Numerical Methods, project A, Number 31

Krzysztof Rudnicki Student number: 307585 Advisor: dr Adam Krzemieniowski

November 6, 2021

Contents

1	\mathbf{Pro}	olem 1 - Finding machine epsilion	3
	1.1	Problem	3
	1.2	Theoretical Introduction	3
		1.2.1 Definition of machine epsilion	3
		1.2.2 Practical applications of machine epsilion	3
	1.3	Solution	4
		1.3.1 Matlab code	4
	1.4	Discussion of the result	4
2	Pro	olem 2 - Solving a system of n linear equations - indicated	
	met	- · · · · · · · · · · · · · · · · · · ·	6
	2.1	Problem	6
	2.2	Theoretical Introduction	6
		2.2.1 Transform matrix into upper-triangular matrix	6
		2.2.2 Backward substitution	8
		2.2.3 Partial Pivoting	8
	2.3	Solution	9
		2.3.1 Main function	9
		2.3.2 checkIfMatrixIsSquareMatrix	9
		2.3.3 gaussianEliminationWithPartialPivoting	10
		2.3.4 partialPivoting	10
		2.3.5 partialPivotingSwapOneRow	10
		2.3.6 swapRowMatrix	11
		2.3.7 swapValueVector	11
		2.3.8 gaussianElimination	11
		2.3.9 substractRows	11
		2.3.10 backSubstitutionPhase	12
		2.3.11 iterativeResidualCorrection	12
		2.3.12 improveSolution	12
	2.4	Discussion of the result	13

3	\mathbf{Pro}	blem 3 - Solving a system of n linear equations - iterative	3
	algo	orithm	15
	3.1	Problem	15
	3.2	Theoretical introduction	15
	3.3	Solution	15
	3.4	Discussion of the result	15
4	Pro	blem 4 - QR method of finding eigenvalues	16
	4.1	Problem	16
	4.2	Theoretical introduction	16
	4.3	Solution	16
	4 4	Discussion of the result	16

Problem 1 - Finding machine epsilion

1.1 Problem

Write a program finding macheps in the MATLAB environment

1.2 Theoretical Introduction

1.2.1 Definition of machine epsilion

Machine epsilion is the maximal possible relative error of the floating-point representation. (Tatjewski, p.14) Machine epsilion is equal to 2^{-t} where t is number of bits in the mantissa. In our case when we use IEEE Standard 754, mantissa is 53 bits long with first bit omitted as it is always equal to '1', so we technically work with 52 bits mantissa which makes the machine epsilion equal to: $2^{-52} = 2.220446e - 16$

1.2.2 Practical applications of machine epsilion

Since macheps is connected to IEEE754 standard it is always equal to the same number, which means that we can safely compare results from different machines without worrying about their individual errors.

Macheps is also essential when we calculate cumulation of errors of given mathematical operation.

1.3 Solution

1.3.1 Matlab code

```
1 macheps = 1;
2 while 1.0 + macheps / 2 > 1.0
3 macheps = macheps/2;
4 end
```

Code above shifts macheps one bit to the right each iteration (by dividing by 2), it ends when we run out of mantissa bits which renders us unable to save smaller number. Due to underflow the value of macheps becomes 0 and therefore 1.0 > (macheps / 2) > 1.0 will become false.

1.4 Discussion of the result

```
format long
2
  disp(Display calculated macheps:)
3
  disp(macheps);
  disp(Display actual eps:)
  disp(eps);
  disp(Display 2^-52)
6
  disp(2^-52)
  disp(Display difference between calculated macheps and actual eps:)
  disp(macheps - eps)
  disp(Display difference between 2^-52 and actual eps:)
  disp(2^{-52} - eps) \setminus
  disp(Display difference between calculated macheps and 2^-52:)
  disp(macheps - 2^-52)
```

Display calculated macheps:

```
2.220446049250313e{-16}
```

Display actual eps:

 $2.220446049250313e{-16}$

Display 2^{-52} :

 $2.220446049250313e{-16}$

Display difference between calculated macheps and actual eps:

0

Display difference between 2^{-52} and actual eps:

0

Display difference between calculated macheps and 2^{-52} :

0

As expected they are all equal to each other. It means that our method of calculating macheps was correct.

Problem 2 - Solving a system of n linear equations - indicated method

2.1 Problem

Write a program solving a system of n linear equations Ax = b using the indicated method (Gaussian elimination with partial pivoting).

2.2 Theoretical Introduction

Gaussian elimination with partial pivoting consists of 3 main steps:

2.2.1 Transform matrix into upper-triangular matrix Starting conditions

We start with the system of linear equations looking like this:

In order for this method to work all the elements of diagonal line:

$$a_{11}, a_{22}, \ldots, a_{nn}$$

Must be different from zero since we will be dividing by them. We will denote rows as ' w_i ' where 'i' is number of the row.

Zeroing first column

We start transforming the system by **zeroing** elements in first column excluding first row element. We do it by multiplying first row by l_{i1} , where:

$$l_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}$$

And then substracting what we got $(l_{i1}w_1)$, from i row.

Doing so we obtain a system of linear equations:

Zeroing second column

We continue onto the second column, this time we will zero all elements except first and second rows. Row multiplier becomes:

$$l_{i2} = \frac{a_{i2}^{(2)}}{a_{22}^{(2)}}$$

Where:

$$a_{22}^{(2)} = (a_{22} - a_{12}l_{21})$$

And:

$$a_{i2}^{(2)} = (a_{i2} - a_{12}l_{i1})$$

They are modified values obtained from previous step. We continue as in the first step and we end up with:

Zeroing next columns

We repeat this process n-1 times and we end up with upper triangular matrix:

2.2.2 Backward substitution

After transforming the system we solve the system from last to first. First we calculate value of last element:

$$x_n = \frac{b_n}{a_{nn}}$$

Then one above:

$$x_{n-1} = \frac{b_{n-1} - a_{n-1,n} x_n}{a_{n-1,n-1}}$$

And so on, for x_k :

$$x_k = \frac{b_k - \sum_{j=k+1}^n a_{kj} x_j}{a_{kk}}$$

2.2.3 Partial Pivoting

Gaussian elimination method has one flaw, where it can come into halt if:

$$a_{kk}^{(k)} = 0$$

To avoid it we use method of pivoting, in our case we will use partial pivoting method. We do it before each Gaussian elimination step since this will lead to smaller error.

We first find a row i such that:

$$|a_{ik}^k| = \max_{i} \{|a_{kk}^k|, |a_{k+1,k}^k|, \cdots, |a_{nk}^k|\}$$

Then we swap this row with k-th row. Since the matrix we use is assumed to be nonsingular then $|a_{ik}^k| \neq 0$ will be always true. After that we continue with the Gaussian elimination method.

2.3 Solution

2.3.1 Main function

```
1
   function x = indicatedMethod(Matrix, Vector) % Name of the method
       as in the textbook
   % x stands for obtained result
       [~,Columns] = size(Matrix); % We need to know how big the
           matrix is in next steps
4
       % notice the '~', since we assume we use square matrix, we do
            not need
       % to have another variable for number of rows since it is the
            same as
6
       % number of columns
       checkIfMatrixIsSquareMatrix(Matrix);
8
        [Matrix, Vector] = gaussianEliminationWithPartialPivoting(
            Columns, Matrix, Vector);
9
       % Change matrix to upper triangular matrix
        [Matrix, Vector, x] = backSubstitutionPhase(Columns, Matrix,
           Vector);
11
       % Get the solution
       x = iterativeResidualCorrection(Matrix, x, Vector); % Improve
12
            on the solution
   end % end function
```

2.3.2 checkIfMatrixIsSquareMatrix

```
function checkIfMatrixIsSquareMatrix(Matrix)
[Rows,Columns] = size(Matrix);
if Rows ~= Columns
error ('Matrix is not square matrix!');
end % end if
end % end function
```

2.3.3 gaussianEliminationWithPartialPivoting

```
1
   function [Matrix, Vector] = gaussianEliminationWithPartialPivoting(
       Columns, Matrix, Vector)
2
        for j = 1: Columns
                centralElement = max(Matrix(j:Columns,j));
                % we stay in the same row (j) but we change columns, as
4
                     in the
                % textbook
5
6
                [Matrix, Vector] = partialPivoting(Matrix, Vector, j,
                    centralElement, Columns);
                % ensures that a_kk != 0 and reduces errors
8
                [Matrix, Vector] = gaussianElimination(j, Columns,
                    Matrix, Vector);
9
                % change matrix into upper triangular matrix
        end % end for
11
   end % end function
```

2.3.4 partialPivoting

2.3.5 partialPivotingSwapOneRow

2.3.6 swapRowMatrix

```
function Matrix = swapRowMatrix(Matrix, j, k)

temp = Matrix(j , :); % ' : ' denote all elements in jth row

Matrix(j , :) = Matrix(k, :);

Matrix(k, :) = temp; % temp equal to previous value of jth row
end
```

2.3.7 swapValueVector

```
function Vector = swapValueVector(Vector, j, k)
temp = Vector(j);
Vector(j) = Vector(k);
Vector(k) = temp; % temp equal to previous value of k element
of vector
end % end function
```

2.3.8 gaussianElimination

2.3.9 substractRows

2.3.10 backSubstitutionPhase

```
1
   function [Matrix, Vector, x] = backSubstitutionPhase(Columns,
       Matrix, Vector)
 2
        for k = Columns : -1 : 1
3
        \% Start at final column and move by -1 each iteration until we
            reach 1
 4
            equation = 0;
5
            for j = k+1: Columns
6
                equation = equation + Matrix(k,j) * x(j, 1);
                % even though x is a vector we still need to put '1' to
 7
                     ensure
                % that number of columns in the first matrix matches
8
                    number of
9
                % rows in second matrix
            end % end for
11
12
            x(k, 1) = (Vector(k,1) - equation) / Matrix(k,k);
13
            % even though x is a vector we still need to put '1' to
14
            % that we do not exceed array bounds
        end % end for
16
   end % end function
```

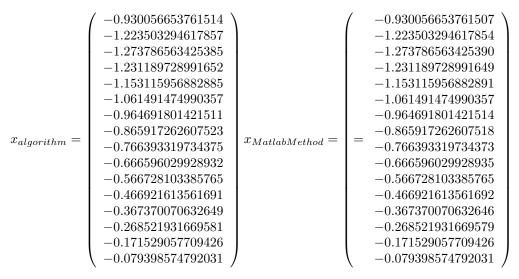
2.3.11 iterativeResidualCorrection

```
function x = iterativeResidualCorrection(Matrix, x, Vector)
residuum = Matrix*x — Vector; % as in the book
newResiduum = residuum;
x = improveSolution(x, newResiduum, residuum, Matrix, Vector);
end % end function
```

2.3.12 improveSolution

2.4 Discussion of the result

Solutions vectors for matrix A and vector A and n = 16:



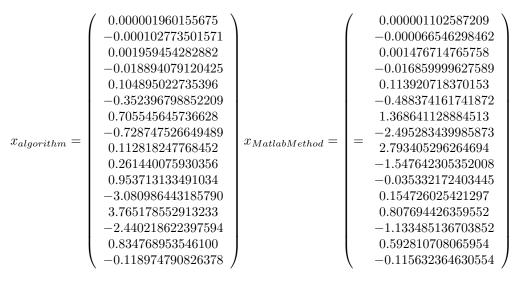
Error for 'A' method for algorithm:

3.383918772654241

Error for 'A' method for matlab method:

3.383918772654241

Solutions vectors for matrix B and vector B and n=16 (Both multiplied by 1.0e+17):



Error for 'B' method for algorithm:

5.699979882700911e + 17

Error for 'B' method for matlab method:

4.569118543317684e + 17

Problem 3 - Solving a system of n linear equations - iterative algorithm

- 3.1 Problem
- 3.2 Theoretical introduction
- 3.3 Solution
- 3.4 Discussion of the result

Problem 4 - QR method of finding eigenvalues

- 4.1 Problem
- 4.2 Theoretical introduction
- 4.3 Solution
- 4.4 Discussion of the result

Bibliography

[1] Piotr Tatjewski (2014) Numerical Methods, Oficyna Wydawnicza Politechniki Warszawskiej