Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 561–568
DOI: 10.15439/2014F315
ACSIS, Vol. 2

The WZ factorization in MATLAB

Beata Bylina, Jarosław Bylina
Marie Curie-Skłodowska University,
Institute of Mathematics,
Pl. M. Curie-Skłodowskiej 5,
20-031 Lublin, Poland
Email: {beata.bylina,jaroslaw.bylina}@umcs.pl

***Abstract*—In the paper the authors present the WZ factorization in MATLAB. MATLAB is an environment for matrix computations, therefore in the paper there are presented both the sequential WZ factorization and a block-wise version of the WZ factorization (called here VWZ). Both the algorithms were implemented and their performance was investigated. For random dense square matrices with the dominant diagonal we report the execution time of the WZ factorization in MATLAB and we investigate the accuracy of such solutions. Additionally, the results (time and accuracy) for our WZ implementations were compared to the similar ones based on the LU factorization.**

Keywords: linear system, WZ factorization, LU factorization, matrix factorization, matrix computations

I. Introduction

IN THE international academic circles MATLAB is accepted as a reliable and convinient software for numerical computations. Particularly, it is used for linear algebra computations. Nowadays, there are a lot of papers devoted to the use of MATLAB in mathematics (linear systems [7], least-squares problems [9]; function approximation [12]; eigenvalues [2], [11] — and many others). In this paper we use MATLAB to solve linear systems.

Solution of linear systems of the form:

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \qquad \text{where} \quad \mathbf{A} \in \mathbb{R}^{n\times n}, \quad \mathbf{b} \in \mathbb{R}^{n}, \qquad (1)$$

is an important and common problem in engineering and scientific computations. One of the direct methods of solving a dense linear system (1) is to factorize the matrix $\mathbf{A}$ into some simpler matrices — it is its decomposition into factor matrices (that is, factorization) of a simpler structure — and then solving simpler linear systems. The most known factorization is the LU factorization. MATLAB provides many ways to solve linear systems, one of them is based on the LU factorization: `[L,U]=lu(A)`. This method is powerful and simple to use.

In [7] an object-oriented method is presented, which is is a meta-algorithm that selects the best factorization method for a particular matrix, whether sparse or dense — allowing the reuse of its factorization for subsequent systems.

In this work we study another form of the factorization, namely the WZ factorization and investigate both the accuracy of the computations and their time. In [4], [5] we showed that there are matrices for which applying the incomplete WZ preconditioning gives better results than the incomplete LU factorization.

The aim of the paper is to analyze the potential of implementations of the WZ factorization in a high-level language (as it is the case of MATLAB). We implement the WZ factorization and compare its performance to a MATLAB function implementing the LU factorization, namely: `[L,U]=lu(A)` — and to the authors' own MATLAB implementation of the LU factorization.

The content of the paper is following. In Section II we describe the idea of the WZ factorization [8], [13] and the way the matrix $\mathbf{A}$ is factorized to a product of matrices $\mathbf{W}$ and $\mathbf{Z}$ — such a factorization exists for every nonsingular matrix (with pivoting) what was shown in [8]. Section III provides information about some modifications of the original WZ algorithm — in a way to decrease the number of loops and to make as much as possible computations in blocks — and this will allow us to use MATLAB efficiently. In Section IV we present the results of our experiments. We analyzed the time of WZ factorization. We study the influence of the size of the matrix on the achieved numerical accuracy. We compare the WZ factorization to the LU factorization. Section V is a summary of our experiments.

II. WZ Factorization (WZ)

Here we describe shortly the WZ factorization usage to solve (1). The WZ factorization is described in [8], [10]. Let us assume that the $\mathbf{A}$ is a square nonsingular matrix of an even size (it is somewhat easier to obtain formulas for even sizes than for odd ones). We are to find matrices $\mathbf{W}$ and $\mathbf{Z}$ that fulfill $\mathbf{WZ} = \mathbf{A}$ and the matrices $\mathbf{W}$ and $\mathbf{Z}$ consist of the rows $\mathbf{w}_i^T$ and $\mathbf{z}_i^T$ shown in Figure 1, respectively.

After the factorization we can solve two linear systems:

$$\mathbf{W}\mathbf{y} = \mathbf{b},$$

$$\mathbf{Z}\mathbf{x} = \mathbf{y}$$

(where $\mathbf{c}$ is an auxiliary intermediate vector) instead of one (1).

Figure 2 shows an example of a matrix nad its WZ factors.

In this paper we are interested only in obtaining the matrices $\mathbf{Z}$ and $\mathbf{W}$. The first part of the algorithm consists in setting

This work was partially supported within the project N N516 479640 of the Ministry of Science and Higher Education of the Polish Republic (MNiSW) "Modele dynamiki transmisji, sterowania, zatłoczeniem i jakością usług w Internecie".

$$
\begin{aligned}
\mathbf{w}_1^T &= (1, \underbrace{0, \ldots, 0}_{n-1}) \\
\mathbf{w}_i^T &= (w_{i1}, \ldots, w_{i,i-1}, 1, \underbrace{0, \ldots, 0}_{n-2i+1}, w_{i,n-i+2}, \ldots, w_{in}) && \text{for} \quad i = 2, \ldots, \tfrac{n}{2}, \\
\mathbf{w}_i^T &= (w_{i1}, \ldots, w_{i,n-i}, \underbrace{0, \ldots, 0}_{2i-n-1}, 1, w_{i,i+1}, \ldots, w_{in}) && \text{for} \quad i = \tfrac{n}{2}+1, \ldots, n-1, \\
\mathbf{w}_n^T &= (\underbrace{0, \ldots, 0}_{n-1}, 1) \\
\mathbf{z}_i^T &= (\underbrace{0, \ldots, 0}_{i-1}, z_{ii}, \ldots, z_{i,n-i+1}, 0, \ldots, 0) && \text{for} \quad i = 1, \ldots, \tfrac{n}{2}, \\
\mathbf{z}_i^T &= (\underbrace{0, \ldots, 0}_{n-i}, z_{i,n-i+1}, \ldots, z_{ii}, 0, \ldots, 0) && \text{for} \quad i = \tfrac{n}{2}+1, \ldots, n.
\end{aligned}
$$

Fig. 1. Rows of the matrices $\mathbf{W}$ and $\mathbf{Z}$

$$
\mathbf{A} = \begin{bmatrix}
2 & 1 & 3 & -6 & 3 & 3 \\
10 & 6 & 9 & -13 & 10 & 14 \\
12 & 13 & 12 & -13 & 19 & 17 \\
8 & 10 & 11 & -4 & 12 & 11 \\
12 & 8 & 13 & -20 & 14 & 17 \\
3 & 1 & 1 & -1 & 1 & 4
\end{bmatrix}
$$

$$
\mathbf{W} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 0 & 2 \\
3 & 1 & 1 & 0 & 2 & 2 \\
1 & 2 & 0 & 1 & 1 & 2 \\
3 & 0 & 0 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\quad
\mathbf{Z} = \begin{bmatrix}
2 & 1 & 3 & -6 & 3 & 3 \\
0 & 2 & 1 & 1 & 2 & 0 \\
0 & 0 & -4 & 6 & 0 & 0 \\
0 & 0 & 2 & 2 & 0 & 0 \\
0 & 3 & 2 & 0 & 3 & 0 \\
3 & 1 & 1 & -1 & 1 & 4
\end{bmatrix}
$$

Fig. 2. A matrix $\mathbf{A}$ and its factors $\mathbf{W}$ and $\mathbf{Z}$

succesive parts of columns of the matrix $\mathbf{A}$ to zeros. In the first step we do that with the elements in columns 1st and nth — from the 2nd row to the $n-1$st row. Next we update the matrix $\mathbf{A}$.

More formally we can describe the first step of the algoritm the following way.

1) For every $i = 2, \ldots, n-1$ we compute w_{i1} and w_{in} from the system:

$$
\begin{cases}
a_{11}w_{i1} + a_{n1}w_{in} = -a_{i1} \\
a_{1n}w_{i1} + a_{nn}w_{in} = -a_{in}
\end{cases}
$$

and we put them in a matrix of the form:

$$
\mathbf{W}^{(1)} = \begin{bmatrix}
1 & 0 & \cdots & 0 & 0 \\
w_{21} & 1 & \ddots & \vdots & w_{2n} \\
\vdots & 0 & \ddots & 0 & \vdots \\
w_{n-1,1} & \vdots & \ddots & 1 & w_{n-1,n} \\
0 & 0 & \cdots & 0 & 1
\end{bmatrix}.
$$

2) We compute:

$$
\mathbf{A}^{(1)} = \mathbf{W}^{(1)}\mathbf{A}.
$$

After the first step we get a matrix of the form:

$$
\mathbf{A}^{(1)} = \begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1,n-1} & a_{1n} \\
0 & a_{22}^{(1)} & \cdots & a_{2,n-1}^{(1)} & 0 \\
\vdots & \vdots & & \vdots & \vdots \\
0 & a_{n-1,2}^{(1)} & \cdots & a_{n-1,n-1}^{(1)} & 0 \\
a_{n1} & a_{n2} & \cdots & a_{n,n-1} & a_{nn}
\end{bmatrix},
$$

where (for $i, j = 2, \ldots, n-1$):

$$
a_{ij}^{(1)} = a_{ij} + w_{i1}a_{1j} + w_{in}a_{nj}.
$$

Then, we proceed analogously — but for the inner square matrices — $\mathbf{A}^{(1)}$ of size $n-2$ and so on.

So, the whole algorithm is following.

For $k = 1, 2, \ldots, \frac{n}{2} - 1$:

1) For every $i = k+1, \ldots, n-k$ we compute w_{ik} and $w_{i,n-k+1}$ from the system:

$$\begin{cases} a_{kk}^{(k-1)} w_{ik} & + & a_{n-k+1,k}^{(k-1)} w_{i,n-k+1} \\ & = & -a_{ik}^{(k-1)} \\ a_{k,n-k+1}^{(k-1)} w_{ik} & + & a_{n-k+1,n-k+1}^{(k-1)} w_{i,n-k+1} \\ & = & -a_{i,n-k+1}^{(k-1)} \end{cases}$$

and we put them in a matrix of the form shown in Figure 3.

2) We compute:

$$\mathbf{A}^{(k)} = \mathbf{W}^{(k)} \mathbf{A}^{(k-1)}$$

(where obviously $\mathbf{A}^{(0)} = \mathbf{A}$).

After $\frac{n}{2} - 1$ such steps we get the matrix

$$\mathbf{Z} = \mathbf{A}^{(\frac{n}{2}-1)}$$

.

Moreover, we know that:

$$\mathbf{W}^{(\frac{n}{2}-1)} \cdot \ldots \cdot \mathbf{W}^{(1)} \cdot \mathbf{A} = \mathbf{Z}$$

so we get:

$$\mathbf{A} = \left(\mathbf{W}^{(1)}\right)^{-1} \cdot \ldots \cdot \left(\mathbf{W}^{(\frac{n}{2}-1)}\right)^{-1} \cdot \mathbf{Z} = \mathbf{W}\mathbf{Z}$$

Figure 4 shows the same algorithm implemented in MATLAB.

III. Vector WZ Factorization (VWZ)

Now we describe a matrix-vector algorithm for the WZ factorization of the matrix $\mathbf{A}$ which is originally presented in [6].

This is a sequential algorithm where we grouped and ordered the scalar operations anew, into matrix-vector operations. We are showing the algorithm without pivoting, working only for matrices for which such a WZ factorization is executable (for example, for matrices with a dominant diagonal.

Let us write the matrices $\mathbf{A}$, $\mathbf{W}$, $\mathbf{Z}$ as block matrices. We can get equations presented in Figure 5.

In Figure 5, $\widehat{\mathbf{W}}$ and $\widehat{\mathbf{Z}}$ are square matrices of the same structure as the matrices $\mathbf{W}$ and $\mathbf{Z}$, respectively; $\widehat{\mathbf{A}}$ is a full square matrix; $\widehat{\mathbf{W}}$, $\widehat{\mathbf{Z}}$ and $\widehat{\mathbf{A}}$ are of the size 2 less than the size of $\mathbf{W}$, $\mathbf{Z}$ and $\mathbf{A}$; vectors $\mathbf{a}_{1*}^T$, $\mathbf{a}_{n*}^T$, $\mathbf{z}_{1*}^T$, $\mathbf{z}_{n*}^T$ are row vectors; vectors $\mathbf{a}_{*1}$, $\mathbf{a}_{*n}$, $\mathbf{w}_{*1}$, $\mathbf{w}_{*n}$ are column vectors.

From the comparison of the corresponding elements in Figure 5 we get:

$$\begin{array}{ll} a_{11} = z_{11}; & a_{1n} = z_{1n}; \\ a_{n1} = z_{n1}; & a_{nn} = z_{nn} \end{array}$$

$$\mathbf{a}_{1*}^T = \mathbf{z}_{1*}^T; \quad \mathbf{a}_{n*}^T = \mathbf{z}_{n*}^T$$

$$\begin{cases} \mathbf{a}_{*1} & = & \mathbf{w}_{*1} z_{11} & + & \mathbf{w}_{*n} z_{n1} \\ \mathbf{a}_{*n} & = & \mathbf{w}_{*1} z_{1n} & + & \mathbf{w}_{*n} z_{nn} \end{cases} \tag{2}$$

$$\widehat{\mathbf{A}} = \mathbf{w}_{*1}\mathbf{z}_{1*}^T + \widehat{\mathbf{W}}\widehat{\mathbf{Z}} + \mathbf{w}_{*n}\mathbf{z}_{n*}^T$$

From (2) we can describe our algorithm for finding $\mathbf{W}$ and $\mathbf{Z}$ as following:

1) let the first row of the matrix $\mathbf{Z}$ be the first row of the matrix $\mathbf{A}$;
2) let the last row of the matrix $\mathbf{Z}$ be the last row of the matrix $\mathbf{A}$;
3) compute the vectors $\mathbf{w}_{*1}$ and $\mathbf{w}_{*n}$ from:

$$\begin{aligned} \mathbf{w}_{*1} &= \alpha \mathbf{a}_{*n} - \beta \mathbf{a}_{*1}, \\ \mathbf{w}_{*n} &= \gamma \mathbf{a}_{*1} - \delta \mathbf{a}_{*n}, \end{aligned}$$

where

$$\begin{aligned} \alpha &= \frac{z_{n1}}{z_{1n}z_{n1} - z_{11}z_{nn}}, \\ \beta &= \frac{z_{nn}}{z_{1n}z_{n1} - z_{11}z_{nn}}, \\ \gamma &= \frac{z_{1n}}{z_{1n}z_{n1} - z_{11}z_{nn}}, \\ \delta &= \frac{z_{11}}{z_{1n}z_{n1} - z_{11}z_{nn}}; \end{aligned}$$

4) update the inner part of the matrix $\mathbf{A}$ (the matrix without its first and last row and column):

$$\widehat{\mathbf{A}}^{\text{new}} = \widehat{\mathbf{W}}\widehat{\mathbf{Z}} = \widehat{\mathbf{A}} - \mathbf{w}_{*1}\mathbf{z}_{1*}^T - \mathbf{w}_{*n}\mathbf{z}_{n*}^T;$$

5) if the size of the matrix $\widehat{\mathbf{A}}^{\text{new}}$ is 3 or more, then start over from 1., but with $\mathbf{A} = \widehat{\mathbf{A}}^{\text{new}}$, $\mathbf{W} = \widehat{\mathbf{W}}$ and $\mathbf{Z} = \widehat{\mathbf{Z}}$ (so all three matrices become smaller and smaller and the algorithm comes eventually to the end).

Figure 6 shows this algorithm implemented in MATLAB.

IV. Numerical Experiments

In this section we tested the time and the absolute accuracy of the WZ factorization. Our intention was to investigate the WZ factorization and compare our implementation (done in MATLAB, which is a high-level language) of the WZ factorization with the LU factorization.

The input matrices are generated (by the authors). They are random matrices with a dominant diagonal of even sizes (500, 1000, 1500 and so on, up to 3000 or 4000).

The MATLAB implementation was compiled and tested under MS Windows on workstations with an AMD processor and an Intel processor. Table I shows details of specification of the hardware used in the experiment. To measure the performance time standard MATLAB functions were used — `tic` and `toc`. We measured the difference between $\mathbf{A}$ and $\mathbf{WZ}$ by absolute error $||\mathbf{A} - \mathbf{WZ}||_2$ — to compute the norm there was used a standard MTLAB function: `norm`. The implementations were tested in MATLAB R2008 and MATLAB R2010.

Figures 8, 9, 10 and 11 show the performance time (in seconds) of the WZ factorization on the AMD processor and the Intel processor, implemented in MATLAB R2008 and MATLAB R2010. Additionally, we compared the WZ

$$\mathbf{W}^{(k)} = \begin{bmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & w_{k+1,k} & \ddots & & w_{k+1,n-k+1} & & \\ & & \vdots & & \ddots & \vdots & & \\ & & w_{n-k,k} & & & w_{n-k,n-k+1} & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{bmatrix}$$

Fig. 3. The form of the matrix $\mathbf{W}^{(k)}$

```
% steps of elimination — from A to Z
for k = 0 : n/2-2
  k2 = n-k-1;
  det = A(k,k)*A(k2,k2)-A(k2,k)*A(k,k2);
% finding elements of W
  for i = k+1 : k2-1
    w(i,k) = (A(k2,k)*A(i,k2)-A(k2,k2)*A(i,k))/det;
    w(i,k2) = (A(k,k2)*A(i,k)-A(k,k)*A(i,k2))/det;
% updating A
    for j = k+1 : k2-1
      A(i,j) = A(i,j)+wk1*A(k,j)+wk2*A(k2,j);
```

Fig. 4. The sequential implementation of the WZ factorization for solving linear systems

.

$$\mathbf{A} = \begin{bmatrix} a_{11} & \mathbf{a}_{1*}^T & a_{1n} \\ \mathbf{a}_{*1} & \widehat{\mathbf{A}} & \mathbf{a}_{*n} \\ a_{n1} & \mathbf{a}_{n*}^T & a_{nn} \end{bmatrix} = \mathbf{WZ} = \begin{bmatrix} 1 & 0 & 0 \\ \mathbf{w}_{*1} & \widehat{\mathbf{W}} & \mathbf{w}_{*n} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} z_{11} & \mathbf{z}_{1*}^T & z_{1n} \\ 0 & \widehat{\mathbf{Z}} & 0 \\ z_{n1} & \mathbf{z}_{n*}^T & z_{nn} \end{bmatrix} =$$

$$= \begin{bmatrix} z_{11} & \mathbf{z}_{1*}^T & z_{1n} \\ \mathbf{w}_{*1}z_{11} + \mathbf{w}_{*n}z_{n1} & \mathbf{w}_{*1}\mathbf{z}_{1*}^T + \widehat{\mathbf{W}}\widehat{\mathbf{Z}} + \mathbf{w}_{*n}\mathbf{z}_{n*}^T & \mathbf{w}_{*1}z_{1n} + \mathbf{w}_{*n}z_{nn} \\ z_{n1} & \mathbf{z}_{n*}^T & z_{nn} \end{bmatrix}$$

Fig. 5. The WZ factorization written as blocks of vector

```
% steps of elimination — from A to Z
for k = 0 : n/2-2
  k2 = n-k-1;
  det =A(k,k)*A(k2,k2)-A(k2,k)*A(k,k2);
% finding elements of W
    W(k+1:k2-1,k) = (A(k2,k)*A(k+1:k2-1,k2)-A(k2,k2)*A(k+1:k2-1,k))/det;
    W(k+1:k2-1,k2) = (A(k,k2)*A(k+1:k2-1,k)-A(k,k)*A(k+1:k2-1,k2))/det;
% updating A
    A(k+1:k2-1,k+1:k2-1) = A(k+1:k2-1,k+1:k2-1)+ W(k+1:k2-1,k)*A(k,k+1:k2-1)
                                                +W(k+1:k2-1,k2)*A(k2,k+1:k2-1);
```

Fig. 6. The vector implementation of the WZ factorization for solving linear systems

```
% steps of elimination — from A to U
for k = 0 : n-2
% finding elements of L
  for i = k+1 : n-1
    l(i,k) =-a(i,k)/a(k,k);
% updating A
    for j = k+1 : n
      A(i,j) = A(i,j) + l(i,k)*A(k,j);
```

Fig. 7. The sequential implementation of the LU factorization for solving linear systems

TABLE I
HARDWARE USED IN EXPERIMENTS

#	CPU	Memory
1	AMD FX-8120 3.1 GHz	16 GB
2	Intel Core 2 Duo 2.53 GHz	4 GB

factorization (both in its sequential — Figure 4 — and vector — Figure 6 — versions) with a standard MATLAB LU factorization function, namely `lu`, which uses the subroutine DGETRF from LAPACK [1], and also with the simple LU implementation (shown in Figure 7).

Results show that the vector WZ factorization (VWZ) is much faster than the sequential WZ factorization in both tested MATLAB versions and on both architectures.

However, on the older processor (Intel Core is here the case) the sequentials algorithms perform better than on the newer (AMD) — and the block algorithms (VWA and the standard MATLAB function `lu`) perform better on the newer one. It is caused by the differences in architectues — newer ones prefer block algorithms because of their stronger inner parallelism.

Tables II, III, IV and V illustrate the accuracy (given as the norms $||\mathbf{A} - \mathbf{WZ}||_2$ and $||\mathbf{A} - \mathbf{LU}||_2$) of the WZ and LU factorizations in MATLAB. The first column shows the norm for the sequential WZ factorization (from Figure 4); the second — the vector WZ factorization (VWZ, from Figure 6); the third presents the norm for the sequential LU factorization (from Figure 7); the fourth — the norm for the standard MATLAB function `lu`.

Based on the results, we can state that different implementations give quite similar accuracies. However, the sizes of the matrix influences the accuracy (it worsens when the size grows).

Tables VI, VII, VIII, IX, illustrate the speedup for the VWZ and LU factorizations in MATLAB (both R2008 and R2010) relative to the sequential WZ factorization. The first column shows the speedup of VWZ, the second — the speedup of the LU factorization and the third — the speed of the standard MATLAB function `lu` — all relative to the sequential WZ factorization.

Based on these results, we can conclude that various implementations of the WZ factorization give different performance. Namely, VWZ is even about 4 times faster then the sequential WZ (on the AMD processor; on the Intel processor the speedup is only about 2). The LU factorization implemented by the authors is the slowest of all the tested implementations. However, the standard MATLAB function `lu` is the fastest — this function implements a block LU factorization, which makes the processor architecture is better utilized.

V. CONCLUSION

In this paper we did some performance analysis of a MATLAB implementations of the WZ factorization. We examined a sequential implementation of the WZ factorization. We also implemented in MATLAB a vector version of the WZ factorization (VWZ) — to avoid loops. We compared these implementations with two versions of the LU factorization — our MATLAB implementation and a standard MATLAB function `[L,U]=lu(A)`.

From the results we can conclude that the reduction of the number of nested loops in the original WZ factorization increased the speed even four times. The sequential WZ factorization is faster than the sequential LU factorization. Of course, the fastest of the implementation is the built-in MATLAB function `lu` — which utilizes LAPACK block factorization [1].

The implementation and the architecture had no impact on the accuracy of the factorization — the accuracy depended only on the size of the matrix what is quite self-evident.

The version of MATLAB has no significant influence on neither the performance time nor the speedup — only the architecture and the size of the matrix count.

VI. FUTURE WORK

To accelerate the WZ factorization, it would be desirable to build a block algorithm for the WZ factorization and to utilize parallelism — especially for the machines with many processing units.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, *LAPACK Users' Guide (Third ed.)*, SIAM, Philadelphia 1999.

[2] T. Betcke, N. J. Higham, V. Mehrmann,Ch. Schröder, F. Tisseur, NLEVP: A Collection of Nonlinear Eigenvalue Problems, *ACM Trans. Math. Softw.*, Volume 39 Issue 2, February 2013, Article No. 7.

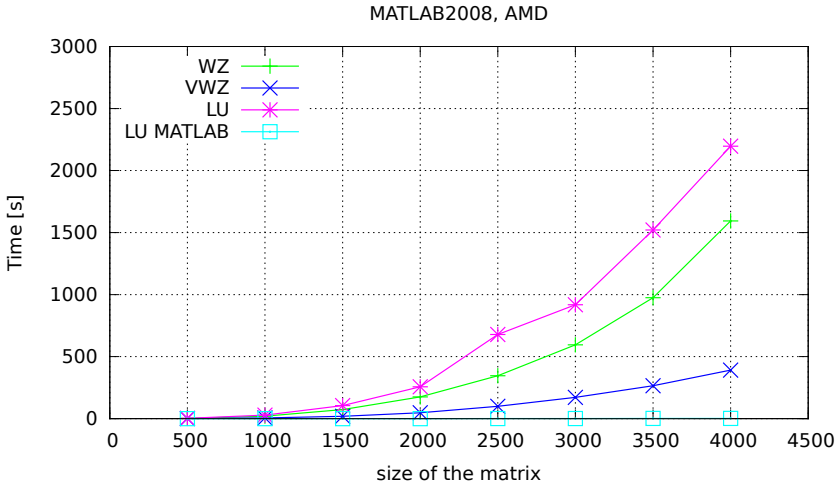

Fig. 8. The WZ factorization performance time (in seconds) on the AMD processor, in MATLAB R2008

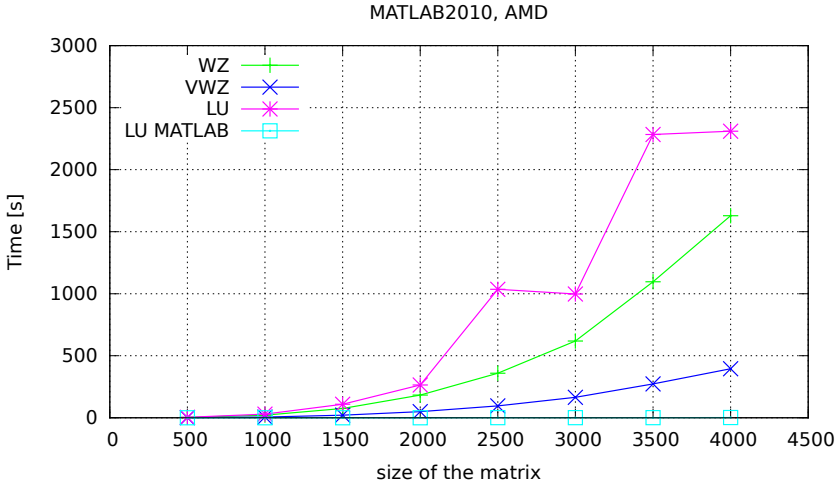

Fig. 9. The WZ factorization performance time (in seconds) on the AMD processor, in MATLAB R2010

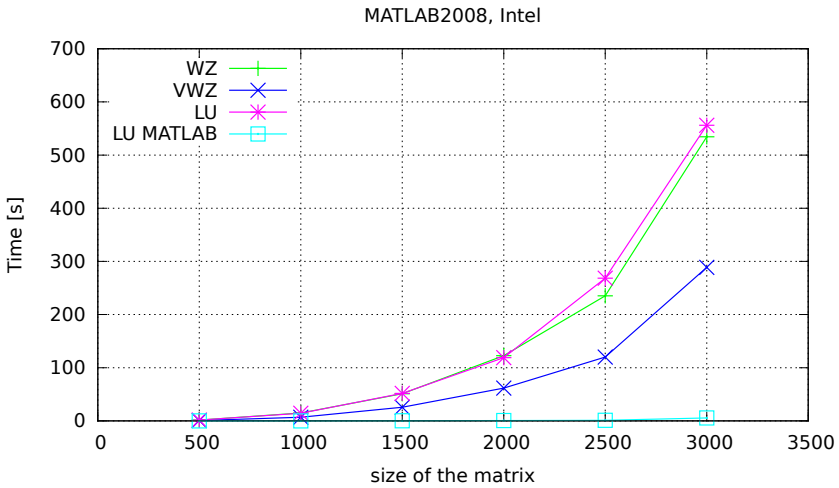

Fig. 10. The WZ factorization performance time (in seconds) on the Intel processor, in MATLAB R2008

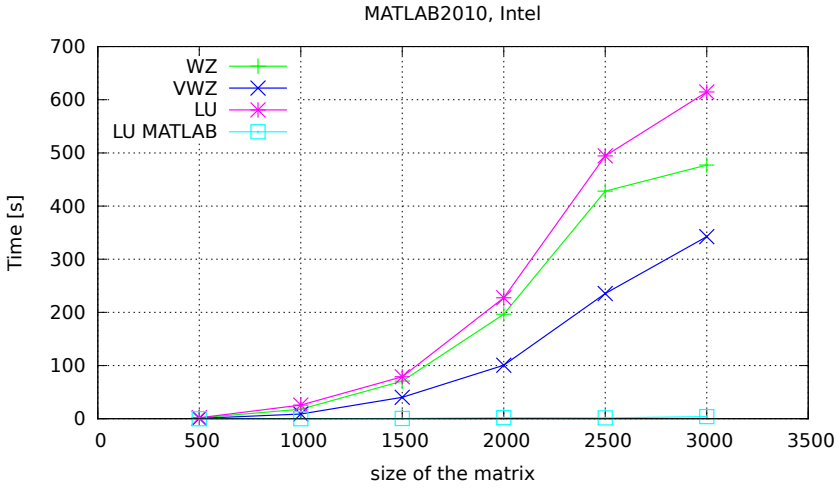

Fig. 11. The WZ factorization performance time (in seconds) on the Intel processor, in MATLAB R2010

TABLE II
THE NORMS FOR THE WZ AND LU FACTORIZATIONS IN MATLAB R2008 ON THE AMD PROCESSOR

matrix size	WZ	VWZ	LU	`lu`
500	$1.08 \cdot 10^{-12}$	$4.00 \cdot 10^{-13}$	$8.53 \cdot 10^{-13}$	$5.68 \cdot 10^{-13}$
1000	$2.84 \cdot 10^{-12}$	$1.02 \cdot 10^{-12}$	$2.61 \cdot 10^{-12}$	$7.96 \cdot 10^{-13}$
1500	$8.18 \cdot 10^{-12}$	$3.86 \cdot 10^{-12}$	$7.50 \cdot 10^{-12}$	$5.00 \cdot 10^{-12}$
2000	$7.96 \cdot 10^{-12}$	$2.73 \cdot 10^{-12}$	$7.73 \cdot 10^{-12}$	$2.95 \cdot 10^{-12}$
2500	$2.50 \cdot 10^{-11}$	$5.91 \cdot 10^{-12}$	$2.09 \cdot 10^{-11}$	$3.18 \cdot 10^{-12}$
3000	$2.51 \cdot 10^{-11}$	$7.28 \cdot 10^{-12}$	$2.09 \cdot 10^{-11}$	$2.09 \cdot 10^{-11}$
3500	$2.13 \cdot 10^{-11}$	$7.73 \cdot 10^{-12}$	$2.50 \cdot 10^{-11}$	$5.91 \cdot 10^{-12}$
4000	$2.54 \cdot 10^{-11}$	$9.09 \cdot 10^{-12}$	$2.64 \cdot 10^{-11}$	$4.55 \cdot 10^{-12}$

TABLE III
THE NORMS FOR THE WZ AND LU FACTORIZATIONS IN MATLAB R2010 ON THE AMD PROCESSOR

matrix size	WZ	VWZ	LU	`lu`
500	$9.08 \cdot 10^{-13}$	$4.00 \cdot 10^{-13}$	$8.53 \cdot 10^{-13}$	$5.68 \cdot 10^{-13}$
1000	$2.84 \cdot 10^{-12}$	$1.02 \cdot 10^{-12}$	$2.61 \cdot 10^{-12}$	$7.96 \cdot 10^{-13}$
1500	$7.27 \cdot 10^{-12}$	$3.86 \cdot 10^{-12}$	$7.50 \cdot 10^{-12}$	$5.00 \cdot 10^{-12}$
2000	$7.96 \cdot 10^{-12}$	$2.73 \cdot 10^{-12}$	$7.73 \cdot 10^{-12}$	$2.95 \cdot 10^{-12}$
2500	$2.09 \cdot 10^{-11}$	$5.91 \cdot 10^{-12}$	$2.09 \cdot 10^{-11}$	$3.18 \cdot 10^{-12}$
3000	$2.32 \cdot 10^{-11}$	$7.28 \cdot 10^{-12}$	$2.09 \cdot 10^{-11}$	$2.09 \cdot 10^{-11}$
3500	$2.58 \cdot 10^{-11}$	$7.30 \cdot 10^{-12}$	$2.50 \cdot 10^{-11}$	$5.91 \cdot 10^{-12}$
4000	$2.45 \cdot 10^{-11}$	$9.09 \cdot 10^{-12}$	$2.64 \cdot 10^{-11}$	$4.55 \cdot 10^{-12}$

TABLE IV
THE NORMS FOR THE WZ AND LU FACTORIZATIONS IN MATLAB R2008 ON THE INTEL PROCESSOR

matrix size	WZ	VWZ	LU	`lu`
500	$9.09 \cdot 10^{-13}$	$9.09 \cdot 10^{-13}$	$8.52 \cdot 10^{-13}$	$5.68 \cdot 10^{-13}$
1000	$2.84 \cdot 10^{-12}$	$2.84 \cdot 10^{-12}$	$2.61 \cdot 10^{-12}$	$6.82 \cdot 10^{-13}$
1500	$7.27 \cdot 10^{-12}$	$7.27 \cdot 10^{-12}$	$7.73 \cdot 10^{-12}$	$5.00 \cdot 10^{-12}$
2000	$8.40 \cdot 10^{-12}$	$8.40 \cdot 10^{-12}$	$7.95 \cdot 10^{-12}$	$1.82 \cdot 10^{-12}$
2500	$2.09 \cdot 10^{-11}$	$2.09 \cdot 10^{-12}$	$2.09 \cdot 10^{-11}$	$1.36 \cdot 10^{-12}$
3000	$2.27 \cdot 10^{-11}$	$2.27 \cdot 10^{-12}$	$2.09 \cdot 10^{-11}$	$3.63 \cdot 10^{-11}$

TABLE V
THE NORMS FOR THE WZ AND LU FACTORIZATIONS IN MATLAB R2010 ON THE INTEL PROCESSOR

matrix size	WZ	VWZ	LU	`lu`
500	$6.83 \cdot 10^{-13}$	$6.83 \cdot 10^{-13}$	$1.19 \cdot 10^{-12}$	$3.98 \cdot 10^{-13}$
1000	$2.39 \cdot 10^{-12}$	$2.39 \cdot 10^{-12}$	$2.50 \cdot 10^{-12}$	$7.96 \cdot 10^{-13}$
1500	$7.96 \cdot 10^{-12}$	$7.96 \cdot 10^{-12}$	$8.18 \cdot 10^{-12}$	$1.59 \cdot 10^{-12}$
2000	$9.78 \cdot 10^{-12}$	$9.78 \cdot 10^{-12}$	$1.00 \cdot 10^{-11}$	$1.82 \cdot 10^{-12}$
2500	$2.18 \cdot 10^{-11}$	$2.18 \cdot 10^{-11}$	$2.36 \cdot 10^{-11}$	$3.64 \cdot 10^{-12}$
3000	$2.36 \cdot 10^{-11}$	$2.36 \cdot 10^{-11}$	$2.41 \cdot 10^{-11}$	$4.55 \cdot 10^{-12}$

TABLE VI
THE SPEEDUP OF VWZ, OF THE LU FACTORIZATION AND OF THE STANDARD MATLAB FUNCTION `LU` — RELATIVE TO THE SEQUENTIAL WZ FACTORIZATION (MATLAB R2008 ON THE AMD PROCESSOR)

matrix size	VWZ	LU	`lu`
500	3.63	0.73	225.00
1000	3.73	0.72	288.57
1500	3.78	0.68	426.41
2000	3.68	0.68	486.44
2500	3.47	0.51	628.85
3000	3.47	0.64	646.47
3500	3.63	0.64	601.82
4000	4.07	0.72	870.95

TABLE VII
THE SPEEDUP OF VWZ, OF THE LU FACTORIZATION AND OF THE STANDARD MATLAB FUNCTION `LU` — RELATIVE TO THE SEQUENTIAL WZ FACTORIZATION (MATLAB R2010 ON THE AMD PROCESSOR)

matrix size	VWZ	LU	`lu`
500	4.09	0.70	237.00
1000	3.69	0.71	519.25
1500	3.63	0.68	574.92
2000	3.72	0.69	536.50
2500	3.75	0.35	641.50
3000	3.76	0.62	703.00
3500	4.02	0.48	856.25
4000	4.12	0.71	920.52

TABLE VIII
THE SPEEDUP OF VWZ, OF THE LU FACTORIZATION AND OF THE STANDARD MATLAB FUNCTION `LU` — RELATIVE TO THE SEQUENTIAL WZ FACTORIZATION (MATLAB R2008 ON THE INTEL PROCESSOR)

matrix size	VWZ	LU	`lu`
500	2.56	0.99	159.00
1000	2.14	1.03	165.44
1500	2.00	0.99	189.30
2000	1.99	1.03	204.50
2500	1.96	0.88	206.35
3000	1.85	0.96	96.11

TABLE IX
THE SPEEDUP OF VWZ, OF THE LU FACTORIZATION AND OF THE STANDARD MATLAB FUNCTION `LU` — RELATIVE TO THE SEQUENTIAL WZ FACTORIZATION (MATLAB R2010 ON THE INTEL PROCESSOR)

matrix size	VWZ	LU	`lu`
500	2.01	0.92	95.01
1000	2.01	0.70	175.04
1500	1.75	0.90	144.39
2000	1.95	0.86	101.59
2500	1.82	0.86	265.22
3000	1.39	0.78	119.58

[3] B. Bylina, J. Bylina: Analysis and Comparison of Reordering for Two Factorization Methods (LU and WZ) for Sparse Matrices, *Lecture Notes in Computer Science* **5101**, Springer-Verlag Berlin Heidelberg 2008, pp. 983–992.

[4] B. Bylina, J. Bylina: Incomplete WZ Factorization as an Alternative Method of Preconditioning for Solving Markov Chains, *Lecture Notes in Computer Science* **4967**, Springer-Verlag Berlin Heidelberg 2008, 99–107.

[5] B. Bylina, J. Bylina: Influence of preconditioning and blocking on accuracy in solving Markovian models, *International Journal of Applied Mathematics and Computer Science* 19 (2) (2009), pp. 207–217.

[6] B. Bylina, J. Bylina: The Vectorized and Parallelized Solving of Markovian Models for Optical Networks, *Lecture Notes in Computer Science* **3037**, Springer-Verlag Berlin Heidelberg 2004, 578–581.

[7] T. A. Davis, Algorithm 930: FACTORIZE: An Object-oriented Linear System Solver for MATLAB, *ACM Trans. Math. Softw.*, Volume 39 Issue 4, July 2013, Article No. 28. pages = 28:1–28:18

[8] S. Chandra Sekhara Rao: Existence and uniqueness of WZ factorization, *Parallel Computing* **23** (1997), pp. 1129–1139.

[9] Choi, T. Sou-Cheng, M. A. Saunders, Algorithm 937: MINRES-QLP for Symmetric and Hermitian Linear Equations and Least-squares Problems, *ACM Trans. Math. Softw.*, Volume 40 Issue 2, February 2014, Article No. 16. pages = 16:1–16:12,

[10] D. J. Evans, M. Hatzopoulos: The parallel solution of linear system, *Int. J. Comp. Math.* **7** (1979), pp. 227–238.

[11] X. Ji, J. Sun, T. Turner, Algorithm 922: A Mixed Finite Element Method for Helmholtz Transmission Eigenvalues, *ACM Trans. Math. Softw.*, Volume 38 Issue 4, August 2012, Article No. 29. pages = 29:1–29:8,

[12] K. Poppe, R. Cools, CHEBINT: A MATLAB/Octave Toolbox for Fast Multivariate Integration and Interpolation Based on Chebyshev Approximations over Hypercubes, *ACM Trans. Math. Softw.*, Volume 40 Issue 1, September 2013, Article No. 2. pages = 2:1–2:13,

[13] P. Yalamov, D. J. Evans: The WZ matrix factorization method, *Parallel Computing* **21** (1995), pp. 1111–1120.