Usage of algorithm of fast updates of QR decomposition to solution of linear regression models

Michał Bernardelli
Institute of Econometrics
Warsaw School of Economic
Warsaw, Poland
michal.bernardelli@sgh.waw.pl

Abstract — **In the paper the description of the algorithm of solving the linear least squares problem with QR decomposition method and their analysis of usefulness to the fast changing data is presented. There exist algorithms that compute the new factorization QR of the matrix with a row or column added or deleted by updating the factors Q and R. The new approach is based on the remark that the Q matrix is not needed explicitly to solve the linear regression problems. Therefore the algorithm of fast updates of the QR decomposition without constructing the orthogonal matrix is developed.**
The proposed algorithm is compared to the algorithm of the QR decomposition's updates using Givens rotations to show its superior computation speed. Results of the computer simulation for the data from the foreign exchange market are presented. For the completeness parts of the source code of the algorithm in Matlab-style are included.

Keywords - linear least square problem, QR decomposition updates, Givens rotation, linear regression, numerical algorithm

I. INTRODUCTION

A. The linear least squares problem

The unknown parameters of the linear regression model are often calculated as the solution of the linear least squares problem. The description of least squares problem in the linear and nonlinear form could be found for example in [2].

The solution of the linear least squares problem is a vector $x^* \in R^n$ such, that

$$\left\| \mathrm{b} - \mathrm{Ax}^* \right\|_2 = \min_{\mathrm{x} \in \mathrm{R}^n} \left\| \mathrm{b} - \mathrm{Ax} \right\|_2, \tag{1}$$

where $b \in R^m$ and the full rank matrix $A \in R^{m \times n}$ is overdetermined, that is $m \geq n$.

There were developed several ways of solving linear least squares problem, most of all the following:

1. Solving normal system of equations $A^T Ax = A^T b$ by finding pseudoinverse A^+
$$x^* = A^+ b = (A^T A)^{-1} A^T b. \tag{2}$$
This method however is not numerically efficient, see [5].
2. Using orthogonal decomposition methods. The QR factorization of the matrix A is given by
$$A = QR, \tag{3}$$
where $Q \in R^{m \times m}$ is orthogonal and $R \in R^{m \times n}$ is upper trapezoidal, that is matrix with elements $r_{ij} = 0$ for $i>j$. Having the factors Q and R computed, one gets the equivalent system of equations
$$Rx = Q^T b = d, \tag{4}$$
which is easy to solve because of the special form of the matrix R. Orthogonal decomposition methods are numerically stable.

B. The QR factorization

There are couple algorithms of calculating the QR decomposition of the given matrix A. All of them vary in computational and numerical complexity. The basic methods are, see [4], [6] or [8]:

1. Gram-Schmidt orthogonalization,
2. Householder reflections,
3. Givens rotations.

Codes of those algorithms could be found in [7] and [9]. The cost of the most efficient method, Householder reflections, is only $2mn^2$ floating point operations (flops). This is however the cost of the factorization without constructing the orthogonal matrix Q explicitly. To get the factor Q one need approximately m^4 extra floating point operations. For two-column matrix A, that is for n=2, the situation is better – the cost of factorization and constructing both Q and R matrices is $4m^2$ flops.

Besides the computational speed of the algorithms it is important to consider the memory necessary for those algorithms to work. Definitely the most consuming is the storage of the matrix Q. In many real-life cases m is much bigger then n. Therefore the memory needed for the storage of the orthogonal factor of the QR decomposition is proportional to m^2. Using Householder reflections or Givens rotations it is enough to store data proportional to mn, which is much less then storing the matrix Q explicitly. For instance each Givens rotation, see [3], is exactly specified by the angle of rotation, sinus or cosines of that angle. This is only one scalar for each of those unitary transformations. It is easy to calculate how many Givens rotations are needed to get the QR factorization of the given matrix A. For the first column it is m-1 rotations, m-2 for the second column and generally m-k Givens transformations for the k-th column. Summing all of that one gets exactly (m-n-1)n/2 Givens rotations, which is much less than m^2 – the cost of storing the whole matrix Q.

It is clear that adding into algorithm the construction of the matrix Q changes dramatically the memory and computational cost of the whole algorithm. The matrix Q however is needed only to calculate the vector of the right side of the system of equations (4). One could achieve this goal actually even without constructing this orthogonal matrix. For example stored numbers defining Givens rotations could be used to transform the vector b from (1) to the vector d form (4) just by doing the sequence of those rotations. This transformation is less time and memory consuming than constructing the matrix Q and multiplying it by the vector b.

C. Fast changing data

There are situations where is a need to solve many linear least squares problems with similar matrices and right side vectors. The typical example is a sequence of constantly coming data that are the inputs to the linear regression model. In that case the two consecutive models differ only in the new just read row[1] of data. One solution is to compute the QR factorization each time. Instead it is better to update the previous factorization due to the newly received data.

Consider a problem with a fast changing data and a linear regression model build only on the newest m rows. There are two basic steps to perform to get the up-to-date QR factorization, see Fig. 1:

1. procedure qrdelete – update of the QR factorization after removing the oldest data (first row),
2. procedure qrinsert – update of the QR factorization after adding the newest data (last row).

The names of the procedures, qrdelete and qrinsert, are taken from the article [1].

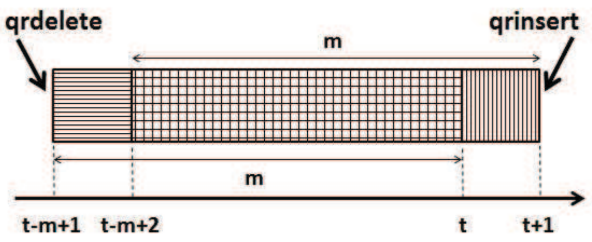


Figure 1. Two steps of updating the QR factorization – procedures qrdelete and qrinsert.

[1] Similarly, you can consider the new columns instead of rows.

Of course the difference in computational complexity between the full QR factorization algorithm and procedures updating the existing factors of the decomposition are negligible for small values of m. The bigger number of rows of the matrix A the more noticeable superiority of the updating algorithms over the QR decomposition method. In [1] there are presented the results of the computer simulation confirming that fact.

D. Updating the factors Q and R

The QR factorization update procedures are based on Givens rotations. The notation $T_{i,j} \in R^{m\times m}$ will be used for the Givens matrix which is the representation of a Givens rotation in the (i, j) plane. In [1] there are included the Matlab-style source codes of the procedures qrdelete and qrinsert. You can also find there the theoretical proof of computational cost of those procedures.

The idea behind updates of the factors Q and R is presented for the completeness of the description and the comparison purpose in the further parts of this paper. We start with the qrinsert procedure. Given the factorization of the matrix A as in (3) and the new row of data r_{m+1} the new equation could be written it the form

$$\begin{pmatrix} A \\ r_{m+1} \end{pmatrix} = \begin{pmatrix} Q & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R \\ r_{m+1} \end{pmatrix}. \tag{5}$$

Using the sequence of Givens rotations

$$T = T_{1,m+1} T_{2,m+1} \dots T_{n-1,m+1} T_{n,m+1} \tag{6}$$

the last row could be transformed into the zero vector. The updated factors $\widetilde{Q}$ and $\widetilde{R}$ are then defined as

$$\widetilde{Q} = \begin{pmatrix} Q & 0 \\ 0 & 1 \end{pmatrix} T \text{ and } \widetilde{R} = T^T \begin{pmatrix} R \\ r_{m+1} \end{pmatrix}. \tag{7}$$

It is worth to stress that the matrix Q is not important in the context of computing the transformation matrix T and the final result of the linear least squares problem.

To present the concept of the procedure qrdelete lets rewrite at first the equation (3) in the form

$$\begin{pmatrix} a_1 \\ A_{rest} \end{pmatrix} = \begin{pmatrix} q_1 \\ Q_{rest} \end{pmatrix} \begin{pmatrix} r_1 \\ R_{rest} \end{pmatrix}. \tag{8}$$

The main step of the procedure is to use the sequence of Givens rotations

$$T = T_{m-1,m} T_{m-2,m-1} \dots T_{3,4} T_{2,3} T_{1,2} \tag{9}$$

to transform q_1, the first row of the matrix Q, to the unit vector getting

$$\begin{pmatrix} a_1 \\ \widetilde{A} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \widetilde{Q} \end{pmatrix} \begin{pmatrix} \widetilde{r}_1 \\ \widetilde{R} \end{pmatrix}, \tag{10}$$

where $\widetilde{A} = A_{rest}$,

$$\begin{pmatrix} 1 & 0 \\ 0 & \widetilde{Q} \end{pmatrix} = \begin{pmatrix} q_1 \\ Q_{rest} \end{pmatrix} T^T \text{ and } \begin{pmatrix} \widetilde{r}_1 \\ \widetilde{R} \end{pmatrix} = T^T \begin{pmatrix} r_1 \\ R_{rest} \end{pmatrix}. \qquad (11)$$

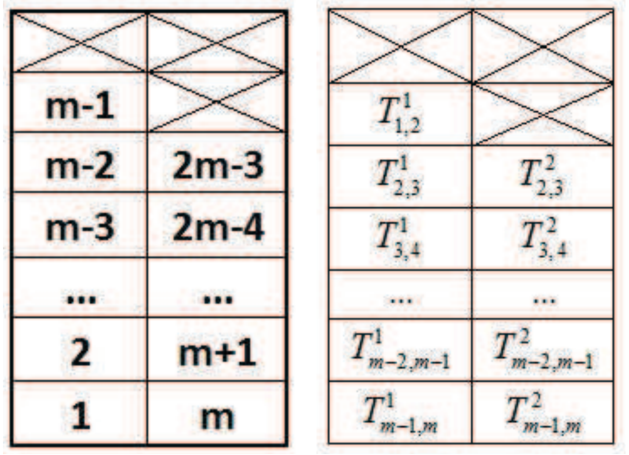
Let's emphasize that, in contrast to the procedure qrinsert, matrix Q is essential in computing the updated factors $\widetilde{Q}$ and $\widetilde{R}$ in the procedure qrdelete. Therefore even if the factor Q is not needed directly in solving linear least squares problem, it is required during calculations while deleting one of the row from the data. This is the main motivation for finding the new algorithm with at least the same numerical efficiency but competitive in terms of computational and memory cost. It could be achieved by developing an algorithm that uses the factor Q in its implicit form.

II. DESCRIPTION OF THE QR DECOMPOSITION UPDATES ALGORITHM

A. *Data structure description*

The proposed algorithm uses the Givens rotations represented by the pair of scalars (s; c) meaning sinus and cosines of the angle of rotation. The generalization of the notation $T_{i,j}^k \in R^{m\times m}$ will be used. It represents the Givens rotation in the (i, j) plane for the matrix elements from the k-th column. The description of the method will be limited to the case n=2 which corresponds to the situation of the simplest linear regression model, that is fitting the straight line to the given data.

Let S be the matrix of all sinuses and C the matrix of all cosines of the angle of rotation numbered in the order of their usage as in Fig. 2. More precisely the QR factorization of the two-column matrix A using the Givens rotations method is represented by the matrices $R \in R^{m\times n}$, $S \in R^{m-1\times n}$ and $C \in R^{m-1\times n}$, where S and C are equivalent to the matrix Q. The pair (s_{ij}, c_{ij}) of elements of the matrices S and C define the Givens rotation number (j-1)(2m-j)/2+i for j=1,2,…,n and i=1,2,…, m-j.



(a) (b)

Figure 2. (a) The order and (b) the definition of the Givens rotations in algorithm of the QR factorization of matrix A.

Using matrices R, S and C as a data structure every linear least squares problem could be written at the expense of memory proportional to the mn instead of m^2 scalars, like in case of the decomposition matrix A to factors Q and R. This presentation of the problem is convenient and useful in the construction of the algorithms of updating the QR factorization.

B. *Deleting first row*

The idea of the algorithm of deleting the first row of the data is described in detail in [1]. To find the proper transformation matrix the elements of the first row of the matrix Q are needed. Using the data structure with matrices S and C it is enough to do the reverse Givens rotations $T^2_{m,m-1}$ and $T^1_{m,m-1}$ and delete the last elements in columns of the matrices S, C, R and d. The main part of the code of the procedure qrdelete_new in Matlab-style is given in Fig. 3.

Both costs of the procedure qrdelete_new, the memory and the computational, are constant. This is enormous difference compared to the original procedure qrdelete from [1], which has the cost of $4m^2$ flops in case n=2.

C. *Inserting new row*

As was mentioned before, inserting the new row of the data does not require the matrix Q itself, but only the second factor R. Therefore the beginning of the procedure qrinsert_new is basically the same as the code of the procedure qrinsert from [1]. This part of the source code is presented in Fig. 4.

What is new in the presented algorithm is the need to update the two others matrices S and C. After the first step of the algorithm the ordering of the Givens rotations is not correct, see Fig. 5, and should be changed.

Procedure qrdelete_new

input: C, S, R, d, m

output: $\widetilde{C}$, $\widetilde{S}$, $\widetilde{R}$, $\widetilde{d}$

```
R(2:3,:) = [C(m-2,2) S(m-2,2);
            -S(m-2,2) C(m-2,2)]*R(2:3,:)
R(1:2,:) = [C(m-1,1) S(m-1,1);
            -S(m-1,1) C(m-1,1)]*R(1:2,:)
d(2:3) = [C(m-2,2) S(m-2,2);
          -S(m-2,2) C(m-2,2)]*d(2:3)
d(1:2) = [C(m-1,1) S(m-1,1);
          -S(m-1,1) C(m-1,1)]*d(1:2)
C(m-2,2) = 0
S(m-2,2) = 0
```

$\widetilde{C}$ = C(1:m-2,:)

$\widetilde{S}$ = S(1:m-2,:)

$\widetilde{R}$ = R(2:m,:)

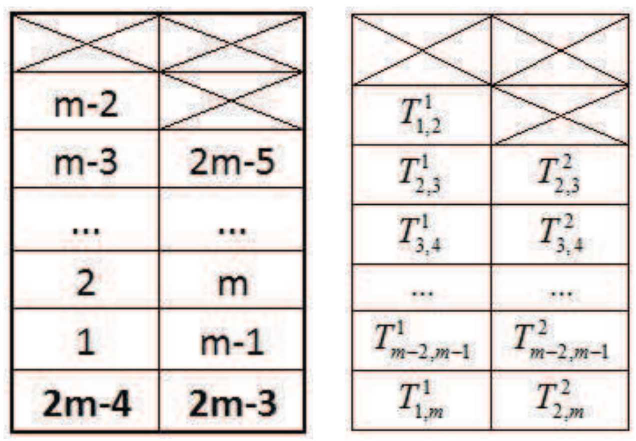
$\widetilde{d}$ = d(2:m)

Figure 3. The code of the procedure qrdelete_new.

```
Procedure qrinsert_new, Part 1
input: C, S, R, d, m, vec, num
output: $\widetilde{C}$, $\widetilde{S}$, $\widetilde{R}$, $\widetilde{d}$
________________________________________

% adding the new row to the matrix R

R(m+1,:) = vec
d(m+1) = num
m = m+1

% transforming the vector d and the matrix R
% using the Givens rotations

[C(m-1,1), S(m-1,1)] = givens(R(1,1), R(m,1))
R(1,1) = C(m-1,1)*R(1,1) - S(m-1,1)*R(m,1)
R(m,1) = 0
tmp = R(1,2)
R(1,2) = C(m-1,1)*R(1,2) - S(m-1,1)*R(m,2)
R(m,2) = S(m-1,1)*tmp + C(m-1,1)*R(m,2)

tmp = d(1)
d(1) = C(m-1,1)*d(1) - S(m-1,1)*d(m)
d(m) = S(m-1,1)*tmp + C(m-1,1)*d(m)

[C(m-2,2), S(m-2,2)] = givens(R(2,2), R(m,2))
R(2,2) = C(m-2,2)*R(2,2) - S(m-2,2)*R(m,2)
R(m,2) = 0
```

Figure 4. The first part of the code of the procedure qrinsert_new.

The concept behind updating the Givens transformations is based on few facts. Firstly it is important to notice that the multiplication of the matrix representation of the rotation $T^1_{1,m}$ is commutative with the Givens rotations of the second column that are before in the order.



(a)

☒	☒
m-2	☒
m-3	2m-5
...	...
2	m
1	m-1
2m-4	2m-3

(b)

☒	☒
$T^1_{1,2}$	☒
$T^1_{2,3}$	$T^2_{2,3}$
$T^1_{3,4}$	$T^2_{3,4}$
...	...
$T^1_{m-2,m-1}$	$T^2_{m-2,m-1}$
$T^1_{1,m}$	$T^2_{2,m}$

Figure 5. (a) The order and (b) the definition of the Givens rotations after the first step of the procedure qrinsert_new.

$$R^{k\times k} \ni T_{i,i+1} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & c_i & s_i & 0 \\ 0 & -s_i & c_i & 0 \\ 0 & 0 & 0 & I \end{bmatrix}$$

Figure 6. The structure of the matrix representation of the Givens rotation.

That is why the situation before and after correction updates could be summarized as follows

$$d = \underbrace{T^1 T^2 R}_{before} = \underbrace{\widetilde{T}^1 \widetilde{T}^2 R}_{after}, \tag{12}$$

where T^1 and T^2 are matrices representing sequences of rotations connected to first and second column respectively before the update, whereas $\widetilde{T}^1$ and $\widetilde{T}^2$ are analogous matrices after reordering.

The most important conclusion is that the updates could be divided into two separate steps. At the beginning the first columns are changed to the proper sets of rotations and then the second columns of matrices S and C are being corrected.

Secondly it is useful to know the structure of matrix created by the multiplication of number of Givens matrices. If by I the identity matrix is denoted and the matrix representation of one Givens rotation has the structure as in Fig. 6, then the matrix representation of the sequence of consecutive Givens rotations has the structure shown in the Fig. 7. The elements of that matrix are given by the formula

$$t_{i,j} = \begin{cases} s_i & j = i+1 \\ (-1)^{i+j} c_{j-1} s_j s_{j+1} \cdots s_{i-1} c_i & j \le i < k \\ (-1)^{i+j} c_{j-1} s_j s_{j+1} \cdots s_{i-1} s_i & j \le i = k \\ 0 & j > i \end{cases} \tag{13}$$

where it is assumed that c_0=1.

The regularity of this structure is used in the construction of the updating algorithm. To find the scalars representing the Givens rotations in the first column it is enough to compare first columns of the matrices before

$$T^1 = T^1_{m-2,m-1} \ldots T^1_{2,3} T^1_{1,2} T^1_{1,m} \tag{14}$$

and after

$$\widetilde{T}^1 = T^1_{m-1,m} T^1_{m-2,m-1} \ldots T^1_{2,3} T^1_{1,2} \tag{15}$$

$$R^{k\times k} \ni T_{k-1,k}\cdots T_{3,4}T_{2,3}T_{1,2} = \begin{bmatrix} c_1 & s_1 & \cdots & 0 \\ -s_1c_2 & c_1c_2 & \cdots & 0 \\ s_1s_2c_3 & -c_1s_2c_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ (-1)^{k+1}s_1\cdots s_{k-2}s_{k-1} & (-1)^{k+2}c_1s_2\cdots s_{k-2}c_{k-1} & \cdots & (-1)^{k+k}c_4s_5\cdots s_{k-2}c_{k-1} \end{bmatrix}$$

Figure 7. The structure of the matrix representation of the sequence of Givens rotations.

this first step of updating procedure. It is worth to emphasize that those two matrices T^1 and $\widetilde{T}^1$ are in general not equal. It is only the first columns that are the same. The detailed code of this part of the procedure qrinsert_new is presented in Fig. 8. The computational cost is proportional to m and for algorithm to work only constant extra memory is needed.

The last part of the reordering of the Givens rotations is related to the second column of the matrices. After the previous step all elements of the matrix $\widetilde{T}^1$ are known, so from (12) we get

$$\widetilde{T}^2 R = \left(\widetilde{T}^1\right)^T d. \tag{16}$$

To get the elements of $\widetilde{T}^2$ it is enough to compare the second columns of matrices form both sides of (16). The code in Matlab-style of the last step of reordering and at the same time the last part of the procedure qrinsert_new is presented in Fig. 9.

As in the first step of updating the structure of matrices S and C, the computational cost is proportional to m. Both costs of the procedure qrdelete_new, the memory and the computational, are constant. Therefore the cost of the whole procedure qrinsert_new is the same, up to the constant, as the cost of the original procedure qrinsert from [1] and is equal approximately m flops in case n=2.

```
% updating the first column of C and S
tmp1 = C(m-1,1)
tmp2 = S(1,1)
S(1,1) = S(m-1,1)
for i = m-1:-1:2
        C(i,1) = tmp1 * C(i-1,1)
        S(i,1) = sqrt(1 - C(i,1)^2)
        S(1,1) = S(1,1) / S(i,1)
        tmp1 = tmp1 * S(i-1,1) / S(i,1)
end
C(1,1) = tmp1 / S(1,1) * tmp2
```

Figure 8. The second part of the code of the procedure qrinsert_new.

```
% updating the second column of C and S
right = d/R(2,2)
tmp = 1
flag = 1
for i = m-2:-1:2
        C(i,2) = flag * tmp * right(m-i)
        S(i,2) = sqrt(1 - C(i,2)^2)
        tmp = tmp / S(i,2)
        flag = -flag
end
C(1,2) = flag * tmp * right(m-1)
S(1,2) = sign(right(m)) * sqrt(1 - C(1,2)^2)
```

Figure 9. The third part of the code of the procedure qrinsert_new.

III. Numerical Experiments

A. Foreign exchange market

The best way to confirm theoretical properties of the new developed algorithm and compare its computational speed with the efficiency of existing algorithms is to perform a computer simulation on real-life data. The data from the foreign exchange market, Forex,are perfectly suited for those purposes.

Forex is the market with discrete data, which are generated in small time intervals. On many websites there are available archival Forex data at different granulation time, for example 10 minutes, 1 minute, 10 seconds or even two seconds. The exemplary graph with the data from the Forex market is shown in Fig. 10.

Because of so short time intervals the on-line analysis of the Forex data used to predict increase or decrease of the exchange rate has strict time limits. The computational effectiveness of used in the analysis algorithms is the issue here. It is also the excellent example of linear regression model.

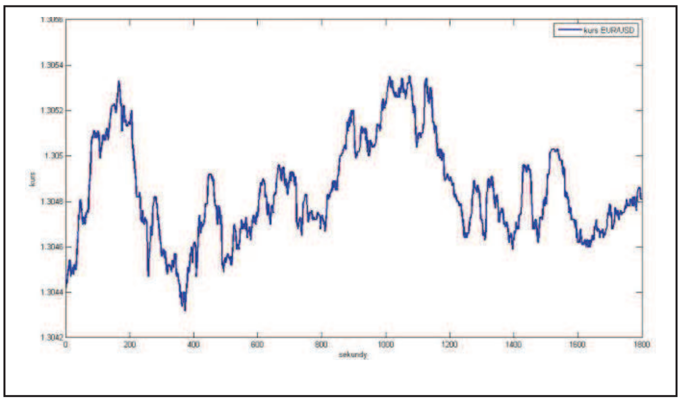


Figure 10. EUR to USD exchange rate for one hour of the day 16.04.2012.

B. *Speed tests*

In computer simulation there were used the source codes presented in this paper and the data from the Forex market in two-second granularity. There were five time series of the exchange rates considered: the euro to the Swiss franc (EUR / CHF), the euro to the Norwegian krone (EUR / NOK), the euro to the U.S. dollar (EUR / USD), silver to the U.S. dollar (AG / USD) and gold to the U.S. dollar (AU / USD).

For different number of rows of the linear regression matrix in the interval from 5 to 700 and for each of five time series of length 25000 the computations were performed. Two competitive methods were considered:

1. M_1 – the QR factorization updating algorithm with matrices Q and R constructed and remembered (procedures qrdelete and qrinsert from [1]),
2. M_2 – the QR factorization algorithm with matrices R, S and C described in this paper (procedures qrdelete_new and qrinsert_new).

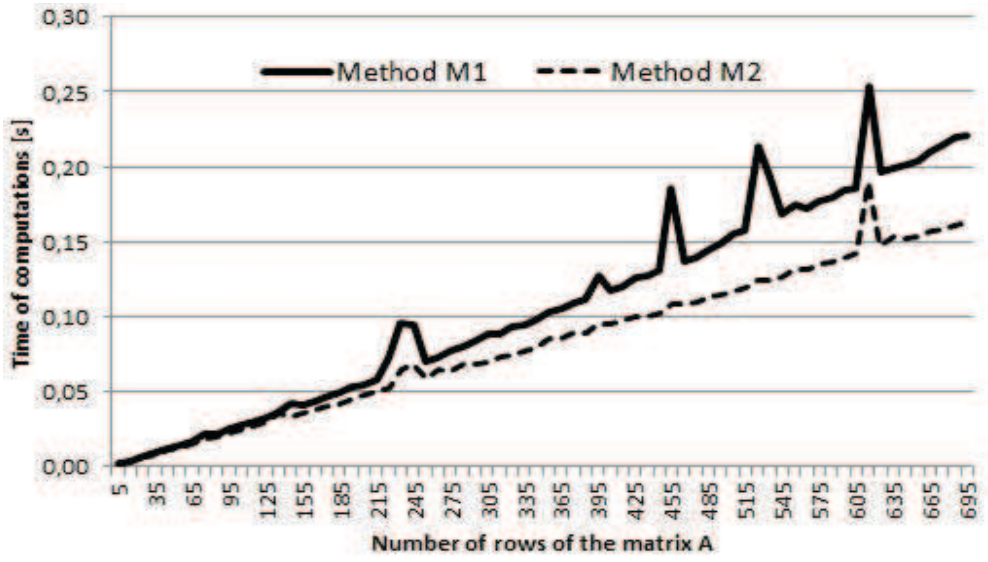
TABLE I. AVERAGE TIME OF COMPUTATIONS DEPENDING ON THE NUMEBER OF ROWS AND THE UPDATING ALGORITHM: M_1 AND M_2. MEASURED TIME ARE PRESENTED IN HUNDREDTHS OF A SECOND.

m / data	10		100	
	M_1	**M_2**	**M_1**	**M_2**
EUR/CHF	0.35	0.30	2.71	2.27
EUR/NOK	0.31	0.27	2.65	2.23
EUR/USD	0.32	0.28	2.62	2.21
AG/USD	0.31	0.28	2.66	2.23
AU/USD	0.31	0.28	2.59	2.20
AVERAGE	**0.32**	**0.28**	**2.65**	**2.23**

m / data	200		300	
	M_1	**M_2**	**M_1**	**M_2**
EUR/CHF	5.50	4.45	8.70	6.67
EUR/NOK	5.61	4.46	8.64	6.67
EUR/USD	5.42	4.39	8.45	6.53
AG/USD	5.47	4.46	8.48	6.53
AU/USD	5.37	4.36	8.42	6.51
AVERAGE	**5.48**	**4.42**	**8.54**	**6.58**

Computations were performed in Octave, program similar to Matlab, but distributed under the terms of the GNU General Public License. The time of computations was measured for every value of m and every m consecutive values of each of time series. Obtain times were averaged over the time series and different values of m. The results are gathered in Tab. I.

The data from the numerical experiment presented in Tab. I are clearly the confirmation of the theoretical theorems. This advantage is even better seen in Fig. 11. The new developed method of updating the QR factorization is faster for every value of m.



.

Figure 11. The comparison of the time of computations two methods M_1 and M_2 depending on number of rows of the matrix A.

IV. Conclusions

Theoretical analysis of new developed algorithm of updating the QR factorization together with the computer simulations leads to the following conclusions:

- the computational speed of the new algorithm is much better than the speed of existing algorithms,
- the numerical properties of the new algorithm is at least as good as the numerical accuracy of other methods based on orthogonal decomposition of the matrix A,
- procedure qrinsert_new is much more time consuming than the procedure qrdelete_new (proportional to m flops vs. constant number of operations),
- the memory complexity of the new method is much improved relatively to existing updating methods (proportional to m^2 vs. proportional to mn).

The presented algorithm of updating the QR factorization without storing the matrix Q explicitly is meant to be the foundation for solving simple linear least squares problem. It should be an excellent start for building faster and more efficient algorithms working on big and fast changing sets of data. Certainly, it is worth to consider the following generalizations and modifications of described in this paper algorithm:

- the extension to the polynomial regression models and to models with greater number of variables,
- the case of nonlinear regression.

References

[1] M. Bernardelli, "Method of QR decomposition's fast updates for linear regression models", Roczniki Kolegium Analiz Ekonomicznych, vol. 27/2012, Warsaw School of Economic, Warsaw, 2012 (in press).

[2] A. Björck, "Numerical methods for least squares problems", SIAM, Philadelphia, USA, 1996.

[3] M. Dryja, J. and M. Jankowscy, „Przegląd metod i algorytmów numerycznych." Vol. 2, Wydawnictwa Naukowo-Techniczne, Warsaw, 1988.

[4] G. H. Golub, Ch. F. Van Loan, "Matrix Computations (2nd ed.)", Johns Hopkins University Press, 1990.

[5] A. Kiełbasiński, H. Schwetlick, „Numeryczna algebra liniowa", Wydawnictwa Naukowo-Techniczne, Warsaw, 1992.

[6] D. Kincaid, W. Cheney, „Numerical Analysis: Mathematics of Scientific Computing, 3rd Edition", American Mathematical Society, Providence, RI, 2002.

[7] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, "Numerical Recipes: The Art of Scientific Computing (3rd ed.)", Cambridge University Press, New York, 2007.

[8] J. Stoer, R. Bulirsch, "Introduction to Numerical Analysis (3rd ed.)", Springer, 2002

[9] L. N. Trefethen, D. Bau, "Numerical Linear Algebra", SIAM, Philadelphia, USA, 1997.