this paper we extend the set of available updates of this representation with four new modifications to A: row replacement, row and column addition or deletion and a rank one update. We show that in all cases the update resolves to one or at most two rank one corrections to the Schur complement. Those, in turn, can be handled in a stable way when dense LU factorization [16] is applied to represent S^{-1}.

The motivation for the development of the method presented in this paper was a need for a reliable inverse representation of the working basis in a new approach to the solution of large linear optimization problems [19]. Hence, our illustrative test examples will come from this particular application. However, we want to stress that the method presented can be used in a much wider context whenever a sequence of equations with large and sparse unsymmetric matrix A is to be solved and some rows/columns of it are supposed to be replaced, deleted or bordered to A.

Let us mention that Schur complements are widely used in general engineering computations [3, 22] as well as in the optimization applications [12]. Their use for handling the inverse representation of linear programming bases has been addressed by many authors [2, 6, 9] independently of the author's own interest [17, 20].

Some features of the method presented in this paper make it a particularly attractive alternative to the approach that uses Bartels-Golub updates of the sparse LU factorization [13]. The most important one is splitting the factorization step into "sparse" and "dense" parts. Fundamental basis A_0 shares the structure of A, hence it is supposed to be large and sparse. All equations with A_0 will take full advantage of this fact. In contrast, the Schur complement matrix S is expected to be small and filled sufficiently to justify the use of dense factorization to represent its inverse. The use of explicit LU factors of S significantly simplifies the control of accuracy: sparsity does not influence the choice of the most stable elementary operator. Another important feature of the approach presented in this paper is that its factorization step is structurally parallelisable. Finally, we underline a simplicity and low cost of handling a rank one update of A.

The paper is organized as follows. In Section 2, the factorization step is discussed in detail. In particular, we address the heuristics for permuting a given matrix to a *bordered triangular form*, their extensions to find a *bordered block triangular form* with 2x2 pivots on the diagonal, and the issue of accuracy control in the A^{-1} representation. In Section 3, the techniques of updating the inverse of A after matrix modifications mentioned earlier are described. Additionally, we

address the issues of complexity and efficiency of the updates. In Section 4, we discuss the well known conflict that appears in sparse matrix computations between preserving a sparsity of A^{-1} representation and ensuring its accuracy. In Section 5, we give some preliminary computational results obtained with an experimental implementation of the method proposed. Final remarks and conclusions are the subject of Section 6.

2 Factorization

2.1 Fundamentals

Let A_0 be an $n \times n$ matrix, call it a fundamental basis, that differs from A with k columns. These different columns are determined by the partitioning matrix $V \in \mathcal{R}^{k \times n}$ that is built of k unit vectors, each matching one column removed from A. Subject to column permutation, A can be expressed in the form [2]

$$A = [A_1|C]\,, \tag{2}$$

where $A_1 \in \mathcal{R}^{n \times (n-k)}$ contains columns that belong to A and A_0 while C contains columns of A that are not in A_0. Hence

$$AV^T = C \qquad \text{and} \qquad A = A_0 + (C - A_0V^T)V. \tag{3}$$

Let us introduce an augmented matrix

$$A_{aug} = \begin{bmatrix} A_0 & C \\ V & 0 \end{bmatrix}. \tag{4}$$

The pivoting operation done on the whole block A_0 reduces A_{aug} to

$$\begin{bmatrix} I_n & A_0^{-1}C \\ 0 & -S \end{bmatrix} \tag{5}$$

where S denotes the Schur complement [3, 22]

$$S = VA_0^{-1}C. \tag{6}$$

Lemma 1. *Let the representation (2)-(3) be given. If $A, A_0 \in \mathcal{R}^{n \times n}$ are nonsingular, then $rank(A_{aug}) = n + k$ and $rank(S) = k$.*

Proof. *From the definition, rows of V are linearly independent. Let us permute the rows and columns of A_{aug} so that the nonzeros of V are placed in the upper left corner of it. Its first k columns form now a unit matrix in $\mathcal{R}^{k \times k}$. As $A_0V^T = V^T$, the permuted matrix has the form*

$$\begin{bmatrix} I_k & 0 & 0 \\ V^T & A_1 & C \end{bmatrix} = \begin{bmatrix} I_k & 0 \\ V^T & A \end{bmatrix}.$$

The nonsingularity of $A \in \mathcal{R}^{n\times n}$ thus yields the nonsingularity of A_{aug} ($rank(A_{aug}) = n + k$). Let us observe that Schur complement S is obtained after pivoting out the (nonsingular) leading block A_0 from nonsingular A_{aug}. Hence it is also nonsingular.

The following two lemmas from [2] show how the equations with A can be replaced with equations that involve A_{aug}. The latter reduce to a sequence of equations with the fundamental basis A_0 and with the Schur complement S.

Lemma 2. *Vector x obtained by solving the following sequence of equations*

$$\begin{aligned} A_0\tilde{x} &= b, \\ Sx_C &= V\tilde{x}, \\ A_0x_0 &= b - Cx_C, \\ x &= x_0 + V^T x_C, \end{aligned} \tag{7}$$

is the solution of the equation

$$Ax = b. \tag{8}$$

Lemma 3. *Vector y obtained by solving the following sequence of equations*

$$\begin{aligned} \tilde{y}^T A_0 &= d^T, \\ y_C^T S &= d^T V^T - \tilde{y}^T C, \\ y^T A_0 &= d^T + y_C^T V, \end{aligned} \tag{9}$$

is the solution of the equation

$$y^T A = d^T. \tag{10}$$

2.2 Bordered triangular form of a matrix

Hellerman and Rarick [23] observed that all large, sparse, unsymmetric matrices can be permuted to a *nearly triangular form*. This form may be exploited to determine a pivot order which well preserves the sparsity of the LU factors of the matrix. Their *Preassigned Pivot Procedure* (P^3) was later extended to a more efficient P^4 heuristic and armed with accuracy control in P^5 algorithm [7] (*Precautionary Partitioned Preassigned Pivot Procedure*).

The P^5 heuristic transforms a given matrix to a *spiked triangular form*, i.e., triangular except for several columns called spikes. This form is closely related to a *bordered triangular form*. The latter can be easily obtained from the former by permuting all spikes to the end of the matrix.

Stadtherr and Wood [30] proposed another ordering scheme, SPK1 to permute the matrix to a spiked triangular form. Both P^5 and SPK1 heuristics proceed similarly: they perform backward and forward triangularizations until nontriangularizable blocks are found and spikes are to be chosen. Then the algorithms look for a row with a minimum number of entries, say p, in the active submatrix. Clearly, $p \geq 2$; otherwise triangularization could proceed. The ordering

heuristics will now eliminate $p-1$ spikes to create at least one singleton row and enable the triangularization process to be continued.

To determine a spike, P^5 algorithm analyzes all columns that have an entry in a row of length p and chooses the one that has the maximum number of nonzero entries in rows of length p for a spike. Removing this spike from the active submatrix leads to creating the maximum possible number of rows that have length $p-1$. These rows will be analyzed in the next step of the method.

In the SPK1 algorithm, the search for spikes is simplified since, in general, more than one spike is eliminated at once. First, the appropriate row with the minimum row count p has to be found. If there is a tie, then for each row candidate, a sum of column counts for columns having nonzeros in that row is computed. The row with the largest sum is chosen. In the next step, all columns having nonzeros in this row are analyzed. The one with the smallest column count is assigned to the row. The remaining $p-1$ columns become spikes.

Let us observe that spike selection rule in SPK1 is considerably less involved than that of P^5 which incorporates some local look–ahead effort into the choice of spike. The comparison of the computational results of running SPK1 and P^4 heuristics [30] showed that on flowsheeting matrices, SPK1 ran always considerably faster and often produced smaller number of spikes. However, even then, it led to an important loss of efficiency in the numerical phase of the LU factorization.

In our Schur complement inverse representation, the whole border (the block of spikes) is handled as a matrix C. Spikes removed from A are replaced with appropriate unit vectors to create a triangular A_0 matrix. Hence the situation is much simpler: we are only concerned with a fast reordering to the bordered triangular form of the matrix that keeps the size of the border as small as possible. SPK1 perfectly satisfies these requirements.

2.3 The use of 2x2 pivots

The most important factor that determines the efficiency of the Schur complement inverse representation is the size of the matrix S, i.e., the number of columns in C. In order to reduce this number, we shall extend the structure of the fundamental basis to a block triangular matrix with at most 2x2 diagonal blocks. The motivation for choosing this particular structure of A_0 is twofold. Firstly, such a matrix still remains easily invertible, which is crucial for the efficiency of solves with A and for the computation of the Schur complement (6). Secondly, the P^5 and SPK1 heuristics for finding a bordered triangular form of the matrix can be easily extended to algorithms for finding a *bordered block–diagonal form* of A.

In order to detect a 2x2 pivot we modify the heuristics in the following way. Every time when the list of singleton rows and columns in the active part of A is exhausted, we check the number of nonzero entries p in the row with the minimum row count. If $p > 2$, then $p-2$ spikes are determined and eliminated from the active submatrix (producing at least one row with only 2 nonzero entries).

When the minimum row count p in the active submatrix equals 2, a special routine of [17] (Section 2.1), is used to look for 2x2 pivots. The algorithm scans all rows of length 2 in the active submatrix. It analyzes a pair of columns that have entries in a given row of length 2: it looks for another row of the same length that has nonzero elements in these two marked columns. If such a row is found, the two rows and two columns marked determine a 2x2 block pivot. In the opposite case, one more spike has to be determined in the active submatrix of A.

A reader interested in the issues of implementation of the technique to detect 2x2 pivots is refered to [17] for more details.

2.4 Stability control

The application of P^5 or SPK1 preorderings in a standard LU factorization leads often to serious accuracy problems when pivots which occur in the numerical phase of the decomposition are too small [8, 31]. Spikes have to be additionally reordered then to ensure the stability of the factorization. Unfortunately, such an operation considerably increases the fill–in.

In the approach presented in this paper, all spikes are handled separately through the Schur complement matrix (6). Matrix S is decomposed to explicit LU factors by the method of [16]. Complete pivoting is performed to ensure stability (cf. [15]). This is equivalent to permuting the columns of matrix C (the spikes of A) but it does not affect the structure of the fundamental basis. Consequently, we may hope to obtain a stable representation of S^{-1} under the additional condition that matrix S alone was computed accurately.

Lemmas 2 and 3 and the definition of the Schur complement (6) indicate another critical point for the accuracy of A^{-1} representation, namely, equations with A_0. Since the fundamental basis is block triangular with at most 2x2 diagonal blocks, we impose stability conditions on its diagonal blocks.

To become a 1x1 pivot, an element a_{ij} has to satisfy

$$|a_{ij}| \geq \mu \max_{1 \leq k \leq n} |a_{kj}|, \tag{11}$$

where μ is a prescribed threshold from the interval $(0, 1]$. The accuracy check for a 2x2 pivot candidate is slightly more complicated. Assume such a candidate has been found in rows i and j and columns p and q of A

$$P_2 = \begin{bmatrix} a_{ip} & a_{iq} \\ a_{jp} & a_{jq} \end{bmatrix}, \tag{12}$$

and that a_{ip} is its largest element in the absolute value. We decompose P_2 to the LU form

$$P_2 = \begin{bmatrix} 1 & 0 \\ \tilde{a}_{jp} & 1 \end{bmatrix} \begin{bmatrix} a_{ip} & a_{iq} \\ 0 & \tilde{a}_{jq} \end{bmatrix}, \tag{13}$$

where $\tilde{a}_{jp} = a_{jp}/a_{ip}$ and $\tilde{a}_{jq} = a_{jq} - a_{iq}a_{jp}/a_{ip}$, and check if the following stability criteria is satisfied

$$|a_{ip}| \geq \mu \max_{1 \leq k \leq n} |a_{kp}| \qquad \text{and} \qquad |\tilde{a}_{jq}| \geq \mu \max_{1 \leq k \leq n} |a_{kq}|, \tag{14}$$

with the same μ as in (11).

The conditions (11) and (14) are supposed to prevent large growth factor [21, 25] when solving equations with A_0 or its transpose. The larger μ (the closer to one), the more stable the equations with A_0 become.

Let us observe that conditions (11) and (14) can easily be verified during the reordering phase and that they add very little computational effort. They introduce a useful safeguard against the loss of accuracy in solves with A_0 [24]. Hence when a pivot candidate is found in the reordering phase, it is checked for accuracy and, if the appropriate condition is not satisfied, it is rejected. A column containing the unacceptably small pivot candidate becomes a spike. If a 2x2 pivot candidate (12) is rejected for stability reasons, then one of two columns p and q becomes a spike and the remaining one becomes a candidate for a 1x1 pivot.

Note that by rejecting pivots that do not satisfy conditions (11) or (14), we only delay the manifestation of numerical difficulties to the phase of LU decomposition of the Schur complement. In this phase, however, we hope to have a reliable tool of *complete pivoting* to preserve the accuracy.

Too large a value of μ would result in an increase of the number of pivot rejections and, consequently, in the growth of a size of matrix C and the resulting Schur complement. This would undoubtedly have an undesireable effect on the efficiency of the method. Hence we always look for a compromise value of μ. Our experience (limited to matrices arising in linear programming applications) justifies the choice of $\mu = 0.01$. Such a value of μ has been found sufficient to solve more than 90% of problems from our collection of about 200 linear optimization models, some of them known to be very difficult. For the remaining 10% of problems, the parameter μ had to be adjusted to $\mu = 0.02$ or $\mu = 0.05$ (in 3 cases even to $\mu = 0.1$). These adjustments are done automatically once the program faces numerical difficulties.

3 Updates

We shall now describe the techniques of updating the A^{-1} representation after five different modifications of A. In all cases, the algorithm follows the same scheme: it preserves the sparsity structure of A_0 (in two cases it leaves A_0 unaltered) and it resolves to updates of the Schur matrix.

We thus avoid a sparsity structure analysis that has to be done in the Bartels–Golub updating method (and nonnegligibely contributes to its overall cost, see e.g., [4, 13, 29, 32]). We also benefit from the simplicity of accuracy control in routines that maintain explicit LU factors of S. Moreover, as long as the size of S is kept small (our experience shows that this is almost always the case in linear programming applications), the updates of S remain cheap.

In the following part of this section we shall assume that matrix A is nonsingular and that is has a representation (2)-(4). In case of each update, the modified matrix will be denoted with $\bar{A}$. We assume it is nonsingular.

3.1 Row and column addition

Assume matrix A is bordered with a row r^T and a column c

$$\bar{A} = \begin{bmatrix} A & c \\ r^T & \sigma \end{bmatrix}. \tag{15}$$

Following (2), we shall partition the new row $r^T = (r_1^T, r_C^T)$ and expand subvector r_1^T to $r_0^T \in \mathcal{R}^n$ filling with zeros all positions marked with the rows of V. This new row r_0^T is appended to A_0 while the remaining part r_C^T as well as column c are appended to the border. A new row $e_{n+1}^T \in \mathcal{R}^{n+1}$ appears in a modified V; it means that the last column of the new fundamental basis has to be replaced with the last column of the modified border. As a result we obtain

$$\bar{A}_0 = \begin{bmatrix} A_0 & 0 \\ r_0^T & 1 \end{bmatrix}, \qquad \bar{C} = \begin{bmatrix} C & c \\ r_C^T & \sigma \end{bmatrix} \qquad \text{and} \qquad \bar{V} = \begin{bmatrix} V & 0 \\ 0 & 1 \end{bmatrix}.$$

Hence, the new Schur complement is an $(k+1) \times (k+1)$ matrix

$$\bar{S} = \bar{V}\bar{A}_0^{-1}\bar{C} = \begin{bmatrix} S & c_S \\ r_S^T & \sigma_S \end{bmatrix}. \tag{16}$$

with

$$\begin{aligned} r_S^T &= r_C^T - r_0^T A_0^{-1} C, \\ c_S &= V A_0^{-1} c, \\ \sigma_S &= \sigma - r_0^T A_0^{-1} c. \end{aligned} \tag{17}$$

Summing up, this update resolves to bordering the Schur matrix with a new row and a new column. The following result gives sufficient conditions for the update to be done.

Observation 4. *If $A \in \mathcal{R}^{n\times n}$ and $\bar{A} \in \mathcal{R}^{(n+1)\times(n+1)}$ of (15) are nonsingular, then the update is well–defined.*

Proof. *Nonsingularity of A_0 implies nonsingularity of $\bar{A}_0$. Hence, from the definition, the Schur complement inverse representation of $\bar{A}$ with matrices $\bar{A}_0$, $\bar{V}$ and $\bar{C}$ is well defined.*

The overall cost of this update is dominated by one forward transformation with A_0 (to compute $A_0^{-1}c$), one backward transformation with A_0 (to compute $r_0^T A_0^{-1}$) and the operations to maintain the explicit LU factors of S after a row and a column is added to it (16).

3.2 Column exchange

Assume a column of A is to be replaced with a new one c. Two cases may occur: the removed column is in A_1 (and, consequently, in A_0) or it is one of the columns of the border C.

Case 1. Column p of A_0 has to be removed.
The update does not alter A_0. Column p of A_0 is suspended by adding a new row e_p^T ($e_p^T \in \mathcal{R}^n$) to V. The entering column is appended to C. Hence

$$\bar{A}_0 = A_0, \qquad \bar{C} = \begin{bmatrix} C & c \end{bmatrix} \qquad \text{and} \qquad \bar{V} = \begin{bmatrix} V \\ e_p^T \end{bmatrix},$$

which gives the following Schur complement

$$\bar{S} = \begin{bmatrix} S & c_S \\ r_S^T & \sigma_S \end{bmatrix} \tag{18}$$

with

$$\begin{aligned} r_S^T &= e_p^T A_0^{-1} C, \\ c_S &= V A_0^{-1} c, \\ \sigma_S &= e_p^T A_0^{-1} c. \end{aligned} \tag{19}$$

Consequently, this update needs a row and a column to be bordered to S. Its cost is dominated by the calculation of $e_p^T A_0^{-1}$, $A_0^{-1}c$ and recomputing the explicit LU factors of $\bar{S}$ from those of the old Schur complement.

The reader may quickly verify that the update is well defined.

Case 2. Column $\tilde{p}$ of C is to be removed.
The update leaves A_0 and V unaltered but it needs modifying C to

$$\bar{C} = C + (c - Ce_{\tilde{p}})e_{\tilde{p}}^T,$$

where $e_{\tilde{p}}$ is a $\tilde{p}$-th unit vector in $\mathcal{R}^k$. The new Schur complement

$$\bar{S} = VA_0^{-1}C + VA_0^{-1}(c - Ce_{\tilde{p}})e_{\tilde{p}}^T = S + (VA_0^{-1}c - Se_{\tilde{p}})e_{\tilde{p}}^T, \tag{20}$$

has thus one column replaced. This upate is well defined and, additionally, it is also very cheap: it needs computing $A_0^{-1}c$ and maintaining LU factors of S after column exchange in it.

The modifications presented by now involve the columns of A and can be handled relatively easily. The reason for that comes from the fact that our Schur complement inverse representation is "column oriented", i.e., that we treat specially column border in A. In the worst case, any operation on a column of A can only affect the border C.

In contrast, operations that involve the rows of A have to affect both the fundamantal basis A_0 and the border C. In the approach presented in this paper, they are always decomposed into two steps. At the first one, a row and a column is dropped from the matrix. However, in order to make sure that the new fundamental basis $\bar{A}_0$ is nonsingular, we impose an additional condition that the row and column removed have the same index p. This particular update is the subject of the following section.

3.3 Deletion of the p-th row and column of A

To delete a row and a column from A, we need to replace row p and column p of A_0 with appropriate unit vectors of $\mathcal{R}^n$ and we have to replace row p of C with zeros. We assume here that column p of A remains at position p in A_0; the update cannot be done if this column stays in the border.

To replace column p of A_0 with e_p, we write

$$A_0^{'} = A_0 + (e_p - A_0 e_p)e_p^T.$$

Next, we replace row p of $A_0^{'}$ with e_p^T

$$A_0^{''} = A_0^{'} + e_p(e_p^T - e_p^T A_0^{'}).$$

Simple manipulations give

$$A_0^{''} = A_0 + e_p e_p^T - A_0 e_p e_p^T - e_p e_p^T A_0 + e_p e_p^T A_0 e_p e_p^T. \tag{21}$$

This matrix can be represented in a form of a rank two update of A_0

$$A_0^{''} = A_0 + Y Z^T, \tag{22}$$

where

$$Y = [e_p - A_0 e_p | e_p] \in \mathcal{R}^{n\times 2} \qquad \text{and} \qquad Z = \left[e_p | a_{pp} e_p - A_0^T e_p\right] \in \mathcal{R}^{n\times 2}, \tag{23}$$

with $a_{pp} = e_p^T A_0 e_p = (A_0)_{pp}$.

Applying Sherman-Morrison-Woodbury formula to (22), we obtain

$$(A_0^{''})^{-1} = A_0^{-1} + e_p e_p^T - \frac{1}{e_p^T A_0^{-1} e_p} A_0^{-1} e_p e_p^T A_0^{-1}. \tag{24}$$

The last equation can be combined with a definition of the new border in which row p is replaced by zeros

$$C^{'} = C - e_p e_p^T C,$$

to define the new Schur complement

$$\bar{S} = V(A_0^{''})^{-1} C^{'} = S + \frac{1}{e_p^T A_0^{-1} e_p}(-V A_0^{-1} e_p)(e_p^T A_0^{-1} C). \tag{25}$$

Summing up, this update transforms A_0 to $A_0^{''}$, removes row p and column p from $A_0^{''}$, removes row p from C and requires a rank one change of S. Note that there is no use to keep a one at position $(A_0^{''})_{pp}$. We thus drop row p and column p from $A_0^{''}$. This "collapsing" operation produces $\bar{A}_0 \in \mathcal{R}^{(n-1)\times(n-1)}$. Analogously, the empty row p is removed from $C^{'}$ giving the new $\bar{C}$. It is easy to see that these operations do not affect $\bar{S}$ defined by (25).

A cost of this change of A is comparable to the one of the row and column addition update: we need to solve two eqations with A_0 (to compute $A_0^{-1}e_p$ and $e_p^T A_0^{-1}$) and to maintain explicit LU factors of S after rank one change to it. Additionally, the sparsity structure of A_0 has to be modified to remove row and column from it.

Unfortunately, there is no guarantee for this update to be completed successfully. It might happen that the matrix A with row p and column p removed becomes singular. We cannot detect such a situation up until the update of the LU factorization of S.

We are now ready to pass to the presentation of the following two updates of A.

3.4 Row exchange

Assume that the p-th row of A is to be replaced with a new one: r^T. We shall decompose this update into two steps. First, we apply the procedure of the previous section, i.e., we remove row p and column p from A_0 and we remove row p from C. Next, we add the new row r^T and the (earlier removed) column p to such modified A.

The cost of this update is roughly speaking two times larger than those of the three update techniques presented earlier.

3.5 Row and column deletion

Assume a row p and a column q of A are to be removed. As this update involves a row of A, it has also to be decomposed into two steps. First, we remove row p and column p from A_0 and we remove row p from C. In the second step, we replace column q with the (earlier removed) column p.

The overall cost of this modification is comparable to that of the row exchange update (and is about two times larger than the other three updates).

3.6 Rank one update

We shall be concerned in this section with the following change of A

$$\bar{A} = A + cr^T, \tag{26}$$

where $c, r \in \mathcal{R}^n$. This modification differs considerably from the five updates presented by now, because it affects the whole matrix at a time. It used to draw a lot of attention [10, 13, 14] due to its presence in numerous applications (although in the linear programming approach [19] this correction did not appear). By its nature, rank one update may cause dramatic fill-in in the LU factors [14].

Below, we shall show that a straightforward extension of the Schur complement inverse representation described in this paper can handle this update. Recall that our aim is to solve equations

$$\bar{A}x = b \qquad \text{and} \qquad \bar{A}^T y = d, \tag{27}$$

where x, b, y and d are vectors from $\mathcal{R}^n$.

Similarly to the analysis presented at the beginning of Section 2, we introduce the new augmented matrix

$$\tilde{A}_{aug} = \begin{bmatrix} A & c \\ r^T & -1 \end{bmatrix}. \tag{28}$$

Although there is an important difference between $\tilde{A}_{aug}$ and the previously defined augmented matrix (4) that is manifested by the presence of -1 in the lower right corner of $\tilde{A}_{aug}$, the idea to use the Schur complement remains the same.

Two observations given below show how equations (27) can be replaced with equations that involve $\tilde{A}_{aug}$.

Observation 5. *Vector x obtained by solving the following system of equations*

$$\tilde{A}_{aug} \begin{bmatrix} x \\ x_0 \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{29}$$

is the solution of the equation

$$\bar{A}x = b. \tag{30}$$

Proof. *From (29) it follows that we can eliminate x_0*

$$x_0 = r^T x,$$

and from the remaining part of that equation obtain

$$Ax + cx_0 = Ax + cr^T x = (A + cr^T)x = \bar{A}x = b.$$

Observation 6. *Vector y obtained by solving the following system of equations*

$$\tilde{A}_{aug}^T \begin{bmatrix} y \\ y_0 \end{bmatrix} = \begin{bmatrix} d \\ 0 \end{bmatrix} \tag{31}$$

is the solution of the equation

$$\bar{A}^T y = d. \tag{32}$$

Proof. *Analogous to the proof of the previous observation.*

Let us observe that, given the Schur complement inverse representation (2)-(4) of A, we easily guess the one of $\tilde{A}_{aug}$:

$$\tilde{A}_0 = \begin{bmatrix} A_0 & 0 \\ r^T & 1 \end{bmatrix}, \qquad \tilde{C} = \begin{bmatrix} C & c \\ 0 & -1 \end{bmatrix} \qquad \text{and} \qquad \tilde{V} = \begin{bmatrix} V & 0 \\ 0 & 1 \end{bmatrix}.$$

The new Schur complement has the form

$$\tilde{S} = \begin{bmatrix} S & c_S \\ r_S^T & \sigma_S \end{bmatrix}, \tag{33}$$

where

$$\begin{aligned} r_S^T &= -r^T A_0^{-1} C, \\ c_S &= V A_0^{-1} c, \\ \sigma_S &= -(1 + r^T A_0^{-1} c). \end{aligned} \tag{34}$$

The overall cost of this update is comparable to the one of row and column addition to A. It is dominated by computing $A_0^{-1}c$ and $r^T A_0^{-1}$ and the operations needed to maintain the explicit LU factors of S after a row and a column are added to it.

We would like to mention once again the difference between the rank one update of A and the other modifications presented in the previous sections. We did not derive the Schur complement inverse representation of $\bar{A}$ form that of A. Instead, we replaced equations (27) with (29) and (31), respectively. They involve a larger matrix $\tilde{A}_{aug}$ for which the required inverse representation can easily be obtained.

We pay for this transformation since we need to deal with a matrix in a larger space. Each time an equation with $\bar{A}$ is to be solved, its right hand side is extended to $\mathcal{R}^{n+1}$. When the solution to (29) or (31) is found, the required vector x or y is retrieved from it. Clearly, these operations can be made very efficiently.

3.7 Modifications of S

The updates of A^{-1} inverse representation cause various modifications to the Schur matrix: column exchange, row and column addition or rank one change. We decided to follow [17] and to apply the explicit LU decomposition for handling the inverse of S. One new update — a rank one change — was added to the set of routines that implement the method of [16].

We have already mentioned that the accuracy of S^{-1} inverse representation is crucial for the stability of the method presented in this paper. The use of the explicit LU factors of S facilitates the accuracy control during the decomposition of (6) as well as during the updates that follow it.

The size of the Schur matrix after refactorization step is usually small; it seldom exceeds 100 and very rarely grows over 200. In the latter case, in particular, the use of the sparsity-exploiting LU decomposition [13] to manage the inverse of S might become advantageous. However, this possibility is not exploited in our implementation.

4 Sparsity, accuracy and efficiency

In this section, we shall very briefly address some issues that are crucial for the efficiency and the stability of the method's implementation.

4.1 Storing A_0 and S

Equations with the fundamental basis constitute the main computational effort in the method presented in this paper. Every refactorization needs k forward transformations with A_0 to calculate the Schur matrix (6), (k is the size of S). Each update of A requires one, two or four equations with A_0 or its transpose to be solved.

It is thus crucial to store A_0 in a way that ensures a comfortable access to its rows and columns. It facilitates the implementation of dropping row and column p from A_0 and it allows exploiting the sparsity of A_0 and that of the right hand side vectors in all equations with A_0. Consequently, we store A_0 as a collection of sparse columns and we use row linked lists for row wise access to it. This requires three integer arrays and one double precision array of length n_{A0}, where n_{A0} is the number of nonzero entries in A_0, and two integer arrays of length n (cf. Duff et al. [5]).

In our approach the Schur matrix is decomposed into explicit LU factors. Hence we need one double precision array of size $k \times k$ to remember it.

Additionally, a few integer arrays of size n are needed to store row and column permutations of A and to provide work arrays for an efficient implementation of the SPK1 heuristic.

4.2 Sparsity versus accuracy

There exists a well known conflict in sparse matrix computations between preserving sparsity and ensuring accuracy of an inverse representation.

For example, it often happens that if the factorization uses a preassigned pivot order based only on the analysis of the sparsity structure, then unacceptably small pivots appear in the numerical phase of the decomposition [8, 31]. To preserve the accuracy, the pivot order has to be changed in such case. The new pivot order may then be far from optimal (in the sense of the previously applied heuristic), which would manifest itself in a significant increase of fill-in.

This difficulty is easier to overcome if the pivot choice is made dynamicaly with the progress of factorization [26]. Such an approach is particularly popular in the linear programming applications [13, 29, 32]. Similar technique is applied when the Bartels-Golub updates of sparse LU factors are computed. The choice betwen two available Gaussian elementary operators favorizes sparsity by default. However, seriously unstable operations (causing large growth factor) are rejected: the sparsity is sacrificed in such case for the accuracy. By the way, even then, there is no guarantee for sufficient accuracy of the decomposition [28] although the cases of instability are very rare in practice.

The conflict mentioned above manifests itself also in the method presented in this paper.

In order to ensure a high accuracy of solves with the fundamental basis, we reject columns with too small diagonal elements (or unstable 2x2 blocks) after accuracy tests (11) and (14). Each rejection of the pivot candidate during the SPK1 heuristic creates a new spike, hence increases the size of the border C. Consequently, we have to deal with a larger Schur matrix.

The "sparse" task of the factorization simplifies: equations with A_0 get easier (although their number increases with the number of columns in C). In contrast, the "dense" task of the factorization — a decomposition of the Schur complement — becomes more important.

Our computational experience showed that the number of columns removed from A_0 for stability reasons is small and, except for some rare cases when A is poorly scaled, negligible.

We would like to underline a role of the choice of the suitable form of the fundamental basis for a particular application. We consider this to be the most promising direction for further development of the inverse representation method described in this paper. An extension of the structure of A_0 (e.g., to a block triangular matrix with diagonal blocks that are small but are not limited to the size of 2) could perhaps help decrease the size of the Schur complement.

4.3 Block–update matrix

Lemmas 2 and 3 show how equations with A and its transpose can be replaced with a sequence of two equations with the fundamental basis A_0 and one equation with the Schur matrix S.

We could save on one solve with A_0 if we decided to store the block–update matrix

$$B = A_0^{-1}C, \tag{35}$$

that must nevertheless be computed to construct S (cf. (6)). This option has been mentioned in [12] and practically analyzed by Eldersveld and Saunders [6]. Note that once B is available, the third of equations (7) simplifies to

$$x_0 = A_0^{-1}b - A_0^{-1}Cx_C = \tilde{x} - Bx_C. \tag{36}$$

Analogously, the first two equations of (9) reduce to

$$y_C^T S = d^T V^T - d^T A_0^{-1} C = d^T V^T - d^T B. \tag{37}$$

The computational experience of [6] did not show clear advantage of the use of block–update matrix. There are examples where the matrix B fills dramatically despite the sparsity of A_0 and C. In these cases, multiplications with B may become prohibitively expensive. Hence, it is advantageous to retrieve only the necessary information from the appropriate rows of B ($S = VB$), accept the need of one more solve with A_0, and forget the block–update matrix.

The block–update option has also another drawback of sometimes excessive storage requirements. This, however, seems less important due to a modern tendency to sacrifice storage economy in order to execute faster.

5 Numerical results

The method presented in this paper has been implemented and applied to handle the working basis updates in the new linear programming approach [19]. The computational results discussed here are derived from this particular application.

Table 1: Problems statistics.

Problem	Original dimensions			After presolve		
	n_r	n_c	nonz	n_r	n_c	nonz
25fv47	820	1571	10400	768	1534	9957
80bau3b	2235	9301	20413	1965	8736	19048
bnl2	2280	3489	13999	1848	3007	12458
degen3	1503	1818	24646	1503	1818	24363
dfl001	6071	12230	35632	5907	12065	35021
fit2p	3000	13525	50284	3000	13525	50284
ganges	1309	1681	6912	835	1168	5423
greenbea	2389	5302	30715	1848	3886	23112
pilot	1440	3449	41092	1340	3326	40454
pilotnov	951	1968	12186	830	1871	11492
sierra	1222	2016	7252	1129	2008	6956
stocfor2	2157	2031	8343	1968	1854	7064
stocfor3	16675	15695	64875	15336	14382	55088
truss	1000	8806	27836	1000	8806	27836
pds-02	2953	7535	21252	2601	7172	15499
pds-06	9881	28655	82269	9114	27852	60038
pds-10	16558	48763	140063	15579	47729	102960
GE	10339	11098	53763	8350	9699	34650
NL	7195	9718	102570	6472	9246	39278
WORLD2	3723	3133	17800	2588	2421	10063

We applied the method to solve a large collection of linear optimization problems that contains about 200 models and includes the Netlib suite of 95 problems [11]. Solution of a nontrivial linear program needs thousands or tens of thousand of iterations, each slightly modifying the working basis. Hence it creates perfect conditions to test a new updating method.

The abovementioned linear optimization approach succeeded to solve almost all tests from our collection of difficult programs, which confirmed the reliability of the Schur complement inverse representation presented in this paper. We have chosen a subset of 20 representative difficult problems to illustrate the behavior of our updating method. Table 1 presents the problems statistics in their original form and after running a presolve analysis on them [18] (n_r, n_c and $nonz$ are the numbers of rows, columns and nonzero entries, respectively). Problems GE, NL and WORLD2 belong to our collection of linear optimization test examples[1]; the remaining problems are available via Netlib.

Table 2 collects information on the solution of these problems. It contains the statistics of the

[1]To get these problems, send a request to the author: gondzio@ibspan.waw.pl

Table 2: Solution statistics.

Problem	Working bases			iters	Updates				Schur		2x2
	n_0	n_{max}	n_{opt}		*RCadd*	*Crep*	*Rrep*	*RCdel*	k_F	k_U	
25fv47	477	623	609	8884	1128	6657	181	991	71	102	38
80bau3b	0	1811	1809	14620	3556	8909	407	1832	12	47	16
bnl2	1050	1448	1432	7269	1815	3298	685	1775	48	83	36
degen3	712	1148	1140	6522	1423	3710	413	1387	88	117	0
dfl001	5860	5905	5904	85964	11732	67865	2987	9492	60	97	1
fit2p	3000	3000	3000	14233	0	14233	0	0	12	51	2
ganges	791	811	811	799	27	773	0	3	33	64	1
greenbea	1667	1768	1746	22206	1005	20184	84	988	74	101	41
pilot	162	1254	1250	16643	3615	9691	783	2711	218	247	44
pilotnov	556	703	702	3545	267	3126	17	149	129	158	54
sierra	499	584	571	703	109	549	8	38	40	76	5
stocfor2	966	1336	1336	919	408	312	195	40	16	50	1
stocfor3	7516	10651	10651	7750	3639	2440	1470	527	110	140	0
truss	999	1000	999	10265	16	10360	0	0	60	109	397
pds-02	2431	2495	2486	2823	113	2644	10	77	2	46	0
pds-06	8525	8716	8716	17885	862	16256	111	786	6	49	0
pds-10	14575	14868	14868	44746	1728	41262	338	1678	8	51	0
GE	3845	5497	5436	15359	4684	5844	2343	3242	130	159	51
NL	1331	5345	5341	45340	15031	14783	5271	11773	78	109	17
WORLD2	72	1466	1396	28807	9011	6623	5708	8455	45	82	30

size of the working bases: n_0, n_{max} and n_{opt} are their initial, maximum and final (corresponding to the optimum) dimensions, respectively. The following columns of Table 2 report the number of iterations to reach optimality, *iters* and the numbers of different working basis updates: row and column addition, *RCadd*, column replacement, *Crep*, row replacement, *Rrep* and row and column deletion, *RCdel*. Its last three columns give an insight into the efficiency of the SPK1 heuristic; they contain the maximum sizes of the Schur complement after refactorizations, k_F and during the updates, k_U and the maximum number of 2x2 pivots.

The results collected in Table 2 bring a lot of useful information. The most important is that the linear programming bases can, in practice, always be permuted to the bordered block triangular form with a size of the border kept manageable. The dimension of the border after refactorization is surprisingly small for some nontrivial large scale bases.

These results confirm also advantages of the use of 2x2 pivots in the fundamental basis. If a triangular fundamental basis were to be used, then the size of the Schur matrix would

Table 3: Efficiency of the representation.

Problem	Working basis				Factorization				Update
	n	$nonz$	k	2x2	t_{order}	t_S	t_{LU}	t_{Factor}	t_{Update}
25fv47	606	3185	68	36	0.07	0.05	0.04	0.16	0.011
80bau3b	1803	5129	10	12	0.15	0.01	0.00	0.16	0.009
bnl2	1445	4126	45	35	0.11	0.05	0.01	0.17	0.012
degen3	1145	8552	86	0	0.18	0.10	0.05	0.33	0.017
dfl001	5867	14122	50	0	0.50	0.36	0.02	0.88	0.011
fit2p	3000	34208	10	0	0.57	0.13	0.00	0.70	0.061
ganges	811	4302	33	0	0.07	0.02	0.00	0.09	0.004
greenbea	1756	10371	63	35	0.20	0.10	0.03	0.33	0.010
pilot	1236	17710	196	25	0.33	0.55	1.00	1.88	0.086
pilotnov	699	3907	129	50	0.08	0.10	0.20	0.38	0.015
sierra	577	1258	36	5	0.04	0.02	0.00	0.06	0.003
stocfor2	1274	3272	13	0	0.08	0.01	0.01	0.10	0.006
stocfor3	10363	27922	108	0	0.81	0.95	0.07	1.83	0.037
truss	1000	3794	56	358	0.06	0.07	0.03	0.16	0.007
pds-02	2484	5078	0	0	0.20	0.00	0.00	0.20	0.005
pds-06	8716	17866	4	0	0.72	0.03	0.00	0.75	0.013
pds-10	14831	30136	2	0	1.27	0.02	0.00	1.29	0.018
GE	4439	10277	116	14	0.38	0.44	0.09	0.91	0.022
NL	5106	18845	60	9	0.49	0.30	0.03	0.82	0.025
WORLD2	1445	4482	40	26	0.12	0.05	0.01	0.18	0.011

increase, on the average, by the number of 2x2 pivots (in a case of TRUSS, this would be a real disaster). The clear advantages of the use of 2x2 pivots undoubtedly encourage to look for other more suitable forms of the easily invertible fundamental bases such that the size of the Schur complement could be further reduced.

Let us now proceed to the analysis of the CPU time efficiency of the method presented. In particular, we shall concentrate on the analysis of the speed of the new updates. The efficiency of the factorization step followed by the simplex updates, i.e., column replacements, has already been analyzed in [17] and compared with efficient implementations of the Bartels-Golub updates of sparse LU factors: MA28 of Duff [4], LUSOL of Gill et al. [13] and LA05 of Reid [29]. The updates handled with the Schur complements were shown there to be slightly faster, on the average, than the LA05 implementation incorporated into XMP code [27].

The matrix modifications analysed in this paper are more complicated, hence more expensive from the computational point of view. In particular, row replacement and row and column deletion are about two times slower than the remaining updates. Fortunately, as the results

collected in Table 2 show, they are less frequent than row and column addition and column exchange.

Table 3 reports CPU time in seconds of the factorization and the average time of a single update for representative working bases in the earlier selected linear programs. The results reported here were obtained by running our FORTRAN implementation of the method on a 33MHz SUN SPARC 10 workstation. The program was compiled with an option -O. The results start with statistics of the working basis: its size, n, the number of nonzero entries, $nonz$, the size of the Schur matrix just after the refactorization, k and the number of 2x2 pivots found. The following columns contain the CPU times for a factorization and for a single update (an average of 50 updates directly following this factorization). Additionally, in the factorization time we distinguish time spent in the SPK1 heuristic to reorder the matrix to a bordered block triangular form, t_{order}, time spent to compute the Schur matrix (6), t_S, time to decompose S to the explicit LU factors, t_{LU} and their sum $t_{Factor} = t_{order} + t_S + t_{LU}$.

Analysis of Table 3 leads to the following observations. The factorization step is quite efficient unless the Schur matrix grows to a large size in which case the computation of S and its LU decomposition become dominating terms. (Note that these steps may benefit the most from the parallelisation.) The average time of the update is reasonable: it varies from 1% to 9% of the factorization time.

6 Conclusions

We have described a set of procedures to update the Schur complement inverse representation of a sparse, unsymmetric matrix after various modifications. We have demonstrated practical efficiency of the approach on a representative set of problems that arise in linear programming applications.

The method was shown to be a reliable and acceptably efficient tool for handling general updates (including a rank one correction) of sparse, unstructured matrices.

We have presented a useful extension of the SPK1 heuristic [30] to a new one that reorders a given matrix to a bordered block triangular form with 2x2 pivots allowed on the diagonal.

The method described in this paper splits the factorization and update steps into sparse and dense tasks. Both of them could benefit much from parallelisation. The latter additionally offers the simplicity of accuracy control in the updates (and is well-suited to vectorization).

Summing up, the method presented in this paper seems to be an attractive alternative to the Bartels-Golub updating scheme.

Acknowledgements
The author is grateful to Artur Swietanowski for reading this paper very carefully.

References

[1] Bartels R.H. and G.H. Golub (1969) The simplex method of linear programming using LU decomposition, *Communication of ACM* 12, pp. 266-268.

[2] Bisschop J. and A. Meeraus (1977) Matrix augmentation and the partitioning in the updating of the basis inverse, *Mathematical Programming* 13, pp. 241-254.

[3] Cottle R.W. (1974) Manifestations of the Schur complement, *Linear Algebra and its Applications* 8, pp. 189-211.

[4] Duff I.S. (1977) MA28 — A set of fortran subroutines for sparse unsymmetric linear equations, Report AERE R8730, Atomic Energy Research Establishment, Harwell, England.

[5] Duff I.S., A.M. Erisman and J.K. Reid (1987) *Direct methods for sparse matrices*, Oxford University Press, New York, 1987.

[6] Eldersveld S. and M. Saunders (1992) A block–LU update for large-scale linear programming, *SIAM Journal on Matrix Analysis and Applications* 13, No 1, pp. 191-201.

[7] Erisman A.M., R.G. Grimes, J.D. Lewis and W.G. Poole (1985) A structurally stable modification of Hellerman-Rarick's P^4 algorithm for reordering unsymmetric sparse matrices. *SIAM Journal on Numerical Analysis* 22, pp. 369-385.

[8] Erisman A.M., R.G. Grimes, J.D. Lewis, W.G. Poole and H.D. Simon (1987) Evaluation of orderings for unsymmetric sparse matrices. *SIAM Journal on Scientific and Statistical Computing* 8, pp. 600-624.

[9] Fletcher R. and J.A.J. Hall (1989) Towards reliable linear programming, Technical Report NA/120, Department of Mathematics and Computer Science, University of Dundee.

[10] Fletcher R. and S.P.J. Matthews (1985) A stable algorithm for updating triangular factors under a rank one change, *Mathematics of Computations* 45, No 172, pp. 471-485.

[11] Gay D.M. (1985) Electronic mail distribution of linear programming test problems, Mathematical Programming Society COAL Newsletter 13, pp. 10-12.

[12] Gill P.E., W. Murray, M.A. Saunders and M.H. Wright (1984) Sparse matrix methods in optimization, *SIAM Journal on Scientific and Statistical Computing* 5, pp. 562-589.

[13] Gill P.E., W. Murray, M.A. Saunders and M.H. Wright (1987) Maintaining LU factors of a general sparse matrix, *Linear Algebra and its Applications* 88/89, pp. 239-270.

[14] Gille P. and E. Loute (1982) Updating the LU Gaussian decomposition for rank–one corrections; application to linear programming basis partitioning techniques, Cahier No 8201, Facultes Universitaires Saint-Louis, Brussels, Belgium.

[15] Golub G.H. and C. Van Loan (1989) *Matrix Computations*, (2nd ed.) The Johns Hopkins University Press, Baltimore and London, 1989.

[16] Gondzio, J. (1992) Stable algorithm for updating dense LU factorization after row or column exchange and row and column addition or deletion, *Optimization* 23, pp. 7-26.

[17] Gondzio, J. (1994) On exploiting original problem data in the inverse representation of linear programming bases, *ORSA Journal on Computing* 6, No 2, pp. 193-206.

[18] Gondzio, J. (1994) Presolve analysis of linear programs prior to applying the interior point method, Technical Report 1994.3, Department of Management Studies, University of Geneva, Switzerland.

[19] Gondzio, J. (1994) Another simplex-type method for large scale linear programming, Technical Report ZTSW-1-G244/94, Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland.

[20] Gondzio J. and A. Ruszczynski (1992). A sensitivity method for basis inverse representation in multistage stochastic linear programming problems, *Journal of Optimization Theory and Applications* 74, pp. 221-242.

[21] Gould N.I.M. (1991) On growth in Gaussian Elimination with complete pivoting, *SIAM Journal on Matrix Analysis and Applications* 12, No 2, pp. 354-361.

[22] Hager W. (1989) Updating the inverse of a matrix, *SIAM Review* 31, No 2, pp. 221-239.

[23] Hellerman E. and D.C. Rarick (1971) Reinversion with the preassigned pivot procedure, *Mathematical Programming* 1, pp. 195-216.

[24] Higham N.J. (1989) The accuracy of solutions to triangular systems, *SIAM Journal on Numerical Analysis* 26, pp. 1252-1265.

[25] Higham N.J. and D.J. Higham (1989) Large growth factors in Gaussian Elimination with pivoting, *SIAM Journal on Matrix Analysis and Applications* 10, No 2, pp. 155-164.

[26] Markowitz, H.M. (1957) The elimination form of the inverse and its application to linear programming, *Management Science* 3, pp. 255–269.

[27] Marsten R.E. (1981) The design of the XMP linear programming library, *ACM Transactions on Mathematical Software* 7, pp. 481-497.

[28] Powell M.J.D. (1987) An error growth in the Bartels-Golub and Fletcher-Matthews algorithms for updating matrix factorizations, *Linear Algebra and its Applications* 88/89, pp. 597-621.

[29] Reid J.K. (1982) A sparsity–exploiting variant of the Bartels–Golub decomposition for linear programming bases, *Mathematical Programming* 24, pp. 55–69.

[30] Stadtherr M.A. and E.S. Wood (1984) Sparse matrix methods for equation–based chemical process flowsheeting — I, reordering phase, *Computers and Chemical Engineering* 8, No 1, pp. 9-18.

[31] Stadtherr M.A. and E.S. Wood (1984) Sparse matrix methods for equation–based chemical process flowsheeting — II, numerical phase, *Computers and Chemical Engineering* 8, No 1, pp. 19-33.

[32] Suhl U. and L. Suhl (1990) Computing sparse LU factorizations of large–scale linear programming bases, *ORSA Journal on Computing* 2, pp. 325-335.