NON-ML Project 20 Report

Computational Mathematics for Learning and Data Analysis

Group 5

Sara Montemaggi, Matteo Ceregini

2021-2022

1 Problem description

The problem to be solved is the linear least squares problem

$$\min_w ||Xw - y||$$

where $X \in \mathbb{R}^{m \times n}$, with $m \gg n$, is given and can be found in the file *matrix.csv*, and $y \in \mathbb{R}^m$ is a random vector. In particular, the goal of the project is to implement the following algorithms:

(A1) Thin QR factorization via Householder reflectors.

(A2) LDL factorization applied to the square symmetric linear system:

$$\begin{bmatrix} I & X \\ X^T & 0 \end{bmatrix} \begin{bmatrix} r \\ w \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}$$

2 Conditioning of the problem

The conditioning of the linear least squares problem depends on both the matrix X and the input vector y. The relative condition number with respect to the input vector y is given by:

$$\kappa_{\text{rel},\, y \to w} \leq \frac{\kappa(X)}{\cos\theta}$$

and with respect to the matrix X:

$$\kappa_{\text{rel},\, X \to w} \leq \kappa(X) + \kappa(X)^2 \tan\theta$$

where $\kappa(X)$ is the condition number of the matrix X, and θ is the angle such that $\cos\theta = \frac{||Xw||}{||y||} = \frac{||Q_1^T y||}{||y||}$. In particular, θ is the angle between the input vector y and $Im(X)$ [2].

In this case, we have $\kappa(X) = 65.7987$. Since $\kappa(X)$ is relatively small, we can vary the input vector y, and the resulting θ, to obtain more ill-conditioned problems. In particular, $\theta \approx 90°$ will give us a more ill conditioned problem, instead $\theta \approx 0°$ will give us well conditioned problems.

In order to test the two algorithms we generate different problem instances by generating vectors y that are at different angles θ with $Im(X)$. In this way, we obtain problems with various levels of ill-conditioning. Input vectors are computed as follows:

1. Compute the QR factorization $X = QR = \begin{bmatrix} Q_1 Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$. The columns of Q_1 are a basis of $Im(X)$ and the columns of Q_2 are a basis for the rest of the space.
2. Generate a random vector $v \in \mathbb{R}^m$ and compute $y = Qv = Q_1 v_1 + Q_2 v_2$.
3. Since $||Q_1 v_1|| = ||v_1||$ and $||Q_2 v_2|| = ||v_2||$, we can vary the angle θ by multiplying v_1 or v_2 for a factor α, so to scale differently the y component on $Im(X)$ and the y component on the rest of the space.

3 Algorithm A1

3.1 Brief algorithm description

We have to compute the thin QR factorization of the matrix X. In order to do this we use the following algorithm:

Algorithm 1 Compute the thin QR factorization of X: $X = Q_1 R_1$

// Compute the R_1 matrix and the householder vectors.
Initialize U as a zero matrix of size $m \times n$.
for j = 1 **to** n **do**
  Compute the householder vector u_j of $X_{j:m,j}$ and store it in the j-th column of U.
  Set $X_{j,j} = ||X_{j:m,j}||_2$ and $X_{j+1:m,j} = 0$
  $X_{j:m,j+1:n} = X_{j:m,j+1:n} - 2u_j(u_j^T X_{j:m,j+1:n})$
end for
$R1 = X_{1:n,1:n}$
// Compute the Q_1 matrix using the householder vectors.
Initialize Q_1 as an identity matrix of size $m \times n$.
for j = n **down to** 1 **do**
  $Q_1 = Q_1 - 2u_j(u_j^T Q_1)$
end for
return Q_1, R_1

Then, the least squares problem $\min_w ||Xw - y||$ can be solved as follows:

1. Compute the vector $c = Q_1^T y$.
2. Solve the triangular linear system $R_1 w = c$ via back-substitution [3].

3.2 What to expect

The matrix X has full column rank, therefore the least squares problem $\min_w ||Xw - y||$ has a unique solution w.

As shown before, in order to compute the vector w, three steps are necessary: computing the thin QR factorization $X = Q_1 R_1$, computing the vector $c = Q_1^T y$ and solving the linear system $R_1 w = c$. All these steps are solved using backward stable algorithms. Therefore, the whole procedure to compute w is backward stable and we can expect w to be a good approximation, in terms of machine precision, of the actual solution.

The overall cost of computing the solution w is dominated by the cost of computing the QR factorization of the matrix X. The computational cost of the thin QR factorization via Householder reflectors is $\sim 2mn^2 - \frac{2}{3}n^3$ flops [4]. Since we have that $m \gg n$, the complexity can be approximated as $O(2mn^2)$.

Moreover, this implementation of the thin QR factorization algorithm does not construct the Q matrix but only stores the Householder reflectors and the Q_1 matrix, requiring $O(mn)$ space. Therefore, the overall space complexity is $O(mn)$.

3.3 Testing of the algorithm

We test our implementation of the algorithm to check if it produces accurate results. Results and performances are compared against the ones obtained using the built-in Matlab utilities.

First of all, we compare the residuals of the QR factorization. Residuals for the QR factorization of a matrix A are obtained as:

$$\frac{||A - \tilde{Q}\tilde{R}||}{||A||} \tag{1}$$

where $\tilde{Q}$ and $\tilde{R}$ are the matrices obtained as output of a factorization algorithm. In Table 1 are shown both the residuals obtained by applying our thin QR factorization algorithm and Matlab's `qr` function to the matrix X. Residuals obtained are in the order of machine precision.

Residuals	3.7662×10^{-16}
Matlab's residuals	2.5032×10^{-16}

Table 1: Residuals obtained by applying the QR factorization to the matrix X.

The thin QR factorization algorithm is also tested on some additional randomly generated tall-thin matrices of various sizes and, again, residuals are in the order of machine precision. Residuals obtained are reported in Table 2: for each dimension, 100 matrices were randomly generated, and the mean residual for each dimension is reported. Residuals are also plotted in Figure 2, we can see here that residuals tend to increase slightly as the matrix size increases. Therefore, the QR factorization algorithm seems to be slightly less accurate on larger matrices. In Table 2 are also reported the mean execution times needed to compute the QR factorization of the matrices of different sizes, execution times are then plotted in Figure 1. We note that the increase in the execution times roughly fits the theoretical time complexity of $O(mn^2)$ of the factorization algorithm: as expected, there is a huge increase in the execution time when n, the number of columns of the matrices, increases from 10 to 100. Moreover, execution time increases roughly linearly with m: this is more evident on the larger matrices, for example we see that by doubling m and keeping the same n the execution time more or less doubles.

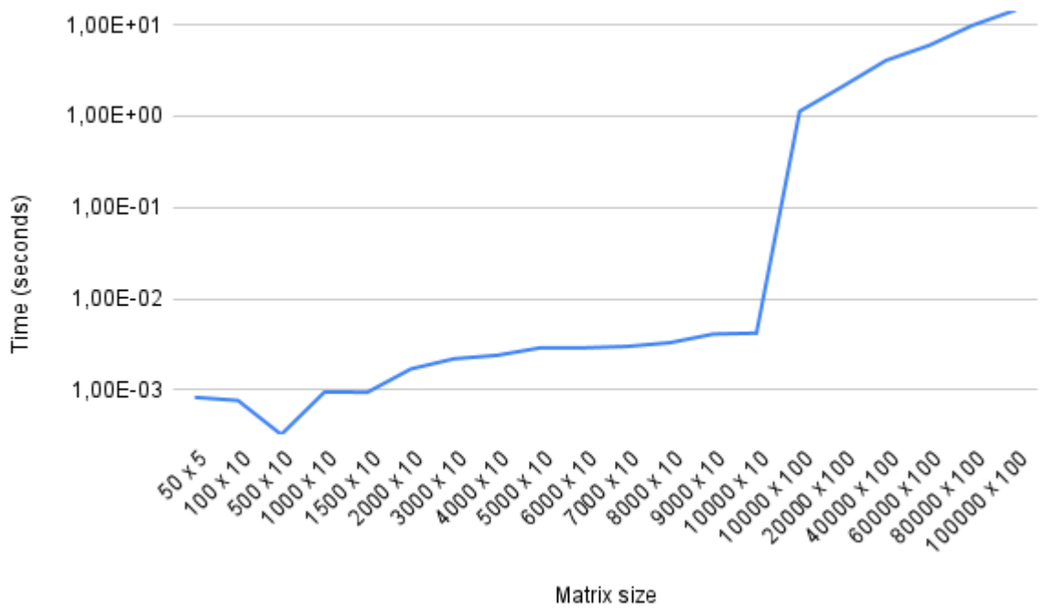

Figure 1: Average time needed to compute the QR factorization of test matrices of different sizes.

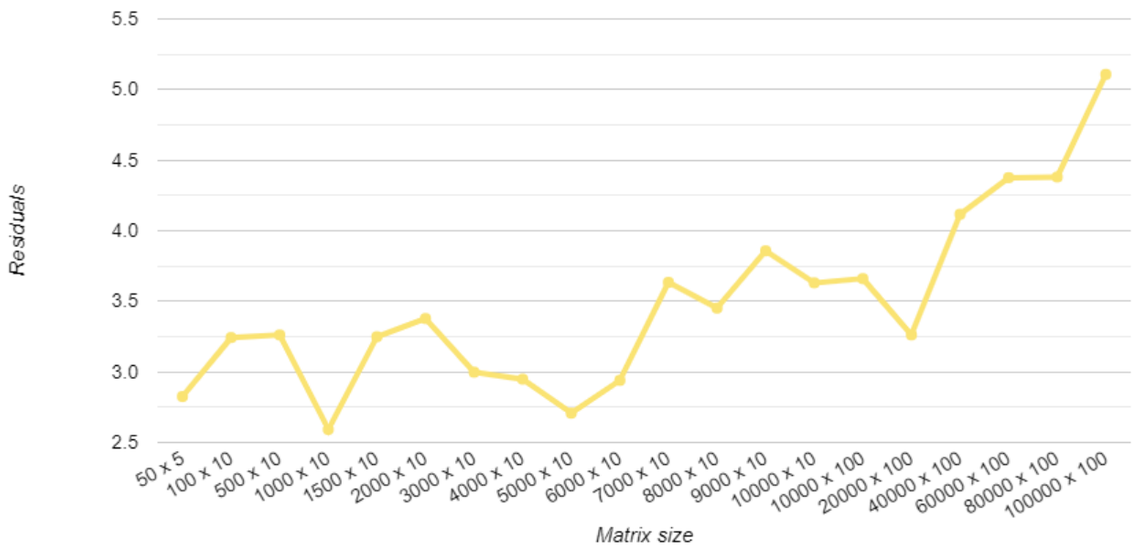

Figure 2: Residuals of the QR factorization ($\times 10^{-16}$) for different matrix sizes.

Matrix size	**Residual**	**Execution time (seconds)**
50×5	2.8258×10^{-16}	8.3177×10^{-4}
100×10	3.2433×10^{-16}	7.6887×10^{-4}
500×10	3.2616×10^{-16}	3.2574×10^{-4}
1000×10	2.5926×10^{-16}	9.4862×10^{-4}
1500×10	3.2491×10^{-16}	9.4726×10^{-4}
2000×10	3.3781×10^{-16}	0.0017
3000×10	2.9981×10^{-16}	0.0022
4000×10	2.9485×10^{-16}	0.0024
5000×10	2.7079×10^{-16}	0.0029
6000×10	2.9402×10^{-16}	0.0030
7000×10	3.6343×10^{-16}	0.0033
8000×10	3.4520×10^{-16}	0.0036
9000×10	3.8576×10^{-16}	0.0041
10000×10	3.6310×10^{-16}	0.0042
10000×100	3.6608×10^{-16}	1.1270
20000×100	3.2620×10^{-16}	2.1204
40000×100	4.1166×10^{-16}	4.0698
60000×100	4.3762×10^{-16}	5.9460
80000×100	4.3809×10^{-16}	9.8466
100000×100	5.1076×10^{-16}	14.4915

Table 2: Mean residuals and mean execution times obtained by applying the QR factorization to test matrices of different sizes.

Next, we test the accuracy of the computed solution $\tilde{w}$ of the problem $\min_w ||Xw - y||$ obtained by applying algorithm A1. Different instances of the problem are obtained by generating vectors y at different angles with $Im(X)$, using the method explained in Section 2. To evaluate the accuracy of the obtained results we look at residuals, computed as:

$$\frac{||Q_1^T(X\tilde{w} - y)||}{||y||} \tag{2}$$

The vectors used to test the algorithm are divided in 10 sets: each set contains 100 vectors having approximately the same angle θ with $Im(X)$. In Table 3 are reported the obtained mean residuals and condition numbers for each set of vectors.

Mean θ	**Mean residual**	$\kappa_{\text{rel},\, y \to w}$	$\kappa_{\text{rel},\, X \to w}$
0.00°	3.2525×10^{-16}	65.7987	65.7987
0.64°	3.2054×10^{-16}	65.8028	1.1416×10^{2}
6.52°	3.1594×10^{-16}	66.2270	5.6061×10^{2}
15.85°	2.9563×10^{-16}	68.3992	1.2950×10^{3}
28.94°	2.8102×10^{-16}	75.1876	2.4597×10^{3}
48.98°	2.0784×10^{-16}	1.0025×10^{2}	5.0428×10^{3}
69.68°	1.2088×10^{-16}	1.8948×10^{2}	1.1757×10^{4}
84.94°	7.0826×10^{-16}	7.4603×10^{2}	4.8962×10^{4}
89.49°	6.7055×10^{-17}	7.3922×10^{3}	4.8645×10^{5}
90.00°	4.0039×10^{-17}	$\to +\infty$	$\to +\infty$

Table 3: Mean residual obtained for each set of vectors applying algorithm A1. The vectors in a given set have approximately the same angle θ with $Im(X)$.

We see that the residuals decreases as θ increases. Despite the problems becoming more ill-conditioned as θ increases, algorithm A1 still computes an accurate solution, since all obtained residuals are in the order of machine precision.

To compute each solution our implementation takes on average 1.0000×10^{-3} seconds, while solving the problem with Matlab's \ operator takes on average 1.9667×10^{-4} seconds[1]. Matlab's implementation is faster by one order of magnitude, however this result was to be expected since Matlab's code is highly optimized.

4 Algorithm A2

4.1 Brief algorithm description

We describe here the LDL factorization algorithm with pivoting. A pivoting strategy is needed to improve the stability of the factorization. Different pivoting strategies have been proposed, the one we choose to implement is the Bunch-Parlett pivot strategy. The Bunch-Parlett method computes the factorization $PAP^T = LDL^T$, where L is lower triangular, D is a block-diagonal matrix made of 1-by-1 and 2-by-2 pivot blocks, and P is a permutation matrix chosen so that the entries in L satisfy $|l_{ij}| \leq 1$.

The Bunch-Parlett method defines the parameters $\alpha \in (0, 1)$ and

$$\mu_0 = \max_{i,j} |a_{ij}|,$$

$$\mu_1 = \max_{i} |a_{ii}|$$

and uses them to choose, at each step, what type of pivot to use: if $\mu_1 \geq \alpha\mu_0$ then a 1-by-1 pivot is used, otherwise a 2-by-2 block pivot is used.

The following pseudocode illustrates the overall factorization algorithm:

[1]Our test machine has the following characteristics: Windows 10 OS, Intel i7-1065G7 CPU (1.30 GHz, 1498 Mhz, 4 cores and 8 logical processors) and 8 GB of RAM.

Algorithm 2 Compute LDL factorization of $A \in \mathbb{R}^{n \times n}$ using the Bunch-Parlett pivoting strategy: $PAP^T = LDL^T$

Initialize L, P as identity matrices of size $n \times n$.
$\alpha = \frac{1+\sqrt{17}}{8}$ *// Optimal value of α according to Bunch and Parlett [1].*
$k = 1$
while $k \leq n - 1$ **do**
  Let $\mu_0 = \max_{i,j} |A_{i,j}|$
  Let $\mu_1 = \max_i |A_{i,i}|$
  if $\mu_1 \geq \alpha \mu_0$ **then**
    // 1-by-1 pivoting.
    Choose the permutation such that $|A_{k,k}| = \mu_1$, permute A, P and L accordingly.
    // Update A and L.
    $L_{k+1:n,k} = A_{k+1:n,k} \ / \ A_{k,k}$
    $A_{k+1:n,k+1:n} = A_{k+1:n,k+1:n} - L_{k+1:n,k} A_{k,k+1:n}$
    $A_{k+1:n,k} = 0$ and $A_{k,k+1:n} = 0$
    // Increment k by 1.
    $k = k + 1$
  else
    // 2-by-2 pivoting.
    Choose the permutation such that $|A_{k+1,k}| = \mu_0$, permute A, P and L accordingly.
    // Update A and L
    $L_{k+2:n,k:k+1} = A_{k+2:n,k:k+1} \ / \ A_{k:k+1,k:k+1}$
    $A_{k+2:n,k+2:n} = A_{k+2:n,k+2:n} - L_{k+2:n,k:k+1} A_{k:k+1,k+2:n}$
    $A_{k+2:n,k:k+1} = 0$ and $A_{k:k+1:n,k+2:n} = 0$
    // Increment k by 2.
    $k = k + 2$
  end if
end while
return $L, D = A, P$

Once the factorization is computed, it can be used to solve the linear system $Ax = b$:

$$PAP^T = LDL^T, \quad Lz = Pb, \quad Dw = z, \quad L^T y = w, \quad x = P^T y \implies Ax = b$$

where the $Dw = z$ system corresponds to a set of 1-by-1 and 2-by-2 symmetric indefinite systems. The $L^T y = w$ and $Lz = Pb$ systems can be solved via back-substitution [1].

4.2 What to expect

Since the matrix X has full column rank, the linear system to be solved is non-singular, therefore it has a unique solution [1].

The first step in solving the linear system consists in computing the LDL factorization of the matrix

$$A = \begin{bmatrix} I & X \\ X^T & 0 \end{bmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}$$

The Bunch-Parlett pivoting strategy has been implemented in order to improve the stability of the factorization. The next steps consist of solving two triangular linear systems and a

set of 1-by-1 and 2-by-2 symmetric linear systems, all of which can be solved using backward stable algorithms. The whole procedure to compute the vector $[r \quad w]^T$ is therefore backward stable and we can expect the computed solution to be a good approximation, in terms of machine precision, of the actual solution of the problem.

The overall cost of computing the solution $[r \quad w]^T$ is dominated by the cost of computing the LDL factorization. The computational cost of the LDL factorization is $\frac{2}{3}(n+m)^3$ flops since all the entries of the submatrix are computed at each step. Moreover, $O((n+m)^3)$ comparisons are needed to compute the pivots [1].

The space complexity of the algorithm is $O((m+n)^2)$.

4.3 Testing of the algorithm

As for algorithm A1, we test our implementation of the algorithm to check its accuracy. Results and performances are compared against the ones obtained using the built-in Matlab utilities.

We compare the residuals of the LDL factorization algorithm. Residuals for the LDL factorization of a matrix A are computed as:

$$\frac{||PAP^T - \tilde{L}\tilde{D}\tilde{L}^T||}{||PAP^T||} \tag{3}$$

where $\tilde{L}$, $\tilde{D}$ and P are the matrices returned by a LDL factorization algorithm.

In Table 4 are shown both the residuals obtained by applying our LDL factorization algorithm and Matlab's `ldl` function to the matrix A. Residuals obtained are in the order of machine precision, actually, with our implementation the factorization of the matrix A turns out to be little bit more accurate.

Residuals	2.5320×10^{-16}
Matlab's residuals	4.5298×10^{-16}

Table 4: Residuals obtained by applying the LDL factorization to the matrix A.

The LDL factorization algorithm is also tested on some additional randomly generated square symmetric matrices of various sizes and, again, residuals are in the order of machine precision. Residuals obtained are reported in Table 5, for each dimension, 100 matrices were randomly generated and the mean residual for each dimension is reported. Residuals are also plotted in Figure 4. In this case residuals do not seem to increase with increasing matrix sizes, however it was not possible to test larger matrices as too much execution time and space was required. In Table 5 are also reported the mean execution times needed to compute the LDL factorization of the matrices of different sizes, execution times are then plotted in Figure 3. Again, the increase in execution times roughly fits the theoretical time complexity: we see that by increasing the matrix dimensions of a factor α, the execution time increases of a factor more or less close to α^3. This is more evident on the larger matrices.

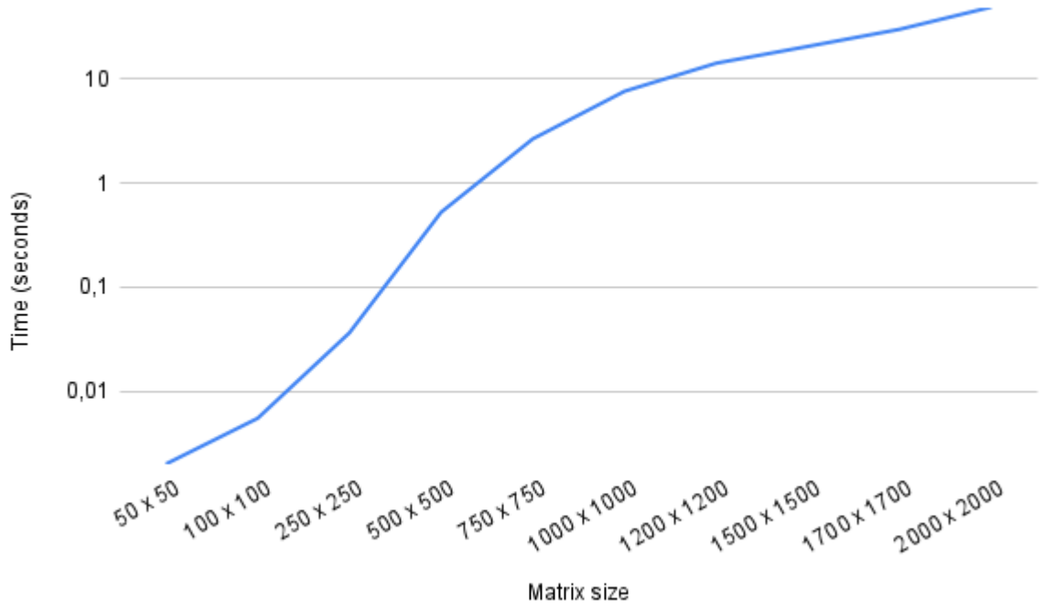

Figure 3: Average time needed to compute the LDL factorization of test matrices of different sizes.

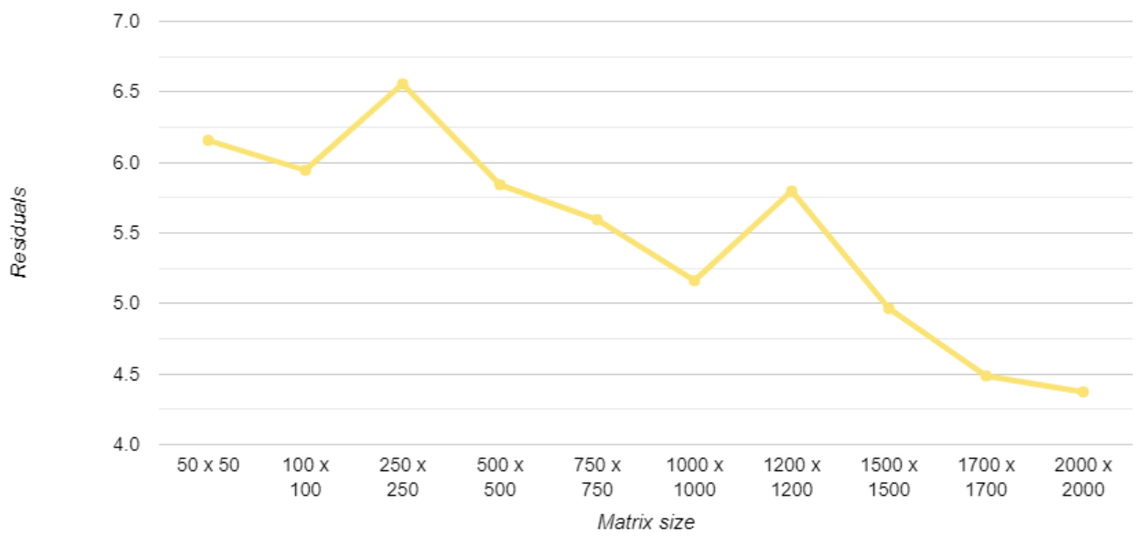

Figure 4: Residuals of the LDL factorization ($\times 10^{-17}$) for different matrix sizes.

Matrix size	**Residual**	**Execution time (seconds)**
50×50	6.1568×10^{-17}	0.0020
100×100	5.9451×10^{-17}	0.0055
250×250	6.5568×10^{-17}	0.0364
500×500	5.8415×10^{-17}	0.5250
750×750	5.5945×10^{-17}	2.6593
1000×1000	5.1616×10^{-17}	7.6201
1200×1200	5.7970×10^{-17}	14.1805
1500×1500	4.9642×10^{-17}	20.5723
1700×1700	4.4867×10^{-17}	30.0070
2000×2000	4.3733×10^{-17}	49.1055

Table 5: Mean residuals and mean execution times obtained by applying the LDL factorization to test matrices of different sizes.

Next, we test the accuracy of the computed solution $[\tilde{r} \quad \tilde{w}]^T$ of the linear system

$$\begin{bmatrix} I & X \\ X^T & 0 \end{bmatrix} \begin{bmatrix} r \\ w \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}$$

obtained by applying algorithm A2. To do this, we use the same input vectors y used to test the algorithm A1. To evaluate the accuracy of the obtained solution to the linear system we look at residuals, that, for the linear system $Ax = b$, are defined as

$$\frac{||b - A\tilde{x}||}{||A|| \times ||\tilde{x}||} \tag{4}$$

where $\tilde{x}$ is the solution returned by the algorithm. In Table 6 are reported the obtained residuals for each set of vectors, the sets are identified by the mean angle θ. We note that the mean residuals are all in the order of machine precision and they do not seem to increase or decrease as θ increases or decreases.

Mean θ	**Mean residual**
0.00°	3.0416×10^{-17}
0.64°	2.9610×10^{-17}
6.52°	4.6667×10^{-17}
15.85°	5.1271×10^{-17}
28.94°	4.7472×10^{-17}
48.98°	5.1254×10^{-17}
69.68°	5.5504×10^{-17}
84.94°	5.8968×10^{-17}
89.49°	5.1277×10^{-17}
90.00°	5.6972×10^{-17}

Table 6: Mean residuals obtained for the solution of the linear system for each set of vectors, the vectors in a given set have approximately the same angle θ with $Im(X)$.

We also evaluate the accuracy of the solution $\tilde{w}$ of the least squares problem obtained with this method by computing the residuals using Formula 2, as for algorithm A1. In Table 7 are reported the obtained mean residuals for each set of vectors. In this case, the residuals seem to increase as θ approaches 90° and the problems become more ill-conditioned.

Mean θ	**Mean residual**	$\kappa_{\text{rel, } y \to w}$	$\kappa_{\text{rel, } X \to w}$
0.00°	1.1951×10^{-15}	65.7987	65.7987
0.64°	1.1327×10^{-15}	65.8028	1.1416×10^{2}
6.52°	2.8653×10^{-15}	66.2270	5.6061×10^{2}
15.85°	7.3552×10^{-15}	68.3992	1.2950×10^{3}
28.94°	1.1605×10^{-14}	75.1876	2.4597×10^{3}
48.98°	2.1124×10^{-14}	1.0025×10^{2}	5.0428×10^{3}
69.68°	2.4242×10^{-14}	1.8948×10^{2}	1.1757×10^{4}
84.94°	2.9054×10^{-14}	7.4603×10^{2}	4.8962×10^{4}
89.49°	2.7340×10^{-14}	7.3922×10^{3}	4.8645×10^{5}
90.00°	2.8762×10^{-14}	$\to +\infty$	$\to +\infty$

Table 7: Mean residual obtained for each set of vectors applying algorithm A2. The vectors in a given set have approximately the same angle θ with $Im(X)$.

To compute each solution our implementation takes on average 20.7363 seconds. The majority of the time here is spent in computing the LDL factorization: both the complete pivoting strategy and the extra work done to compute the symmetric part of the matrix significantly slow down the computation.

5 Comparison of the algorithms and conclusions

In Table 8 we compare the residuals of the least squares problem obtained with the two methods. Residuals reported are the ones computed with Formula 2.

Although both methods compute quite accurate results, the mean residuals of algorithm A1 are smaller by one or two order of magnitudes than the ones of algorithm A2. Moreover, residuals of algorithm A2 increase as the angle θ approaches 90° and the problems become more ill-conditioned, while this does not happen with algorithm A1. Finally, method A1 is some orders of magnitude faster than method A2 in solving the least squares problem. For those reasons we conclude that algorithm A1 is the better one in solving the given problem.

Mean θ	**Mean residual A1**	**Mean residual A2**
0.00°	3.2525×10^{-16}	1.1951×10^{-15}
0.64°	3.2054×10^{-16}	1.1327×10^{-15}
6.52°	3.1594×10^{-16}	2.8653×10^{-15}
15.85°	2.9563×10^{-16}	7.3552×10^{-15}
28.94°	2.8102×10^{-16}	1.1605×10^{-14}
48.98°	2.0784×10^{-16}	2.1124×10^{-14}
69.68°	1.2088×10^{-16}	2.4242×10^{-14}
84.94°	7.0826×10^{-16}	2.9054×10^{-14}
89.49°	6.7055×10^{-17}	2.7340×10^{-14}
90.00°	4.0039×10^{-17}	2.8762×10^{-14}

Table 8: Mean residual and mean execution times obtained for each set of vectors, the vectors in a given set have approximately the same angle θ with $Im(X)$.

References

[1] Gene H. Golub and Charles F. Van Loan. “Matrix Computations”. In: The Johns Hopkins University Press. Chap. 4.4.

[2] Lloyd N. Trefethen and David Bau. “Numerical Linear Algebra”. In: Siam. Chap. 18.

[3] Lloyd N. Trefethen and David Bau. “Numerical Linear Algebra”. In: Siam. Chap. 11.

[4] Lloyd N. Trefethen and David Bau. “Numerical Linear Algebra”. In: Siam. Chap. 10.