Comments on 'A structure-preserving method for the quaternion LU decomposition in quaternionic quantum theory' by Minghui Wang and Wenhao Ma

Stephen J. Sangwine[a]

[a]*School of Computer Science and Electronic Engineering, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, United Kingdom*

Abstract

Claims were made in an article by Wang and Ma in 2013 that they had devised an algorithm for the quaternion LU decomposition that was significantly faster than the LU decomposition implemented in the Quaternion Toolbox for MATLAB (QTFM). These claims have been tested and found to be unsupported by MATLAB code supplied to the author by Wang and Ma. The author's tests are presented, and test code made available as supplementary material. It is found that not only is the QTFM code faster, but that Wang and Ma's algorithm has run-time that scales with the square of the size of the matrix, whereas the algorithm in QTFM has run time approximately linear in matrix size. These findings are consistent with an inspection of the code.

1. Background

In a paper published in 2013 [1], Wang and Ma described an algorithm for the quaternion LU decomposition that they claimed was faster than the LU decomposition implemented in the Quaternion Toolbox for Matlab [2]. The author of the present comment is one of the authors of QTFM, and the author of the LU decomposition code (which is unremarkable, being based very closely on the classical LU decomposition described by Golub and van Loan [3] and referenced in the source code file `lu.m`).

The author tested a version of Wang and Ma's code copied and pasted from Algorithm 3.1 in the published article (with minimal editing to make it into syntactically correct MATLAB code) and found that it ran slower than the LU decomposition in the QTFM toolbox. Correspondence with the editor then resulted in the author being supplied with a copy of Wang and Ma's code (provided as supplementary material to this comment). This code was tested

Email address: `sjs@essex.ac.uk` (Stephen J. Sangwine)

and the results are presented below. Not only was the QTFM code found to run faster on all except small matrices, but its run time was confirmed to scale linearly with the number of rows in the matrix, whereas Wang and Ma's code has run time proportional to the square of the number of rows in the matrix. Note that Wang and Ma's code does not work on rectangular matrices, unlike the QTFM code. The tests were therefore limited to square matrices, as in Wang and Ma's paper. In the rest of this comment, the shorthand notation WM will be used to refer to Wang and Ma's algorithm/code.

2. Outline of test issues

The code supplied by Wang and Ma is in a MATLAB function with the array size as the input parameter (the array to be decomposed is created inside the function). The QTFM LU function is embedded in the file as a sub-function. It is not known whether this is how Wang and Ma ran the QTFM code, but this is not an important issue as the author has verified that it makes no material difference to the run time whether the QTFM code is embedded in the function file, or run from the QTFM installation directory (which would be the normal way to use it).

Calling the QTFM code with a function call, while using inline code for the WM version, is clearly not a fair comparison. Both sets of code should be inline, or both should be called as functions, or the overhead of the function call and parameter passing should be accounted for. The importance of comparing code using similar calling profiles is underlined by [4] which describes differences encountered in the running time of MATLAB code due to differences in the handling of large array parameters inside or outside a function.

To attempt a fair comparison and to permit testing of the code on matrices of various sizes, the WM code was modified to make it into a function, and called from a script, looping over matrix sizes and saving measured run-times using the MATLAB `tic` and `toc` functions. The modified code is available as supplementary material. The appendix lists the file names and describes each file.

Further issues that were studied and found to have little or no significance were:

- the version of QTFM,
- the version of MATLAB,
- the operating system and platform (Windows PC or MacBook/OS X)

A test was run with version 1.9 of the QTFM library, which was the most recent version at the time Wang and Ma wrote their paper, running on a Windows PC computer. The same code has been run using the latest version of QTFM running on a Mac computer, and no significant difference was found: Wang and Ma's claim to have developed a method that is faster than QTFM is not supported by the test results. The version of MATLAB used on the Windows

PC was R2013a (8.1) whereas on the MacBook it was R2014b (8.4). This made no difference to the results (QTFM was faster in every case, except for small matrix sizes). The 'crossover' point between the WM LU code and the QTFM code is about 30×30 (that is both versions take about the same time on a 30×30 matrix, above this size QTFM is markedly faster). Note that Wang and Ma's claim is based on tests with matrices up to 1500 square (Table 1 in their paper). This comment is based on tests with matrices up to 500×500 in size as this is sufficient to demonstrate the point.

3. Test results in detail

The author tested both sets of code on array sizes from 10 to 500 square, and found that, except at the smallest sizes, the QTFM code is much faster than that of Wang and Ma. At 200 elements square, the WM code takes over 24 seconds to compute the LU decomposition, whereas the QTFM LU runs in just less than a second. This is inconsistent with the results in Table 1 of Wang and Ma's paper, which shows that QTFM ran for about 1 second on a 200 square array, whereas their own code took about half a second. Numerical results from the author's tests are given in Table 1, and the data is plotted in Figures 1 and 2.

The WM code does achieve smaller residual error than QTFM by about a factor of 3, which roughly agrees with Table 1 in their paper, but the errors are small (of the order of 10^{-10} maximum difference in absolute value between elements of the original matrix and the reconstructed version from the L and U matrices).

To check why the WM code runs so slowly, the author ran the MATLAB profiler on it, and found that almost all of their run time is spent on lines 40-43 of the file `structured_lu.m`. This is not surprising, as this code is computing with matrix elements one at a time – it is not vectorised, unlike the QTFM code, which works with whole rows and columns at a time. The WM code in the paper, and the version supplied to the author, lacks proper vectorisation, and largely works one element at a time. This is consistent with the test results.

4. Conclusion

The tests carried out by the author do not support Wang and Ma's claim that their algorithm is faster than QTFM's LU decomposition, except for small matrices with less than 30 rows/columns. Further the tests show that the run-time of the QTFM LU code scales as expected, approximately linearly with the number of rows in the matrix (since the code contains only one loop, iterating over all rows in the matrix), whereas the run-time of the WM code scales approximately as the size of the matrix squared (due to nested loops that process rows and columns respectively).

Table 1: Test results comparing the WM LU code and the QTFM LU code.

n	WM LU		QTFM LU	
	run time (s)	error $\times 10^{11}$	run time (s)	error $\times 10^{11}$
10	0.087	0.007	0.366	0.009
15	0.017	0.015	0.045	0.009
20	0.028	0.018	0.060	0.012
25	0.051	0.047	0.076	0.016
30	0.086	0.043	0.093	0.020
35	0.138	0.059	0.110	0.025
40	0.203	0.053	0.125	0.025
45	0.282	0.088	0.145	0.044
50	0.390	0.089	0.159	0.033
75	1.318	0.201	0.252	0.065
100	3.097	0.214	0.353	0.086
125	5.859	0.355	0.476	0.170
150	10.302	0.471	0.611	0.178
175	16.354	0.590	0.764	0.266
200	24.361	0.836	0.985	0.289
300	82.917	0.962	2.531	0.393
500	378.207	2.149	9.638	0.695

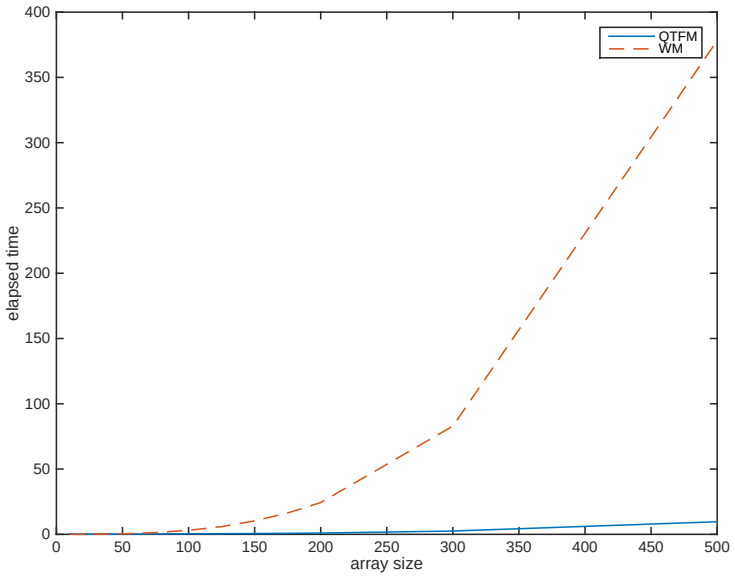

Figure 1: Plot of the elapsed time data from Table 1.

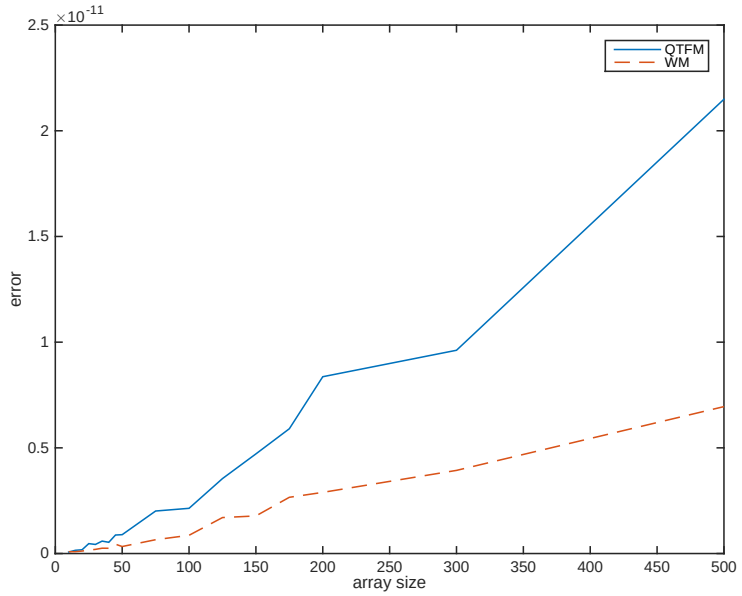

Figure 2: Plot of the error data from Table 1.

Appendix A. List of code files

The following supplementary material was used for the tests described in this comment:

- `wang_qlu7.m` — Source code file supplied by Wang and Ma through the editor.
- `structured_lu.m` — Wang and Ma's (WM) code made into a function with the same parameter profile as the QTFM function.
- `structured_test.m` — a script to run the QTFM and WM code for various matrix sizes and plot the results. The data in the variables `t1` and `t2` is available for inspection after the code runs. The corresponding matrix sizes are in variable `S`.

The QTFM code is not made available as supplementary material, as it is a large toolbox, and is freely available from the Sourceforge site [2]. Version 1.9 (used by Wang and Ma) and later versions are available there. The QTFM LU code is in the source file named `lu.m` which will be found in the directory `@quaternion`.

[1] M. Wang, W. Ma, A structure-preserving method for the quaternion LU decomposition in quaternionic quantum theory, Computer Physics Communications 184 (9) (2013) 2182–2186.

[2] S. J. Sangwine, N. Le Bihan, Quaternion Toolbox for Matlab®, [Online], software library available at: `http://qtfm.sourceforge.net/` (2005).

[3] G. H. Golub, C. F. van Loan, Matrix Computations, 3rd Edition, Johns Hopkins studies in the Mathematical Sciences, The Johns Hopkins University Press, Baltimore and London, 1996.

[4] R. Nibali, Enclosing MATLAB code in a function makes it super slow, Online forum thread, available at `www.mathworks.com/matlabcentral/newsreader/view_thread/247063` (19 March 2009).