ASSURING NUMERICAL STABILITY IN THE PROCESS OF MATRIX REFACTORIZATION WITHIN LINEAR PROGRAMMING PACKAGE ON PC

INFORMATICA 4/89

Keywords: linear programming, HR matrix, matrix factorization, supersparsity, microcomputers

Janez Barle and Janez Grad
Ekonomska fakulteta Borisa Kidriča, Ljubljana

ABSTRACT: One of the most challenging tasks of those who develop linear programming software is development of quick, efficient and reliable matrix refactorization subroutine. The paper describes the implementation of this subroutine within the PC-LIP programming package, which we developed for the IBM-PC personal computers. A major design criterium for PC-LIP was to combine storage economy with numerical stability. The former was achieved using data structures which exploit super-sparsity and the latter implementing state of the art algorithms for basis matrix refactorization. These algorithms were combined with different tools for improving numerical stability. The resulting subroutine performs satisfactory even on a badly scaled data which are quite common in practice.

ZAGOTAVLJANJE NUMERIČNE STABILNOSTI MED REFAKTORIZACIJO BAZNE MATRIKE V PROGRAMSKEM PAKETU ZA LINEARNO PROGRAMIRANJE NA OSEBNEM RAČUNALNIKU: Razvoj hitrega, učinkovitega in zanesljivega podprograma za faktorizacijo bazne matrike spada med najbolj zahtevne naloge pri izgradnji programske opreme za linearno programiranje. Članek podaja opis implementacije tega podprograma v okviru programskega paketa PC-LIP, ki smo ga razvili za IBM kompatibilne osebne računalnike. Pri načrtovanju programskega paketa PC-LIP je bil glavni cilj vskladitev ekonomične izrabe pomnilnika z numerično stabilnostjo. To je bilo doseženo predvsem z uporabo podatkovnih struktur, ki izrabljajo hiperrazpršenost, in najbolj učinkovitih sodobnih algoritmov za izvajanje refaktorizacije bazne matrike. Ti algoritmi so bili kombinirani z različnimi postopki za zagotavljanje numerične stabilnosti. Razviti podprogram za refaktorizacijo je bil uspešen tudi na slabo pogojenih problemih, ki so v praksi dokaj pogosti.

Introduction

A major concern of those who develop linear programming software is how to produce efficient, reliable and numericaly stable computational procedures for solving large-scale problems. When microcomputer software is considered, the problem of fitting algorithms and data structures within the limited storage is also very important. Contemporary literature on computational linear programming offers a plethora of different methods for achieving these goals. Roughly speaking these algorithms and techniques can be divided into following groups:

i) Data structures which are designed for exploiting sparsity in LP data. They can be also tailored for utilization of structure and distribution of nonzero elements contained in LP matrix.

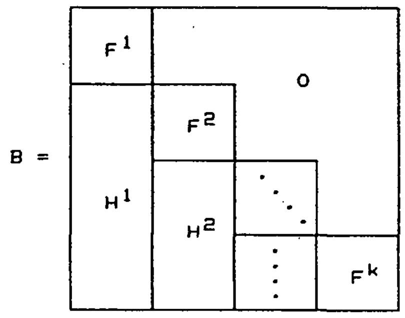
ii) Revised simplex algorithm with product form factorization of basis matrix. State of the art implementations of this method are based on LU factorization.

iii) Subroutines for refactorization of basis matrix. They are designed for controlling size and accuracy of product form factorization of basis matrix during the LP solving process.

Each of these groups offers a great choice of more or less elaborate techniques or algorithms. Sometimes it is not practical to use only the most advanced methods. For example, it is often desirable to sacrifice some computational speed in favour of reliability or storage economy. But in any case refactorization subroutine have to do its job correctly. Refactorization subroutine is also very important part of PC-LIP package which we developed for the IBM-PC personal computers. Practical experiences on real life problems show as that this package is capable to solve even very badly scaled problems. We attribute such a performance mainly to the careful implementation of matrix refactorization subroutine. Our implementation can be described as a successful combination of several state of the art numerical methods with tools for controlling numerical stability. That is why description of our refactorization subroutine may be of interest.

Analysing Structure of Basis Matrix

Refactorization subroutine starts with analytical phase, where the structure of matrix nonzero elements is analysed. In our implementation Hellerman and Rarick algorithm (Hellerman, Rarick, 1971) is used for this purpose. This is quite a famous algorithm which is de facto standard for analytical phase implementations. Results of this algorithms are row and column permutations which transform matrix into form of so called HR matrix. Every HR matrix can be, after suitable rearangement of rows and columns, represented in the form of block lower triangular matrix:



where $F^i (i=1,...,k)$ are square matrices and H^i are rectangular matrices. HR matrices are distinguished from general block lower triangular matrices by structure of matrices F^i. These matrices are always nonsingular. When their dimension exceed 1*1 they are called bumps or external bumps and must have structure similar to one on a next picture, where # is symbol for element which must be different from zero and * symbol for element which can be both zero or nonzero.

$$F^i = \boxed{\begin{array}{ccccccc} \# & & & & & & \# \\ * & * & & & \# & & * \\ * & * & \# & & * & & * \\ * & * & * & \# & * & & * \\ * & * & * & * & * & & * \\ * & * & * & * & * & \# & * \\ * & * & * & * & * & * & * \end{array}}$$

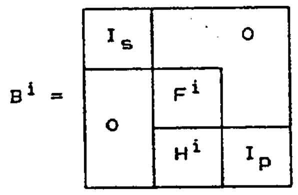
Columns with at least one nonzero above the main diagonal are called spikes. There are two rules concerning spikes:

i) Nonspike columns within bump must have nonzero diagonal elements.

ii) Last column within the bump is a spike having a nonzero uppermost element.

Typical overall number of spikes is much smaller than the number of columns within the matrix. This is an important fact which can be exploited for the economical storing of the factorized basis matrix. It is easy to prove that when product form factorization is formed, only those elementary matrices which correspond to spikes must be actually computed and stored. Other matrices from the product can be replaced by pointers to the non-spike columns (Chvatal, 1983).

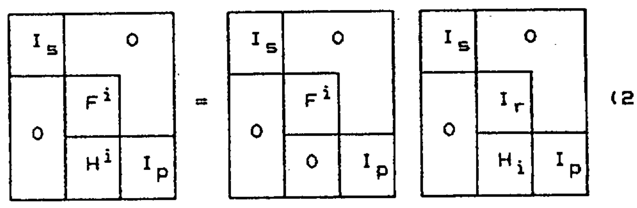
It is easy to check that matrix B with described structure can be represented as a product of matrices having a following form:



where F^i and H^i are situated in the same rows and columns as in matrix B. I_s and I_p are unit matrices of dimension s and p respectively. It is assumed that s and p are numbers of columns to the left and to the right of matrix F^i, which is of dimension r $(s+r+p = m)$. Therefore

$$B = B^1 B^2 \dots B^k \tag{1}$$

Adequate definition for this identity can be "generalized product form of matrix B". B^i can be defined as generalized elementary matrices (ordinary elementary matrices, which are contained in product form factorization, can differ from identity matrix only in one column). For matrices structured in such a way the following factorization formula is valid:



This identity explains why elementary matrices within particular bump can be computed completely independent from other parts of matrix.

If submatrix F^i is a bump, then factorization (2) is called splitting the bump (Helgason, Kennington, 1982). Main purpose for its usage is to reduce number of nonzero elements in the product form factorization of B. Fill in (creating of new nonzero elements) during the factorization of B^i is restricted to r rows which belong to external bump F^i. It is an improvement if compared with the usual product form factorization, where creation of new nonzero elements is possible in rows belonging to submatrix H^i as well. Experiences show that this approach saves computer storage in spite of some overhead which is necessary to store additional r elementary matrices (Hellerman, Rarick, 1971, Helgason, Kennington, 1982).

Numerical Phase of the Algorithm

Our algorithm for the numerical phase of refactorization which includes splitting the bump is presented in the continuation. This algorithm is modification of recent algorithm (Helgason, Kennington, 1982) in which we included additional techniques for assuring numerical stability. It was necessary due to the fact that in the mentioned algorithm well scaled matrix was assumed. This assumption is in general quite a realistic one because many mainframe programming packages use some procedures for automatic scaling of data prior to applying the revised simplex algorithm. For example, this is true when MPSX/370 is considered (Benichou et al, 1977). However, such kind of procedure is not included in the PC-LIP. We avoided this for the sake of storage economy. The use of scales for rows and columns requires additional storage and, what is more important, practicaly prevents employing of such data structures which take advantage of supersparsity. This is a characteristics of large scale problems which means that the number of distinct numerical values in the problem matrix is usually much smaller than the number of nonzero coefficients (Greenberg, 1976). In the PC-LIP supersparsity is exploited in a standard way: nonzero values within problem matrix are represented by pointers to the table of all distinct nonzero values (Barle, Grad, 1987).

When basis matrix is not well scaled, automatically or by means of proper problem formulation, preassigned pivot can appear to be too small and for this reason inadequate. Two cases, which must be treated differently are:

1. Inadequate pivot is situated within the external bump. In such a case its corresponding column can be treated in a same way as a spike. This means that such columns are included in the process of "spike swapping" (Helgason, Kennington, 1980). In fact this procedure is a variant of partial pivoting which is restricted to spikes within the same bump.
2. Inadequate pivot belongs to column which is outside the bumps (column from the triangle part of the matrix). In this case the only solution is to permute this pivot to the right bottom of the matrix. Such pivots will be referred to as "unstable pivots".

In the continuation of the paper we describe our implementation, which includes handling of above cases.

Algorithm S [Product form factorization for a HR matrix including splitting the bump]

Preassigned sequence of pivots is represented with vector C, consisting of column indices, and vector R, consisting of row indices. It is assumed that these sequences are the results of Hellerman and Rarick's algorithm. Other information obtained with this algorithm can be included into R and C using the following method: indices of spike columns are stored in C with opposite (negative) sign, as well as components of R where external bumps are beginning. Algorithm's input is also basis matrix B, which is of dimension m and parameter TPIVR ("pivot relative tolerance"). All pivots y_r, for which the inequality $|y_r| < \text{TPIVR}*y_{max}$ holds, where y_{max} is the largest absolute value of available pivots, are counted as inadequate. Typical values for TPIVR are 0.001 or 0.01.

```
S0: [Divide the pivots into stable and
     unstable]
    a) Set
       (i)   $n = m$ for the number of stable
             pivots
       (ii)  TPIVR = 0.001
       (iii) $i = 1$
    b) If $i>m$, go to S1.
    c) If $R_i<0$, go to S0 g).
    d) Set
       (i)   $r = R_i$
       (ii)  $l = C_i$
       (iii) $y_{max} = \max_k |B_{kl}|$
    e) If $|B_{rl}| < \text{TPIVR}*y_{max}$, set
       (i)   for $j = i, \dots, n-1$:
                $R_j = R_{j+1}$ and $C_j = C_{j+1}$
       (ii)  $R_n = r$ and $C_n = l$
       (iii) $n = n-1$
    f) Set $i = i+1$ and go to S0 b)
    g) Set
       (i)   $t$ = number of columns in this
             external bump
       (ii)  $i = i + t$
    h) Go to S0 b)
```

```
S1: [Initialize]
    a) Set
       (i)   $i = 1$  (pivot counter)
       (ii)  $l = C_i$  (pivot column index)
       (iii) $r = |R_i|$  (pivot row index)
       (iv)  $j = 1$  (counter for elementary
                      matrices)
    b) Alocate storage for
       (i)   $y$ - real vector of dimension $m$
       (ii)  ETA-file (data structure for
                      storing matrices $E_j$).
    c) Set  $y = B_{*l}$ ($l^{th}$ column of the
                      matrix $B$).
    d) If  $R_i < 0$, go to S5.
S2: [Obtain new lower triangular elementary
     matrix]
    a) Set  $z = y$.
       Vector $z$ is the $r^{th}$ column of the
       elementary matrix $E_j$.
    b) Set  $j = j+1$.

S3: [Test for the last stable pivot]
    a) If  $i=n$, go to S16.
    b) If  $i<m$, set
       (i)   $i = i+1$
       (ii)  $l = |C_i|$
       (iii) $r = |R_i|$
       (iv)  $y = B_{*l}$
S4: [Test for External Bump]
    If  $R_i>0$, go to S2.
S5: [Initialize for Bump]
    a) Set
       (i)   $d = j$ (current length of ETA file)
       (ii)  $p = i$ (first column in this
                     external bump)
       (iii) $b$ = number of columns in this
                   external bump
       (iv)  $t = i+b-1$ (last column in this
                         external bump)
    b) Set  $k = p$.
    c) If  $k>t$, go to S6.
    d) If  $C_k<0$, set  $k = k+1$ and go to S5 c).
    e) Set
       (i)   $u = C_k$
       (ii)  $v = R_k$
       (iii) $y_{max} = \max_q |B_{qu}|$
    f) If  $|B_{vu}| <$ TPIVR$*y_{max}$, set $C_k = - C_k$
    g) Set  $k = k+1$ and go to  S5 c).
S6: [Obtain new lower triangular elementary
     matrix]
    a) Set
          $z_k = \begin{cases} y_k & \text{, for } k = |R_s| \quad (i \le s \le t) \\ 0 & \text{, otherwise} \end{cases}$
       Vector $z$ is the $r^{th}$ column of the
       elementary matrix $E_j$.
    b) Set  $j = j+1$.
S7: [Test for end of bump]
    a) If  $i=t$, go to S11.
    b) Set
       (i)   $i = i+1$
       (ii)  $l = |C_i|$
       (iii) $r = |R_i|$
S8: [Test for spike]
    If  $C_i>0$, set  $y = B_{*l}$ and go to S6.

S9: [Spike update]
    Solve system of linear equations
       $E_d \cdots E_{j-1} y = B_{*l}$
S10: [Swap spikes if  $|y_r| <$ TPIVR$*y_{max}$]
    a) Compute $y_{max} = \max |y_k|$
       for $k \in (R_i,\ldots,R_t)$
    b) If  $|y_r| \ge$ TPIVR$*y_{max}$, go to S6.
    c) Obtain new $l$, for which
       $l = |C_s|$ ($i<s\le t$, $C_s<0$).
       Criteria for choosing $l$ is partial
       pivoting inside row $r$.
    d) Solve system of equations
       $E_d \cdots E_{j-1} y = B_{*l}$.
    e) Interchange $C_i$ in $C_s$ and go to S6.
S11: [Obtain new upper triangular elementary
      matrix]
    a) Set
          $z_k = \begin{cases} 1 & \text{, for } k = r \\ y_k & \text{, for } k = |R_s| \quad (s<i) \\ 0 & \text{, otherwise} \end{cases}$
       Vector $z$ is the $r^{th}$ column of the
       elementary matrix $E_j$.
    b) Set
       (i)   $j = j+1$
       (ii)  $i = i-1$
       (iii) $l = |C_i|$
       (iv)  $r = |R_i|$
S12: [Test for beginning of bump]
    If  $i=p$, set $y = B_{*l}$ and go to S14.
S13: [Test for spike]
    a) If  $C_i>0$, set  $i = i-1$ and go to S12.
    b) If  $C_i<0$, solve system of equations
       $E_d \cdots E_{j-1} y = B_{*l}$
       and go to S11.
S14: [Obtain new lower triangular elementary
      matrix]
    a) Set
          $z_k = \begin{cases} 1 & \text{, for } k = r \\ y_k & \text{, for } k = |R_s| \quad (s>t) \\ 0 & \text{, otherwise} \end{cases}$
       Vector $z$ is the $r^{th}$ column of the
       elementary matrix $E_j$.
    b) Set  $j = j+1$
S15: [Test for end of bump]
    a) If  $i=t$, go to S3.
    b) If  $i<t$, set
       (i)   $i = i+1$
       (ii)  $l = |C_i|$
       (iii) $r = |R_i|$
       (iv)  $y = B_{*l}$
    c) Go to S14.

S16: [Partial pivoting]
    a) If  $m=n$, go to S17.
    b) Set  $i = n+1$
    c) Sort columns having indices from $C_i$ to
       $C_m$ in such a way that the number of
       their nonzero elements is increasing.
    d) In the submatrix containing the rows
       from $R_i$ to $R_m$ and the columns from $C_i$
       to $C_m$ perform Gaussian elimination
       with partial pivoting.
S17: [End]
   Product form of B including splitting the
   bump is obtained.
```

At the termination of algorithm S, matrix B is represented as a product of elementary matrices E_j:

$$B = E_1E_2 \ldots E_{j-1} \qquad (3)$$

After obtaining this new factorisation of B, its accuracy must be tested. State of the art method for doing this is the so called Aird-Lynch estimate (Rice, 1985). If this estimate shows that (3) is not accurate enough, algorithm S must be repeated with larger value of TPIVR. In our implementation of algorithm S within the PC-LIP, old value of TPIVR is multiplied by 10.

Partial pivoting within step S10 c) can be performed by using subroutine BTRAN, which can be restricted to only those elementary matrices which belong to the current bump (Saunders, 1976). BTRAN (Backward TRANsformation) is a historical name for the subroutine which solves systems of the form $B^Ty = d$, where B is in a product form. The systems of the form $Bz = u$, which appear within several steps of the S algorithm, can be solved by using subroutine FTRAN (Forward TRANsformation). BTRAN and FTRAN are also important subroutines within the revised simplex algorithm.

If row and column permutations determined by R and C are taken into account, all matrices E_i $(i=1,2,\ldots,j-1)$ are either upper triangular or lower triangular, but they are intermixed. That is why (3) is not LU factorization of B. For this reason the revised simplex algorithm with ordinary product form of basis matrix must be applied after performing the refactorization (as it is in PC-LIP). It is not possible to use those algorithms which use and maintain LU format of the basis matrix, for example Forrest and Tomlin method (Forrest, Tomlin, 1972).

If one wishes to use LU format of the basis matrix, splitting the bump can be used only partially on an overall bump or kernel. Kernel is that part of HR matrix which is obtained after the lower and upper triangles have been removed from the matrix. Recently an algorithm was proposed (Helgason, Kennington, 1982) which performs splitting the bump while maintaining the LU format. We briefly sketch how our methods for handling the unstable columns can be incorporated in this algorithm.

By rearangement of rows and columns, the HR matrix may be placed in the following form:

$$B' = \begin{array}{|c|c|c|} \hline U & V & W \\ \hline O & L & T \\ \hline O & M & N \\ \hline \end{array}$$

L and U are lower and upper triangular matrix respectively, O are zero matrices of suitable dimensions. We use T instead of O which is used in the mentioned algorithm (Helgason, Kennington, 1982). This enables transfer of nonstable columns to the rightmost part of the matrix. It is easy to check that for matrix B' the following factorization is valid:

$$B' = \begin{array}{|c|c|c|} \hline I & O & O \\ \hline O & L & T \\ \hline O & M & N \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline U & V & W \\ \hline O & I & O \\ \hline O & O & I \\ \hline \end{array}$$

LU factorization can be performed in usual way for the first matrix at the right hand side. The second matrix is already upper triangular. Due to the fact that a product of two upper triangular matrices is also an upper triangular matrix, LU factorization of B' is obtained.

Conclusions

The matrix refactorization subroutine as described in the paper has been included in the PC-LIP linear programming software package. Our main contribution was that we have combined the already known methods for "splitting the bump" with some methods for assuring numerical stability. The algorithm was tested on many real life problems and proved to be stable even on a very badly scaled data.

Algorithm satisfies also with respect to the computational speed. Unfortunately we have not yet had an opportunity for comparising its performance with some other algorithm performance. It is possible however to measure the amount of reinversion computational time in overall run time. Another interesting test is to examine the effect of inversion frequency on the solution time. We performed these two test on a real life problem with 342 constraints, 454 structural variables and 2048 nonzero elements. With the inversion frequency 20, the optimal solution was obtained after 221 iterations and 565.1 seconds of elapsed time. During the process 12 refactorizations were performed in 149.7 seconds. This means that refactorizations amounts 26.49% to the overall computational time. We used the same problem for the test with the inversion frequency 50.

The effect of inversion frequency on solution time:

Inversion frequency (iterations)	Solution time (secs)	Iterations	Time per iteration (secs)
20	565.1	221	2.55
50	544	224	2.43

Results show that higher inversion frequency does not effect much the overall solution time. This can be explained with relatively slow execution of the product form variant of revised simplex method. With the use of the Forrest-Tomlin method the performance could be slightly improved (Ashford, Daniel, 1988).

References

1. Ashford R.W., R.C. Daniel: " A note on evaluating LP software for personal computers", European Journal of Operations Research, 35(1988), pp. 160-164.

2. Benichou M., J.M. Gauthier, G. Hentges, G. Ribiere: "The efficient solution of large-scale linear programming problems - some algorithmic techniques and computational results", Mathematical Programming, 13(1977), pp. 280-322.

3. Chvatal V.: Linear Programming, New York - San Francisco, W.H. Freeman and Company 1983.

4. Forrest J.J.H., Tomlin J.A.: "Updated triangular factors of the basis to maintain sparsity in the product form simplex method", Mathematical Programming, 2(1972), pp. 263-278.

5. Greenberg H.J.: "A Tutorial on Matricial Packing", Design and Implementation of Optimization software, Urbino (Italy), (Ed. Greenberg H.J.), Alphen aan den Rijn (Netherlands), Sijthoff and Nordhoff 1978, pp. 109-142.

6. Helgason R.V., Kennington J.L.: "Spike swapping in basis reinversion", Naval Research Logistics Quarterly, 27(1980), pp. 697-701.

7. Helgason R.V., Kennington J.L.: "A note on splitting the bump in an elimination factorization", Naval Research Logistics Quarterly, 29(1982), pp. 169-178.

8. Hellerman E., Rarick D.: "Reinversion with the preassigned pivot procedure", Mathematical Programming, 1(1971), pp. 195-216.

9. Rice J.R.: Numerical Methods, Software, and Analysis, New York, McGraw-Hill 1985.

10. Saunders M.A.: "A fast, stable implementation of the simplex method using Bartels-Golub updating", Sparse Matrix Computations, (Eds. Bunch J.R., Rose D.J.), New York, Academic Press 1976, pp. 213-226.

11. Tomlin J.A.: "On scaling linear programming problems", Mathematical Programming Study, 4(1975), pp. 146-166.

12. Barle J., Grad J.: "PC-LIP: A Microcomputer Linear Programming Package", (program description), Ljubljana, 1987.