Numerical Algorithms
Laboratory Report 1

Direct Methods for Solving Linear Systems

Submitted in partial fulfillment of
the requirements for the award of the degree of

Master of Science in
Advanced Applied Electronics

Submitted by

Roll No.	Name of Student
241194	Kamil Mężyński
226336	Konrad Dudziak

on 24th March 2021

Under supervision of
Rafał Zdunek, PhD

Wrocław University
of Science and Technology

Faculty of Advanced Applied Electronics

Summer Semester 2021

Contents

1 Laboratory tasks

1.1 Task 1

Solve the following system of equations and find the pivots.

$$\begin{cases} 2u - v & = 0 \\ -u + 2v - w & = 0 \\ -v + 2w - z & = 0 \\ -w + 2z & = 5 \end{cases}$$

1.1.1 Analytical solution

System of linear equations are expressed in augmented matrix and Gauss elimination was applied.

$$\left[\begin{array}{c|c} A & b \end{array}\right] = \left[\begin{array}{cccc|c} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & 5 \end{array}\right] \xrightarrow{R_2 = R_2 + 1/2R_1} \left[\begin{array}{cccc|c} 2 & -1 & 0 & 0 & 0 \\ 0 & 3/2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & 5 \end{array}\right]$$
$$\xrightarrow{R_3 = R_3 + 3/2R_2} \left[\begin{array}{cccc|c} 2 & -1 & 0 & 0 & 0 \\ 0 & 3/2 & -1 & 0 & 0 \\ 0 & 0 & 4/3 & -1 & 0 \\ 0 & 0 & -1 & 2 & 5 \end{array}\right] \xrightarrow{R_4 = R_4 + 3/4R_3} \left[\begin{array}{cccc|c} 2 & -1 & 0 & 0 & 0 \\ 0 & 3/2 & -1 & 0 & 0 \\ 0 & 0 & 4/3 & -1 & 0 \\ 0 & 0 & 0 & 5/4 & 5 \end{array}\right] \quad (1)$$

The results were computed using back substitution.

$$\begin{cases} u & = 1 \\ v & = 2 \\ w & = 3 \\ z & = 4 \end{cases}$$

1.1.2 MATLAB solution

To check the analytical solution, the Gauss elimination without pivoting and back substitution algorithms were used. Obtained matrix and vector of solutions are the same as analitical solution. MATLAB code is presented below:

```
A = [2 -1 0 0; -1 2 -1 0; 0 -1 2 -1; 0 0 -1 2];
b = [0; 0; 0; 5];

X = gaussNoPivot([A b])
x = backSubstitution(X)
```

Listing 1: main.m - task 1 source code

1.2 Task 2

Solve the following system of equations using Gaussian elimination with pivoting.

$$\begin{cases} x_1 + x_2 + x_3 & = 1 \\ x_1 + x_2 + 2x_3 & = 2 \\ x_1 + 2x_2 + 2x_3 & = 1 \end{cases}$$

1.2.1 Analytical solution

This problem cannot be solved using Gaussian elimination due to zero value in default pivot. It causes division by zero and lead to wrong solution. Augmented matrix and analytical solution are presented below:

$$\left[\begin{array}{c|c} A & b \end{array}\right] = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 2 & 2 & 1 \end{array}\right] \xrightarrow[R_3 = R_3 - R_1]{R_2 = R_2 - R_1} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] \xrightarrow{swap(R_2, R_3)} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right] \quad (2)$$

The results were computed using back substitution.

$$\begin{cases} x_1 & = 1 \\ x_2 & = -1 \\ x_3 & = 1 \end{cases}$$

1.2.2 MATLAB solution

To solve the problem, the Gauss algorithm with partial pivoting was used. The results were similar to analytical solution:

```
A = [1 1 1; 1 1 2; 1 2 2];
b = [1; 2; 1];

% Gaussian elimination without pivoting
R1 = gaussPartialPivot([A b]);
x1 = backSubstitution(R1);

% Gaussian elimination with parital pivoting
R2 = gaussPartialPivot([A b]);
x2 = backSubstitution(R2);

% Gaussian elimination with complete pivoting
[R3 Q] = gaussCompletePivot([A b]);
y = backSubstitution(R3);
x3 = Q * y;
```

Listing 2: main.m - task 2 source code

1.2.3 Conclusions

The Gauss elimination without pivoting cannot be used due to the fact that pivot element equals zero. The easiest solution to this problem was to use Gauss elimination with partial pivoting.

1.3 Task 3

Solve the following system of equations using Gaussian elimination with and without pivoting and observe the impact of round-off error limited to 3 significant digits. Compute the condition number of the system matrix.

$$\begin{cases} 0.0001x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2 \end{cases}$$

The condition number of system matrix: k = 2.6184 was calculated using following formula:

$$k = norm(A) * norm(inv(A)) \tag{3}$$

1.3.1 Analytical solution

Gauss elimination without pivoting:

$$\left[\begin{array}{c|c} A & b \end{array}\right] = \left[\begin{array}{cc|c} 10^{-4} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right] \xrightarrow{R_2 = R_2 - 10^4 R_1} \left[\begin{array}{cc|c} 0 & 1 & 1 \\ 0 & -10^4 & -10^4 \end{array}\right] \tag{4}$$

Round-off:

$$\begin{gathered} 10^{-4} \approx 0 \\ 1 - 10^4 = -0.9999 \cdot 10^4 \approx -10^4 \\ 2 - 10^4 = -0.9998 \cdot 10^4 \approx -10^4 \end{gathered}$$

Results:

$$\begin{cases} x_1 &= 0 \\ x_2 &= 1 \end{cases}$$

Gauss elimination with pivoting:

$$\left[\begin{array}{c|c} A & b \end{array}\right] = \left[\begin{array}{cc|c} 10^{-4} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right] \xrightarrow{Swap(R1,R2)} \left[\begin{array}{cc|c} 1 & 1 & 2 \\ 10^{-4} & 1 & 1 \end{array}\right] \xrightarrow{R_2 - 10^{-4} R_1} \left[\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1 & 1 \end{array}\right] \tag{5}$$

Round-off:

$$\begin{gathered} 1 - 10^{-4} = 0.9999 \approx 1 \\ 2 - 10^{-4} = 0.9998 \approx 1 \end{gathered}$$

Results:

$$\begin{cases} x_1 &= 1 \\ x_2 &= 1 \end{cases}$$

1.3.2 MATLAB solution

```
A = [0.0001 1; 1 1];
b = [1; 2];

% Gaussian elimination without pivoting
R1 = gaussPartialPivot([A b]);
x1 = backSubstitution(R1);

% Gaussian elimination with complete pivoting
[R2 Q] = gaussCompletePivot([A b])
y = backSubstitution(R2)
x2 = Q * y;
```

Listing 3: main.m - task 3 source code

1.3.3 Conclusions

There is a big difference in results obtained with and without pivoting. The higher condition number of system matrix the smaller change of coefficients of the equation can affect the result.

1.4 Task 4

Solve the following system of equations. Observe the affect of condition number to system sensitivity, when perturbing b_2 from 0.067 to 0.066.

$$\begin{cases} 0.835x_1 + 0.667x_2 &= 0.168 \\ 0.333x_1 + 0.266x_2 &= 0.067 \end{cases}$$

1.4.1 Analytical solution

The problem was solved using Gaussian elimination with partial pivoting.

Computations for $b_2 = 0.067$ Augmented matrix:

$$[\ A \mid b\ ] = \left[\begin{array}{cc|c} 0.835 & 0.667 & 0.168 \\ 0.333 & 0.266 & 0.067 \end{array}\right]$$

Result:

$$[\ A \mid b\ ] = \left[\begin{array}{cc|c} 0.8350 & 0.667 & 0.168 \\ 0 & -1.1976e-06 & 1.1976e-06 \end{array}\right], x = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{6}$$

Computations for $b_2 = 0.066$ Augmented matrix:

$$[\ A \mid b\ ] = \left[\begin{array}{cc|c} 0.835 & 0.667 & 0.168 \\ 0.333 & 0.266 & 0.066 \end{array}\right]$$

Result:

$$[\ A \mid b\ ] = \left[\begin{array}{cc|c} 0.835 & 0.667 & 0.168 \\ 0.0658 & -1.1976e-06 & -9.9880e-06 \end{array}\right], x = \begin{bmatrix} -666 \\ 834 \end{bmatrix} \tag{7}$$

1.4.2 MATLAB solution

```
A = [0.835 0.667; 0.333 0.266];
b1 = [0.168; 0.067];

R1 = gaussPartialPivot([A b])
x1 = backSubstitution(R1)

b2 = [0.168; 0.066];

R2 = gaussPartialPivot([A b])
x2 = backSubstitution(R2)
```

Listing 4: main.m - task 4 source code

1.4.3 Conclusions

The condition number of system matrix k = 1.3238+e06 is high, that makes the system sensitive to small perturbations of coefficients.

1.5 Task 5

Find inverse to matrix A to solve equation $AX = I_3$.

$$A = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 3 \\ 4 & 1 & 2 \end{bmatrix}$$

1.5.1 Analytical solution

The Gauss-Jordan Elimination was performed.

$$\left[\ A \mid I\ \right] \to \left[\ I \mid A^{-1}\ \right]$$

$$\left[\ A \mid I\ \right] = \left[\begin{array}{ccc|ccc} 2 & 1 & 2 & 1 & 0 & 0 \\ 1 & 2 & 3 & 0 & 1 & 0 \\ 4 & 1 & 2 & 0 & 0 & 1 \end{array}\right] \xrightarrow[R_3 - 2 \cdot R_1]{R_2 - 1/2 \cdot R_1} \left[\begin{array}{ccc|ccc} 2 & 1 & 2 & 1 & 0 & 0 \\ 0 & 3/2 & 2 & -1/2 & 1 & 0 \\ 0 & -1 & -2 & -2 & 0 & 1 \end{array}\right]$$

$$\xrightarrow[R_1 \cdot 1/2, R_2 \cdot 2/3 R_3 \cdot (-3/2)]{R_3 + 2/3 \cdot R_1} \left[\begin{array}{ccc|ccc} 1 & 1/2 & 1 & 1/2 & 0 & 0 \\ 0 & 1 & 4/3 & -1/3 & 2/3 & 0 \\ 0 & 0 & 1 & -7/2 & -1 & -3/2 \end{array}\right]$$

$$\xrightarrow{R_1 - 1/2 \cdot R_2} \left[\begin{array}{ccc|ccc} 1 & 0 & 1/3 & 2/3 & -1/3 & 0 \\ 0 & 1 & 4/3 & -1/3 & 2/3 & 0 \\ 0 & 0 & 1 & -7/2 & -1 & -3/2 \end{array}\right] \xrightarrow[R_2 - 4/3 \cdot R_3]{R_1 - 1/3 \cdot R_3} \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & -1/2 & 0 & 1/2 \\ 0 & 1 & 0 & -5 & 2 & 2 \\ 0 & 0 & 1 & 7/2 & -1 & -3/2 \end{array}\right] \tag{8}$$

$$A^{-1} = \begin{bmatrix} -1/2 & 0 & 1/2 \\ -5 & 2 & 2 \\ 7/2 & -1 & -3/2 \end{bmatrix}$$

1.5.2 MATLAB solution

The calculations were performed using following code:

```
51 A = [2 1 2; 1 2 3; 4 1 2];
52 inv_A = inv(A)
53 I = A * inv_A
```

Listing 5: main.m - task 5 source code

1.5.3 Conclusions

The results obtained from analytical solution and Matlab solution were similar.

1.6 Task 6

Apply the LU factorization to the matrix A, then calculate its determinant using resulting matrices and solve equation $Ax = b$ for $b = [1...1]^T$.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ -1 & 1 & 2 & 1 \\ 0 & 2 & 1 & 3 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

1.6.1 Analytical solution

The LU decomposition was performed, the L and U matrices:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0.6667 & 1 & 0 \\ 0 & 0 & -0.4286 & 1 \end{bmatrix}, U = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 3 & 5 & 5 \\ 0 & 0 & -2.3333 & -0.3333 \\ 0 & 0 & 0 & 0.8571 \end{bmatrix} \tag{9}$$

The determinant of matrix A can be calculated using following formula:

$$det(A) = det(L) * det(U) \xrightarrow{det(L)=1} det(A) = det(U) = -6 \tag{10}$$

The next step was to solve the system of linear equations.

$$\begin{cases} Lz = b \\ Ux = z \end{cases}$$

The vector z was obtained by performing forward substitution on matrix L and vector b. The vector of solutions x was obtained by performing back substitution on matrix U and vector z.

$$z = \begin{bmatrix} 1 \\ 2 \\ -0.3333 \\ 0.8571 \end{bmatrix} x = \begin{bmatrix} -1 \\ -1 \\ 9.5162e-17 \\ 1 \end{bmatrix}$$

1.6.2 MATLAB solution

```
A = [1 2 3 4; -1 1 2 1; 0 2 1 3; 0 0 1 1];
b = [1; 1; 1; 1];

[L U] = LUFactor(A)
y = forwardSubstitution([L b]);
x = backSubstitution([U y])

det_A = det(L) * det(U)
det(A)
```

Listing 6: main.m - task 6 source code

1.6.3 Conclusions

The L and U matrices calculated using coded algorithms were the same as from built-in Matlab function. The vectors of solutions was consistent with the result obtained form x=A\b function.

1.7 Task 7

Let $A = [a_{ij}] \in R^{NxN}$ with $a_{ij} = \frac{1}{i+j-1}$ (Hilbert matrix). For $N = 5$, perform the LU factorization of the matrix A and compute its determinant.

1.7.1 Analytical solution

The Hilbert matrix was generated and LU decomposition was performed.

$$A = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 & 0 \\ 0.3333 & 1 & 1 & 0 & 0 \\ 0.25 & 0.9 & 0.5 & 1 & 0 \\ 0.2 & 0.8 & 1.7143 & 2 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & 0.5 & 0.3333 & 0.25 & 0.2 \\ 0 & 0.0833 & 0.0833 & 0.075 & 0.0667 \\ 0 & 0 & 0.0056 & 0.0083 & 0.0095 \\ 0 & 0 & 0 & 3.5714e-04 & 7.1429e-04 \\ 0 & 0 & 0 & 0 & 2.2676e-05 \end{bmatrix}$$

The determinant was calculated using following formula:

$$\det(A) = \det(U) = 3.7493e-12 \tag{11}$$

1.7.2 MATLAB solution

```
67 A = hilb(5);
68
69 [L U] = LUFactor(A)
70 det_A = det(L) * det(U)
71 det(A)
```

Listing 7: main.m - task 7 source code

1.7.3 Conclusions

The matrices L and U are different than matrices obtained by using built-in LU decomposition. Despite this after multiplying L*U the matrix A was obtained.

1.8 Task 8

Check whether the symmetric matrix A is positive-definite. If so, apply the Choleksy factorization. Then, compute its inverse.

$$A = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

1.8.1 Analytical solution

To check if matrix is symmetric function issymetric(A) was used and to check if matrix is positive-definite coded algorith isPosDef(A) was used. Then the Cholesky decomposition was preformed.

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

To compute matrix inverse following formula was used:

$$A^{-1} = (L^{-1})^T (L^{-1}) \tag{12}$$

$$A^{-1} = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

1.8.2 MATLAB solution

```
74  A = [1 -1 0 0; -1 2 -1 0; 0 -1 2 -1; 0 0 -1 2];
75
76  if(issymmetric(A))
77      if(isPosDef(A))
78          [L] = Cholesky_factor(A);
79      else
80          disp('matrix is not positive-definite')
81      end
82  else
83      disp('matrix is not symetric')
84  end
85  inverse = (inv(L)')*(inv(L))
```

Listing 8: main.m - task 8 source code

1.9 Task 9

Let A be the Pascal matrix of order 100. Check whether the matrix is positive definite. If so, apply the Cholesky factorization and give interpretation of the obtained factor.

1.9.1 MATLAB solution

The matrix is not positive definite. It was checked using coded algorith isPosDef(A). The Cholesky factorization couldn't be performed.

```
89 A = pascal(100);
90
91 if(isPosDef(A))
92     [L] = Cholesky_factor(A);
93 else
94     disp('matrix is not positive-definite')
95 end
```

Listing 9: main.m - task 9 source code

1.9.2 Conclusions

The Cholesky decomposition couldn't be performed for non positive definite or barely positive definite matrices. The problem is round-off error, the values of elements of matrix are very small. The eigenvalues and eigenvecors of that matrix are close to those of matrix A but are not exacly the same.

1.10 Task 10

Transform matrix A to the RREF, determine its rank and identify the columns corresponding to the basic and free variables.

$$A = \begin{bmatrix} 1 & 2 & 2 & 3 & 1 \\ 2 & 4 & 4 & 6 & 2 \\ 3 & 6 & 6 & 9 & 6 \\ 1 & 2 & 4 & 5 & 3 \end{bmatrix}$$

1.10.1 Analytical solution

$$A = \begin{bmatrix} 1 & 2 & 2 & 3 & 1 \\ 2 & 4 & 4 & 6 & 2 \\ 3 & 6 & 6 & 9 & 6 \\ 1 & 2 & 4 & 5 & 3 \end{bmatrix} \xrightarrow[R_4 - R_1]{R_2 - 2 \cdot R_1, R_3 - 3 \cdot R_1} \begin{bmatrix} 1 & 2 & 2 & 3 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 2 & 2 & 2 \end{bmatrix} \xrightarrow[R_4/2]{R_3/2} \begin{bmatrix} 1 & 2 & 2 & 3 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$\xrightarrow{Swap(R_2, R_4)} \begin{bmatrix} 1 & 2 & 2 & 3 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow[R_2 - R_3]{R_1 - R_3} \begin{bmatrix} 1 & 2 & 2 & 3 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{R_1 - 2 \cdot R_2} \begin{bmatrix} 1 & 2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{13}$$

Rank of matrix A = 3, as there are 3 pivotal entries in RREF of matrix A. The basic columns are 1st, 3rd and 5th. Free columns are 2nd and 4th.

1.10.2 MATLAB solution

```
98  A = [1 2 2 3 1;
99       2 4 4 6 2;
100      3 6 6 9 6;
101      1 2 4 5 3];
102
103 X = RREF(A)
104 rank(A)
```

Listing 10: main.m - task 10 source code

1.10.3 Conclusions

The results obtained analytically and from Matlab are the same.

1.11 Task 11

Compute the LU factorization for matrix A. Determine set of basic and free variables and find homogeneous solution to $Ax = 0$ and rank of A.

$$A = \begin{bmatrix} 1 & 3 & 3 & 2 \\ 2 & 6 & 9 & 5 \\ -1 & -3 & 3 & 0 \end{bmatrix}$$

1.11.1 MATLAB solution

The results were obtained using built in MATLAB function [L U] = lu(A). The matrix A is not square and the coded algorithm couldn't cope with it.

```
A = [1 3 3 2; 2 6 9 5; -1 -3 3 0];
b = [0; 0; 0];

[L U] = lu(A)
A \ b

rref(A)
rank(A)
```

Listing 11: main.m - task 11 source code

Obtained matrices:

$$L = \begin{bmatrix} 0.5 & 1 & 0 \\ 1 & 0 & 0 \\ -0.5 & 0 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 2 & 6 & 9 & 5 \\ 0 & 0 & -1.5 & -0.5 \\ 0 & 0 & 7.5 & 2.5 \end{bmatrix}$$

$$RREF = \begin{bmatrix} 1 & 3 & 0 & 1 \\ 0 & 0 & 1 & -0.5 \\ 0 & 0 & 7.5 & 2.5 \end{bmatrix}$$

1.11.2 Conclusions

Rank of matrix = 2, basic variables are x_1 and x_3. x_2 and x_4 are free variables.

1.12 Task 13

Let $Ax = b$, where $A = I_N \otimes C^T C$, the symbol $\otimes$ denotes the Kronecker product, $I_N \in R^{NxN}$ is an identity matrix, $C \in R^{MxM}$ is a random matrix with a uniform distribution, $M = 100$, and $N = 50$, and $x \in N(0, I_{MN})$. Find the direct method that solves the above system of linear equations with the lowest computational cost. Estimate the cost with a roughly calculated number of flops and with the elapsed time.

1.12.1 MATLAB solution

The code snippet of setting the parameters and Gauss elimination method is presented below. The code is similar for other functions.

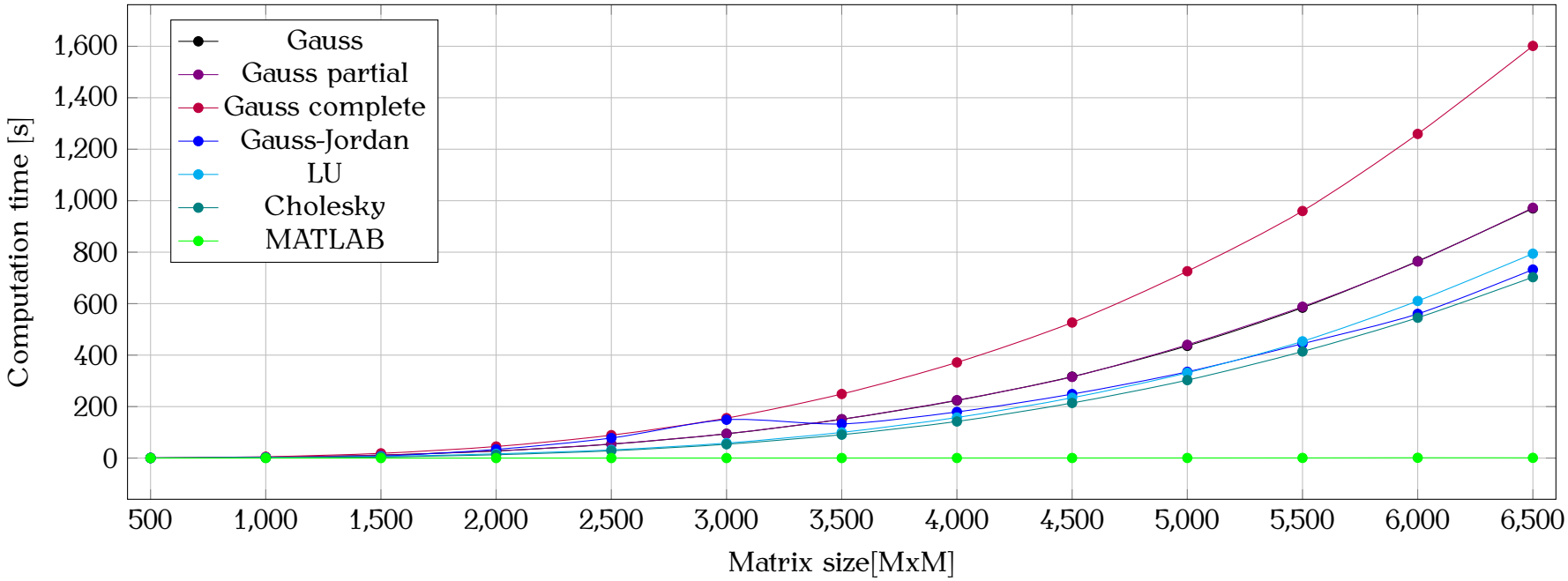
```
Methods = ["Gauss"; "GaussPartialPivot"; "GaussCompletePivot"; "GaussJordan";
            "BuiltIn"; "LU"; "Cholesky"];

M = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]';
N = 50;
[m,~]=size(M);
MatrixSize =0;

ResultsG = zeros(m,4);
% <code omitted>
Time = 0;
rng('default');

for i = 1:m
        x = rand(M(i)*N,1);
        C = rand(M(i),M(i));
        A = kron(eye(N), C'*C);
        b = A*x;
        Xest = zeros(size(x));

    if(ismember("Gauss",Methods))
        disp('gauss'); %gaussNoPivot
        disp(M(i));
        tic;
        Xe = gaussNoPivot([A,b]);
        Xest = backSubstitution(Xe);
        Time = toc;

        e = norm(Xest-x)/norm(x);

        ResultsG(i,1) = M(i); % M value
        ResultsG(i,2) = M(i)*N;  % Matrix size
        ResultsG(i,3) = e;    % error
        ResultsG(i,4) = Time; % Time
    end
    % <code omitted>
```

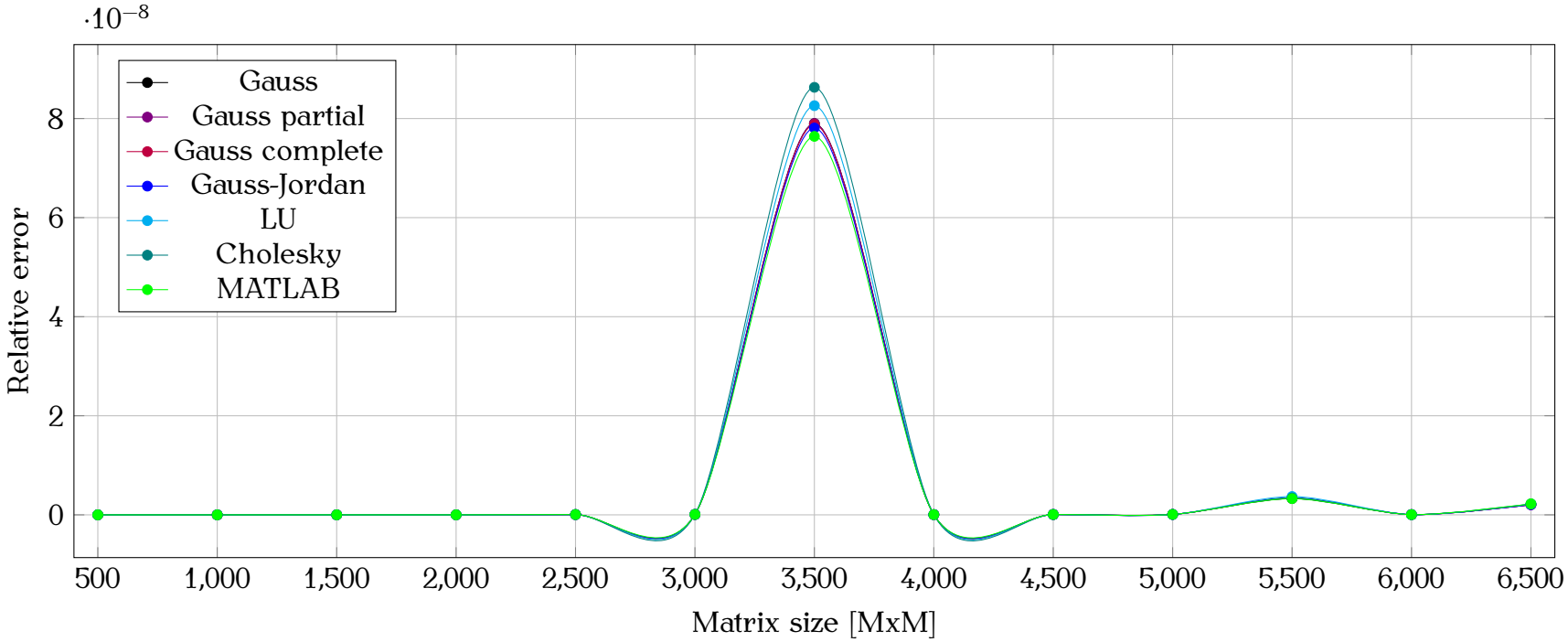
Listing 12: main.m - task 13 source code fragment

Size [MxM]	Gauss δ_{meth}	Gauss T_{tot} [s]	Gauss partial pivoting δ_{meth}	Gauss partial pivoting T_{tot} [s]	Gauss complete pivoting δ_{meth}	Gauss complete pivoting T_{tot} [s]	Gauss-Jordan δ_{meth}	Gauss-Jordan T_{tot} [s]
500	$7.97 \cdot 10^{-14}$	0.39	$6.92 \cdot 10^{-14}$	0.38	$7.6 \cdot 10^{-14}$	0.55	$7.04 \cdot 10^{-14}$	0.14
1,000	$3.51 \cdot 10^{-13}$	3.36	$3.51 \cdot 10^{-13}$	3.4	$3.64 \cdot 10^{-13}$	5.11	$3.37 \cdot 10^{-13}$	1.54
1,500	$5.34 \cdot 10^{-12}$	11.6	$5.34 \cdot 10^{-12}$	11.69	$5.66 \cdot 10^{-12}$	18.43	$5.41 \cdot 10^{-12}$	7.7
2,000	$4.12 \cdot 10^{-12}$	27.59	$4.12 \cdot 10^{-12}$	27.84	$4.33 \cdot 10^{-12}$	44.99	$4.15 \cdot 10^{-12}$	33.81
2,500	$4.81 \cdot 10^{-11}$	54.07	$4.81 \cdot 10^{-11}$	54.66	$4.85 \cdot 10^{-11}$	89.01	$4.92 \cdot 10^{-11}$	77.9
3,000	$8.12 \cdot 10^{-11}$	94.12	$8.12 \cdot 10^{-11}$	94.6	$7.82 \cdot 10^{-11}$	155.23	$7.97 \cdot 10^{-11}$	148.99
3,500	$7.9 \cdot 10^{-8}$	150.52	$7.9 \cdot 10^{-8}$	150.98	$7.88 \cdot 10^{-8}$	248.93	$7.81 \cdot 10^{-8}$	132.64
4,000	$4.49 \cdot 10^{-11}$	223.66	$4.49 \cdot 10^{-11}$	224.85	$4.29 \cdot 10^{-11}$	371.69	$4.49 \cdot 10^{-11}$	179.35
4,500	$7.23 \cdot 10^{-11}$	316.41	$7.23 \cdot 10^{-11}$	315.33	$7.22 \cdot 10^{-11}$	526.61	$7.25 \cdot 10^{-11}$	248.61
5,000	$8.41 \cdot 10^{-11}$	436	$8.41 \cdot 10^{-11}$	440.03	$8.37 \cdot 10^{-11}$	725.99	$8.46 \cdot 10^{-11}$	335.14
5,500	$3.31 \cdot 10^{-9}$	584.37	$3.31 \cdot 10^{-9}$	588.07	$3.42 \cdot 10^{-9}$	959.95	$3.42 \cdot 10^{-9}$	444.42
6,000	$4.46 \cdot 10^{-11}$	765.23	$4.46 \cdot 10^{-11}$	763.74	$4.44 \cdot 10^{-11}$	1,259.46	$4.99 \cdot 10^{-11}$	560.3
6,500	$2.06 \cdot 10^{-9}$	970.08	$2.06 \cdot 10^{-9}$	972.54	$2.06 \cdot 10^{-9}$	1,601.41	$2.06 \cdot 10^{-9}$	732.46

Size [MxM]	MATLAB function δ_{meth}	MATLAB function T_{tot} [s]	LU δ_{meth}	LU T_{tot} [s]	Cholesky δ_{meth}	Cholesky T_{tot} [s]
500	$8.94 \cdot 10^{-14}$	0	$9.14 \cdot 10^{-14}$	0.52	$9.96 \cdot 10^{-14}$	0.27
1,000	$4.15 \cdot 10^{-13}$	0.01	$4.01 \cdot 10^{-13}$	2.46	$3.72 \cdot 10^{-13}$	1.37
1,500	$5.86 \cdot 10^{-12}$	0.03	$6.57 \cdot 10^{-12}$	6.41	$6.02 \cdot 10^{-12}$	4.17
2,000	$5.01 \cdot 10^{-12}$	0.05	$4.78 \cdot 10^{-12}$	16.54	$4.71 \cdot 10^{-12}$	12.93
2,500	$5.8 \cdot 10^{-11}$	0.09	$6.17 \cdot 10^{-11}$	31.74	$6.91 \cdot 10^{-11}$	28.24
3,000	$8.07 \cdot 10^{-11}$	0.12	$9.42 \cdot 10^{-11}$	58.2	$8.16 \cdot 10^{-11}$	53.61
3,500	$7.64 \cdot 10^{-8}$	0.19	$8.26 \cdot 10^{-8}$	99.81	$8.63 \cdot 10^{-8}$	90.85
4,000	$4.64 \cdot 10^{-11}$	0.29	$4.47 \cdot 10^{-11}$	157.63	$5.4 \cdot 10^{-11}$	142.25
4,500	$7.53 \cdot 10^{-11}$	0.34	$7.95 \cdot 10^{-11}$	234.3	$7.43 \cdot 10^{-11}$	214.06
5,000	$9.03 \cdot 10^{-11}$	0.44	$8.26 \cdot 10^{-11}$	331.32	$8.72 \cdot 10^{-11}$	302.85
5,500	$3.27 \cdot 10^{-9}$	0.6	$3.7 \cdot 10^{-9}$	453.57	$3.4 \cdot 10^{-9}$	414.11
6,000	$4.52 \cdot 10^{-11}$	1.14	$4.9 \cdot 10^{-11}$	610.52	$4.98 \cdot 10^{-11}$	545.35
6,500	$2.23 \cdot 10^{-9}$	0.89	$2.17 \cdot 10^{-9}$	794.18	$2.22 \cdot 10^{-9}$	702.78

Table 1: Task 13 - comparison of direct methods on matrix MxM


Gauss
Gauss partial
Gauss complete
Gauss-Jordan
LU
Cholesky
MATLAB
Computation time [s]
0
200
400
600
800
1,000
1,200
1,400
1,600
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
4,500
5,000
5,500
6,000
6,500
Matrix size[MxM]


$\cdot 10^{-8}$
Gauss
Gauss partial
Gauss complete
Gauss-Jordan
LU
Cholesky
MATLAB
Relative error
0
2
4
6
8
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
4,500
5,000
5,500
6,000
6,500
Matrix size [MxM]

1.12.2 Conclusions

The data proves, that MATLAB built-in function is the fastest. Gauss elimination with complete pivoting is the slowest due to number of pivoting operations performed to obtain the solution. Random matrix generated for matrix of size 3500x3500 was unfavorable for direct methods. The error of the methods was of the same order.

1.13 Task 14

Generate 10x10 Hilbert matrix and normally distributed x vector. Solve $Ax = b$ with built in MATLAB algorithm. Introduce small change to A matrix and observe the results.

1.13.1 MATLAB solution

The problem was solved using following solution:

```
264 A = hilb(10);
265 x = normrnd(0,1,[10,1]);
266 b = A*x;
267 k = norm(A) * norm(inv(A));
268
269 X_estimated = A \ b;
270 E_estimated = norm(X_estimated-x)/norm(x);
271
272 A(5,5) = A(5,5)+0.001; % small change of matrix A
273
274 X_perturbed = A \ b;
275 E_perturbed = norm(X_perturbed-x) / norm(x);
```

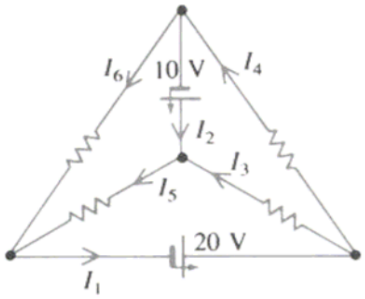
Listing 13: main.m - task 14 source code

1.13.2 Conclusions

The error in first solution was 7.3414e-05. After small change in matrix A the error incresed to 1.1680. The condition number of system matrix k = 1.6024e+13 it extremely high, due to that fact the system is sensitive to small perturbations.

1.14 Task 15

Find the current in the circuit, assuming all resistances are 10Ω.



1.14.1 Analytical solution

Branch-current circuit analysis was applied to find the currents in the ΔY circuit presented in the figure above. The reference point is set for the bottom left node. After applying KVL and KCL following system of linear equations was obtained:

$$\begin{cases} 10I_5 - 10I_6 &= 10 \\ 10I_3 - 10I_4 &= -10 \\ 10I_3 + 10I_5 &= 20 \\ I_1 - I_3 - I_4 &= 0 \\ I_2 + I_3 - I_5 &= 0 \\ I_2 - I_4 + I_6 &= 0 \end{cases}$$

Applying Gaussian eliminations to the system results with the following currents:

$$\begin{cases} I_1 &= 2 \\ I_2 &= 1 \\ I_3 &= 0.5 \\ I_4 &= 1.5 \\ I_5 &= 1.5 \\ I_6 &= 0.5 \end{cases}$$

1.14.2 MATLAB solution

```
278 A = [0 0 0 0 10 -10; 0 0 10 -10 0 0; 0 0 10 0 10 0; ...
279     1 0 -1 -1 0 0; 0 1 1 0 -1 0;  0 1 0 -1 0 1];
280 b = [10; -10; 20; 0; 0; 0];
281
282 A \ b
```

Listing 14: main.m - task 15 source code

2 MATLAB functions

2.1 Gaussian elimination without pivoting

```
function [R] = gaussNoPivot(A)
    [m, ~] = size(A);

    for k = 1:(m-1)
        if A(k, k) ~= 0
            i = k+1:m;
            A(i, k) = A(i, k) / A(k, k);
            A(i, :) = A(i, :) - A(i, k) * A(k, :);
        end
    end
    R = A;
```

Listing 15: gaussNoPivot.m

2.2 Gaussian elimination using partial pivoting

```
function [R] = gaussPartialPivot(A)
    [m, ~] = size(A);
    % record perumations of rows
    p = 1:m;

    for k = 1:(m-1)
        % find max column value in submatrix of A
        [~, max_row] = max(abs(A(p(k:m), k)));

        % transform relative row index to global row index of A
        max_row = max_row + k-1;

        % permute rows
        p([k max_row]) = p([max_row k]);

        % perform gaussian elimination with the greatest pivot in column
        if A(k, k) ~= 0
            i = (k+1):m;
            A(p(i), k) = A(p(i), k) / A(p(k), k);
            A(p(i), :) = A(p(i), :) - A(p(i), k) * A(p(k), :);
        end
    end
    R = A(p, :);
```

Listing 16: gaussPartialPivot.m

2.3 Gaussian elimination using complete pivoting

```
function [R, Q] = gaussCompletePivot(A)
    [m, n] = size(A);
    % record perumations of rows (p) and cols (q)
    p = 1:m; q = 1:n;

    for k = 1:(m-1)
        % find coords of max value (max_row, max_col) in submatrix of A
        [max_row_vals, row_indices] = max(abs(A(p(k:m), q(k:m))));
        [~, max_col] = max(max_row_vals);
        max_row = row_indices(max_col);

        % transform submatrix relative coords to global coords of A
        max_row = max_row + k-1;
        max_col = max_col + k-1;

        % permute rows and columns
        p([k max_row]) = p([max_row k]);
        q([k max_col]) = q([max_col k]);

        % perform gaussian elimination with the greatest pivot
        if A(k, k) ~= 0
            i = (k+1):m;
            A(p(i), q(k)) = A(p(i), q(k)) / A(p(k), q(k));
            A(p(i), :) = A(p(i), :) - A(p(i), q(k)) * A(p(k), :);
        end
    end
    R = A(p, q);
    Q = eye(m); Q = Q(:, q(1:m));
```

Listing 17: gaussCompletePivot.m

2.4 Backward substitution

```
function [x] = backSubstitution(A)
    [m, n] = size(A);
    x = zeros(m, 1);

    for k = m:-1:1
        coefs_sum = A(k, k:m) * x(k:m);
        x(k) = (A(k, n) - coefs_sum) / A(k, k);
    end
```

Listing 18: backSubstitution.m

2.5 Forward substitution

```
function [x] = forwardSubstitution(A)
    [m, n] = size(A);
    x = zeros(m, 1);

    for k = 1:m
        coefs_sum = A(k, 1:k) * x(1:k);
        x(k) = (A(k, n) - coefs_sum) / A(k, k);
    end
```

Listing 19: forwardSubstitution.m

2.6 LU factorization

```
function [L, U] = LUFactor(A)
    [m, ~] = size(A);
    L = zeros(m); U = eye(m);

    % perform Crout's algorithm
    for k = 1:m
        for j = k:m
            U(k, j) = A(k, j) - dot(L(k, 1:(k-1)), U(1:(k-1), j));
        end
        for i = k:m
            L(i, k) = (A(i, k) - dot(L(i, 1:(k-1)), U(1:(k-1), k))) / U(k, k);
        end
    end
```

Listing 20: LUFactor.m

2.7 Cholesky factorization

```
function [L] = Cholesky_factor(A)
    [rows, ~] = size(A); L = zeros(rows);

    % perform Crout's algorithm
    for k = 1:rows
        L(k, k) = sqrt(A(k, k) - sum (L(k, 1:(k-1)).^2));
        for j = (k+1):rows
            L(j, k) = (1/L(k, k)) * (A(j, k) - sum((L(j, 1:(k-1)).*L(k, 1:(k-1)))));
        end
    end
```

Listing 21: Cholesky_factor.m

2.8 QR factorization

```
function [Q R] = QRFactor(A)

[m, n] = size(A);

if ~(m >= n)
   error("m has to be greater or equal n!")
end

for j = 1:n
    [v, beta] = householder(A(j:m, j));
    A(j:m, j:n) = (eye(m-j+1) - beta*(v*v.')) * A(j:m, j:n);
    if j < m
        A(j+1:m, j) = v(2:m-j+1);
    end
end
R = triu(A);
H = tril(A, -1)

end
```

Listing 22: QRFactor.m

2.9 Checking if matrix is positive-definite

```
function [posDef] = isPosDef(A)
    [rows, ~] = size(A);
    L = zeros(rows);
    Determinants= zeros(1,rows);
    posDef = 0;


    for i = 1:rows
            Determinants(i) = det(A(1:i,1:i)); %i upper left determinants
        end

        if (all(Determinants>0)) %chceck if all determinants are positive
            posDef=1;
        end
end
```

Listing 23: isPosDef.m

2.10 Gauss-Jordan elimination

```
function [RR] = gaussJordan(A)
    [m, ~] = size(A);

    for k = 1:m
        % make k'th row pivot equal 1
        A(k, :) = A(k, :) / A(k, k);
        current_row = A(k, :);

        for i = 1:m
            if i ~= k
                A(i,:) = A(i, :) - A(i, k) * current_row;
            end
        end
    end
    RR = A;
```

Listing 24: gaussJordan.m

2.11 Reduced-row echelon form computation

```
function [RR] = RREF(A)
    [m, n] = size(A);
    j = 0;

    for k = 1:m
        j = j + 1;
        if j > n
            break
        end

        % make left-most coefficient 1 (pivot)
        row = A(k, :);
        if row(k) == 0
            j = j+1;
        end
        row = row/row(j);
        A(k, :) = row;

        for i = 1 : m
            if i ~= k
                A(i, :) = A(i, :)-(A(i, j)) * row;
            end
        end

        for i = k + 1 : m
           A(i:end, k+1:end);       % submatrix of A
           A(i:end, k+1);           % left-most column
           if ~any(A(i:end, k+1))   % if the left-most column has only zeros check the next one
               k = k+1;
           end
           A(i:end, k+1:end);
           if A(i, k+1) == 0
               non_zero_row = find(A(i:end, k+1), 1);
               if isempty(non_zero_row)
                   continue
               end
               A([i, i+non_zero_row-1], :) = deal(A([i+non_zero_row-1, i], :));
           end
        end
    end
    RR = A;
```

Listing 25: RREF.m

Literature

[GVL96] G. H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.

[GVL13] G.H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, fourth edition, 2013.