Warsaw, 09.IV.2019

Numerical Methods (ENUME)

Assignment A: Solving linear algebraic equations

Final Report

Name: Jan Radzimiński

Index number: 293052

Supervisor: Andrzej Miękina

Table of contents:

0. General Remarks
1. Task 1: Generation of Matrices
    1. Introduction
    2. Tests
2. Task 2: Determinants and Condition Numbers
    1. Introduction
    2. Smallest α_N
    3. Graph of determinants
    4. Graph of condition numbers
3. Task 3: Inverse of Matrices with usage of LU and LLT matrix factorization
    1. Introduction
    2. Algorithm
    3. Tests
4. Task 4: Estimates of inverse of matrices generated by function made in Task 1
    1. Introduction
5. Task 5: Indicators of uncertainty of estimated inverses
    1. Introduction
    2. Norms
    3. Errors
    4. Graphs
    5. Graphs conclusions
    6. Comparison of matrices inverses made with built in function, with these computed in Task 4

0. General Remarks

This project was implemented using Matlab environment. The Matlab script containing solutions to all tasks is called ‘’AssA_JR.m”.

After presenting solution to a given task, script stops (by function “pause”) and it waits for any key to be pressed to move on to the next task.

All plots are made by “semilogy()” function, which makes them much more readable and clear.

Also, after every task, script calls “clear”, “close all”, “hold off” and “clc” functions, so tasks don’t interfere with each other and each task is more readable.

1. Task 1 – Generation of matrices

1.1. Introduction

The **first task** was about writing Matlab function generating such matrices:

$$\mathbf{A}_{N,x} = \begin{bmatrix} x^2 & \frac{2x}{3} & -\frac{2x}{3} & \frac{2x}{3} & \cdots & \frac{2x}{(-1)^{N-1}\cdot 3} & \frac{2x}{(-1)^{N}\cdot 3} \\ \frac{2x}{3} & \frac{8}{9} & -\frac{8}{9} & \frac{8}{9} & \cdots & \frac{8}{(-1)^{N-1}\cdot 9} & \frac{8}{(-1)^{N}\cdot 9} \\ -\frac{2x}{3} & -\frac{8}{9} & \frac{12}{9} & -\frac{12}{9} & \cdots & \frac{12}{(-1)^{N-4}\cdot 9} & \frac{12}{(-1)^{N-3}\cdot 9} \\ \frac{2x}{3} & \frac{8}{9} & -\frac{12}{9} & \frac{16}{9} & \cdots & \frac{16}{(-1)^{N-5}\cdot 9} & \frac{16}{(-1)^{N-4}\cdot 9} \\ -\frac{2x}{3} & -\frac{8}{9} & \frac{12}{9} & -\frac{16}{9} & \cdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \frac{(N-1)\cdot 4}{9} & -\frac{(N-1)\cdot 4}{9} \\ \frac{2x}{(-1)^{N}\cdot 3} & \frac{8}{(-1)^{N}\cdot 9} & \frac{12}{(-1)^{N-3}\cdot 9} & \frac{16}{(-1)^{N-4}\cdot 9} & \cdots & -\frac{(N-1)\cdot 4}{9} & \frac{N\cdot 4}{9} \end{bmatrix}$$

for N, x given as function arguments (where N should be a natural number). For any N and x, such matrix is a positive definite matrix (which will be important in further tasks).

In the script this function is called Generate(N, x) and it returns matrix A.

1.2. Tests

To test correctness of implemented function, script generates 5 low dimensional matrices, with different values of x. Matrices are shown mostly in fractional format, so they are easier to compare with matrix example shown in task 1. Here are two examples:

```
For N=3, x=-2 generating matrix C:

C =

       4          -4/3          4/3
    -4/3           8/9         -8/9
     4/3          -8/9          4/3
```

```
For N=5, x=1, generating matrix E:

E =

       1          2/3         -2/3          2/3         -2/3
     2/3          8/9         -8/9          8/9         -8/9
    -2/3         -8/9          4/3         -4/3          4/3
     2/3          8/9         -4/3         16/9        -16/9
    -2/3         -8/9          4/3        -16/9         20/9
```

It is clearly visible that in all cases generated matrices are in the format presented in task 1, which means that function works properly.

2. Task 2 – Determinants and condition numbers

2.1. Introduction

In the **second task**, main goal was to find such number α_N, for which matrix generated for N={2, 10, 20} and x = ln(α) has determinant equals to 0. Then, this result should be presented on the graphs of determinant and condition number, for arguments very close to given α.

2.2. Smallest α_N

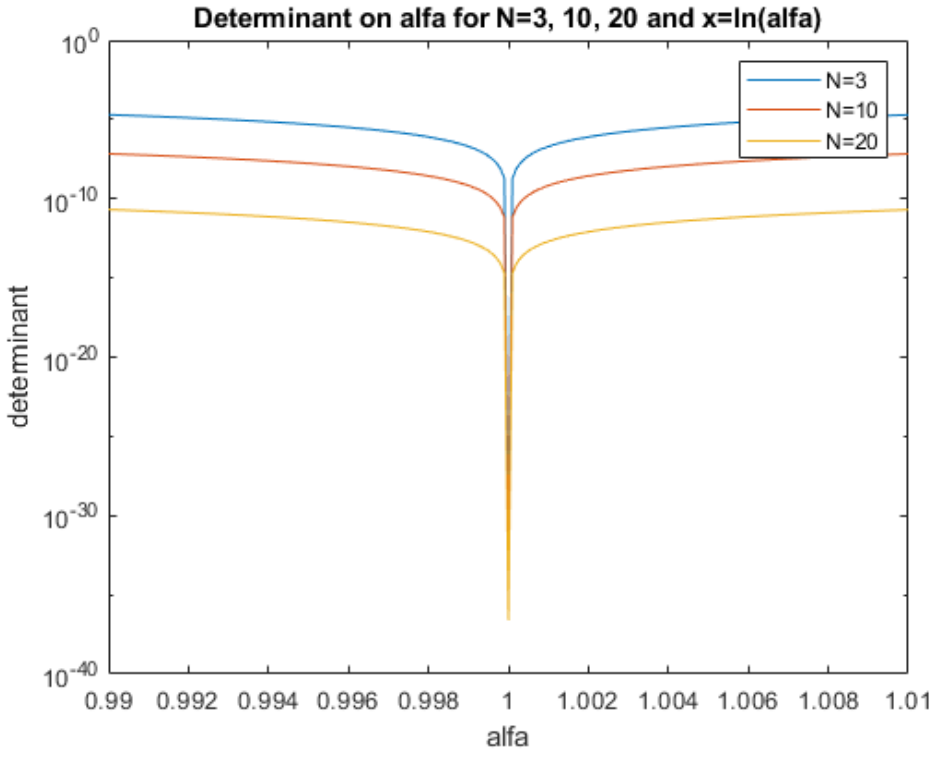
First of all, to find such α, one should determine when determinant of matrices such in task 1 is 0. Since only first row and column of these matrices depend on α, determinant of given matrices is 0 **only when** all **values in first row and column are equal to 0** (for any value of N). Such situation appears only when x is 0 (since all arguments in first row and column are multiplied by x), therefore we must find α for which x is 0. Obviously, natural logarithm is equal to 0 only when its argument is equal to 1, so finally, **our α_N (for any value of N) must be equal to 1.**

This conclusion may be also confirmed by results of "SmallesAlpha" function in my Matlab script – it finds alpha with given approximation (of determinant) and precision (both can be set in the script). For step and precision equal to 1.0e-5 it finds α = 0.9929 – value very close to 1 (for smaller step and precision value will be even closer to 1 – more precise).

2.3. Graph of determinants

Next part of this task consisted of drawing dependence of determinants of matrix generated by function from Task 1 for N={3, 10, 20} and x = ln(α), for α belonging to interval [α_N – 0.01, α_N + 0.01]. Graph was generated with step of α equal to 0.001. Graph of curves for all N's:

Figure (2.1):



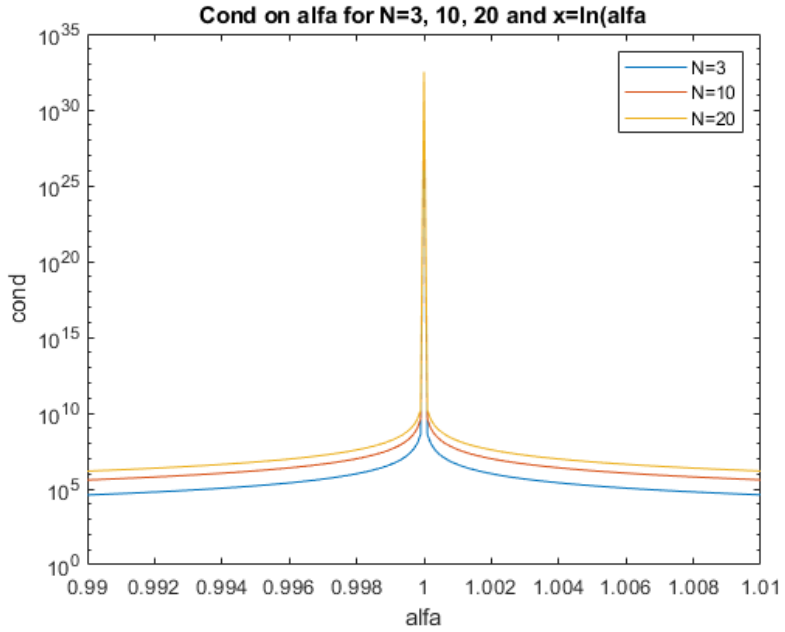
From graph (*Figure (2.1)*) can be clearly seen that the closer α gets to 1, the smaller value of determinant is (for smaller N values of determinants are slightly bigger since for bigger N, there are more bigger values in the matrix, so values of determinants are smaller). At α=1 we see big drop of all curves, since this is the value for which our determinants are equal to 0.

2.4. Graph of condition number

Last thing in this task was to draw dependence of condition number on α. Similarly as in previous case, graph was drown for N={3, 10, 20} and x = ln(α), for α belonging to interval [α_N – 0.01, α_N + 0.01], step of α equal to 0.001.

Graph consisting of curves for all N's:

Figure (2.2):



Generally speaking, condition number of a matrix is the ratio of the largest singular value of that matrix to the smallest singular value. So, it tells us how much the singular values of the matrix differs. On this graph (*Figure (2.2)*) we can see, that for values relatively far from α=1 ratio is rather small – values does not differ that much, but the condition number values get smaller, if α gets closer to 1. For values really close to α=1– condition number rapidly grows. That's because for these α's our smallest singular values come closer and closer to 0 (since x comes closer to 0) so our ratio goes to infinity (which at point 1 on our graph is very large number > 10^{30}).

3. Task 3 – Inverse of matrices

3.1. Introduction

The main problem of task 3 was to create function that calculates inverses of matrices, with usage of LU and LL^T factorizations.

3.2. Algorithm

With both factorizations we decompose matrix A into two matrices which first one has zeroes in "upper triangle" of matrix and second one in "lower triangle" of matrix. To calculate inverses of matrices we can use identity:

$$A \cdot A^{-1} = I_N$$

After decomposition of A we have

$$L \cdot U \cdot A^{-1} = I_N$$

Then, we can make substitution:

$$A^{-1} = Y/U$$

$$Y = I_N/L$$

After such substitution, by treating each column of Y and I_N as vectors, we can apply forward substitution to find matrix Y, and then similarly we can apply backward substitution to find matrix A^{-1}.

This is how both functions in my Matlab script works (with L^T instead U in LL^T case). They are called "InverseOfMatrixLU()" and "InverseOfMatrixLLT()". Both functions take one matrix as input argument and return one matrix as output argument. First function uses built in "lu()" function, with P (pivot) argument, with which final result is multiplied, since some columns are swapped. The second function uses built in function "chol()" which returns L matrix (then L^T matrix is just transpose of L). Also both function uses function "eye()" for generating identity matrices.

3.3. Tests

To test correctness of these functions, they are applied to three random positive definite matrices A, B and C with sizes respectively 2, 3 and 4. Script applies functions "InverseOfMatrixLU" on them and saves results to Ainv, Binv and Cinv. Then it compares multiplications A*Ainv, B*Binv and C*Cinv to identity matrices, with corresponding sizes and prints result matrices in command line (they should all be files with ones). Then it does the same with "InverseOfMatrixLLT", but it saves results to A2inv etc. **Note**: Script comperes **rounded** results of multiplication of matrices and their inverses, since there are very small differences between result of it and identity matrix (that comes mainly from that often calculating exact value is impossible E.g. 2/3) and Matlab calculates estimate of it) and otherwise result wouldn't be correct (answer matrices wouldn't be filled with ones since values wouldn't be exactly identical).

4. Task 4 – Estimates of inverse of matrices generated by function made in Task 1

4.1. Introduction

The goal of Task 4 was to apply functions made in task 3 for matrices generated by function from task 1 (since any matrix generated by this function is positive definite they will work properly) for N = {3, 10, 20} and x = $(2^k)/300$ (for k = {0, 1, … , 21}). Script generates these matrices and saves them as 3 dimensional matrices under names A3, A10 and A20, where first two arguments correspond to rows and columns of matrix and third is a number for which k given matrix was generated – it uses k+1 since counting in Maltab starts from 1.

For example: A3(:, :, 3) is a matrix generated for N = 3 and k = 2.

Then script generates inverses of these matrices and saves them in 4 dimensional matrices names Ainv3, Ainv10, Ainv20. First 3 arguments of them are as in previous case and last argument is 1 for inverse calculated with LU factorization and 2 for LLT factorization.

For example: Ainv3(:, :, 3, 1) is a inverse of matrix generated for N = 3 and k = 2 with usage of LU factorization.

Note: In the script task 4 and 5 are made as one since they apply to the same matrices.

5. Task 5 – Indicators of uncertainty for matrices from Task 4

5.1. Introduction

Last task in this Assignment was about calculating Root Mean Squared and Maximal errors of inverse of matrices calculated in Task 4, and then presenting them on graphs.

5.2. Norms

To calculate errors stated in the Task, there are functions to calculate norms: “norm2” and “normInf”. Both functions take matrix on input and return single scalar on output.

After comparing values of norms calculated with my functions with norms calculated by Matlab, for all matrices generated in Task 4 for N=20 we can see that in case of 2-norm there are differences between them, however they are relatively small – biggest of them is equal to 3.7253e-08, which is much less than value of norm itself.

On the other hand, infinity-norms calculated by Matlab function are exactly identical as these calculated by mine function – the biggest difference is 0.

5.3. Calculated Errors

After running the Script, Root Mean Squared errors are stored in matrices called RMS3, RMS10, RMS20. Each of them is two dimensional matrix where first argument is for which k was this error calculated (as in Task 4 names) and second argument is 1 if its calculated for inverses made with LU factorization and 2 for those made with LLT factorization.

Similarly Maximal errors are called ME3, ME10, ME20 and they have the same arguments as RMS errors.

For Example: RMS3(3, 1) is Root Mean Squared error calculated for inverse of matrix generated for N=3 and k=2, made with LU factorization.

5.4. Graphs

For this task, Script generates 2 graphs: RMS errors and ME errors for matrices calculated with LU and LLT factorization. Each graph consist of six curves corresponding to size of matrices N=3, 10 and 20, and factorization used in making them.

Figure (5.1):

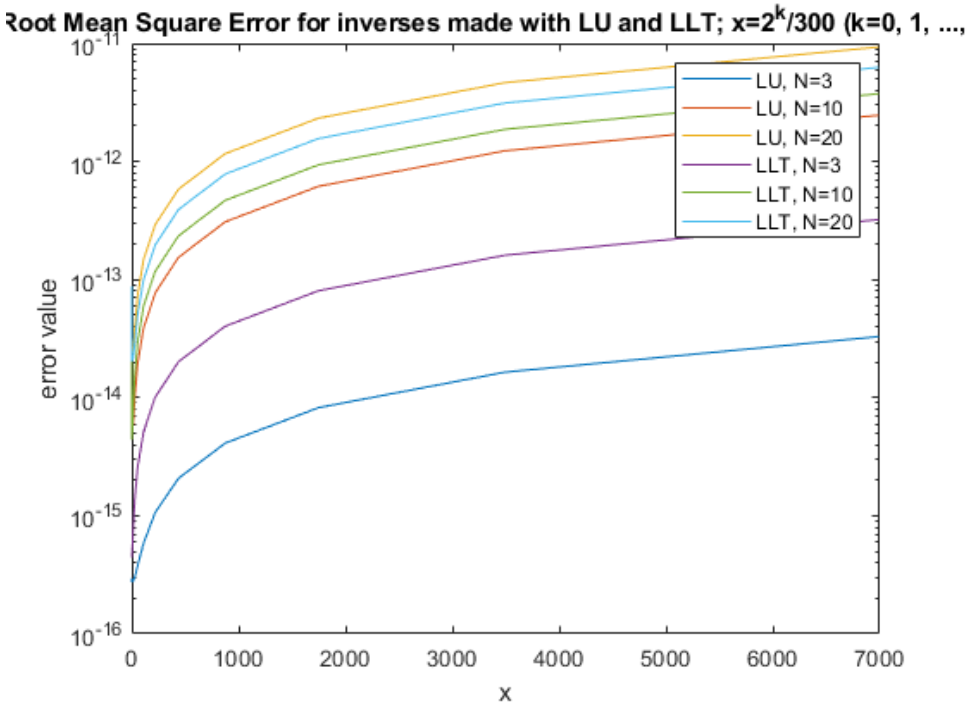
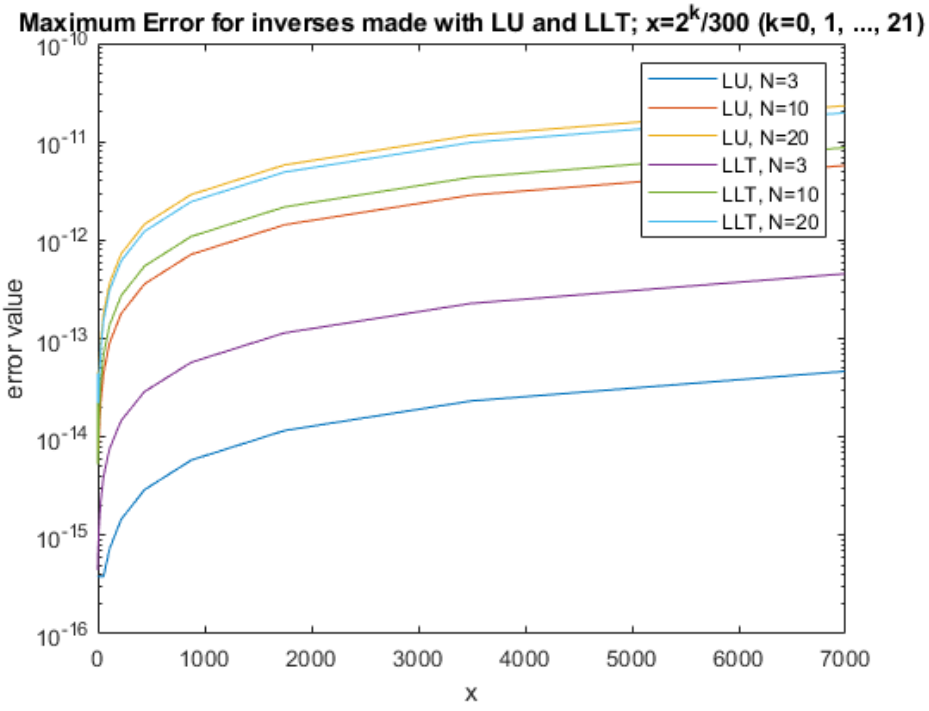


Figure (5.2):



5.5. Graphs conclusions

The first thing that can be seen from these graphs (*Figure (5.1)* & *Figure (5.2)*) is that in both cases (LU and LLT) the smaller size of the matrix the smaller are RMS errors as well as Max. errors. Also, the smaller x that was used to generate these matrices, the smaller the error. Therefore, since values of matrix depend both on N and x, there can be drawn conclusion that the smaller size of the matrix and the smaller are values in it, the better and more accurate the inversion by used algorithms will be.

Secondly, these graphs (*Figure (5.1)* & *Figure (5.2)*) can be used to compere LU and LLT factorization methods for finding inverse of matrices. It can be seen, that in both graphs, curves are arranged in the same order – from small N's at the bottom to big N's at the top. However, the LLT factorization method isn't always below LLT for the same N (or otherwise). Actually it can be seen, that for N=3 and N=10 errors of LU method are smaller than errors for LLT method, while for N=20, the errors of LLT method are slightly below those for LU. From this observation, can be drown conclusion, that **for small matrices size (N<20) better results one can achieve by inverting matrices with usage of LU factorization, however, for bigger matrices (N>20) the LLT method will be more accurate.**

5.6. Comparison of matrices inverses made with built in function, with these computed in Task 4

My Matlab script prints out biggest differences of single values of matrices, for matrices generated in task 4 and matrices generated by "inv()" function. From that we observe that these differences are not 0, but they are very small – biggest one jest smaller than 10^{-10}. We can see, that they differ in much greater way for smaller k numbers (differences for k=0 in all cases are about 10^{-11}) - for bigger k values differences are much smaller. Also, in all cases, the differences are bigger when size of the matrix is bigger. **Therefore, generally algorithms implemented in task 3 are more similar to these built in matlab ("inv" function) for smaller sizes of matrices and for smaller single values inside these matrices.**