

ENUME project report

Project A: linear equation systems and matrix eigenvalues

Michał Szopiński

<https://github.com/Lachcim/szopinski-enum>
Project number 62

November 12, 2020

0 Abstract

This project explores the numerical methods of solving systems of linear equations and finding eigenvalues of matrices. The following concepts are discussed within this document:

- Computing the machine epsilon of an environment
- Solving linear equation systems using Gaussian elimination with partial pivoting
- Solving linear equation systems using the Jacobi and Gauss-Seidel algorithms
- Finding the eigenvalues of a symmetric matrix using the QR method with and without shifts

1 Task 1: Computing the machine epsilon

1.1 Overview

The goal of this task was to find the machine epsilon of the MATLAB environment, i.e. the maximum relative error of a floating-point representation of a number.

1.2 Implementation

The computation of the machine epsilon is based on the following two observations:

1. In binary computer environments, the epsilon is a negative power of two.
2. The machine epsilon can also be expressed as:

$$eps = \min\{x \in M : fl(1 + x) > 1 \wedge x > 0\} \quad (1.1)$$

i.e. the smallest representable number such that, when added to 1, has a floating point representation greater than 1.

An arbitrary starting point, in this case 2^0 , can thus be chosen, and its value can then be sequentially halved until the criterion $fl(1 + x) > 1$ is no longer met.

1.3 Program output

```
Found epsilon is 2.2204e-16  
Epsilon is 2.2204e-16
```

1.4 Observations

An epsilon of $2.2204 \cdot 10^{-16} = 2^{-52}$ is indicative of a standard double-precision IEEE 754 floating point binary representation.

2 Task 2: Solving systems of linear equations using Gaussian elimination

2.1 Overview

The task consisted of two subtasks, each specifying a different system of equations to solve. The systems were to be solved using Gaussian elimination with partial pivoting. The size of the system was sequentially doubled until the computation time became prohibitive. The error of each solution was to be noted.

Additionally, for systems whose number of equations was equal to 10, residual correction was to be applied and the solution was to be noted.

The systems were given by:

$$A_{ij}^{(a)} = \begin{cases} 4 & \text{for } i = j \\ 1 & \text{for } i = j \pm 1 \\ 0 & \text{otherwise} \end{cases} \quad b_i^{(a)} = 4 + 0.3i$$
$$A_{ij}^{(b)} = \frac{6}{7(i+j+1)} \quad b_i^{(b)} = \begin{cases} \frac{1}{3i} & \text{for even } i \\ 0 & \text{for odd } i \end{cases}$$

2.2 Implementation

Gaussian elimination is a process that takes a matrix describing a linear equation system and returns its LU (lower-upper) decomposition, enabling the system to be cheaply solved for any vector b . The triangular matrices L and U can be supplemented with b to find the solution vector x through a process called back-substitution.

2.2.1 LU decomposition of a matrix

The function `gausseli` takes an equation system and returns the L matrix, the U matrix, the permutation matrix P as well as a matrix describing the entire system post-elimination, to be immediately handed to back-substitution for the calculation of the result.

The elimination itself is done by traveling along the diagonal of `eqsys` and for each encountered cell, eliminating (reducing to 0) the coefficients directly below it. The program does so by multiplying the current row by a constant and subtracting it from the rows beneath. This constant, called `reductor` in

the code, is given as the ratio of the cell to be reduced and the current diagonal cell. The reductor is then stored in the L matrix.

2.2.2 Partial pivoting

The above algorithm alone is prone to errors when a value on the diagonal is equal to 0 and the reductor cannot be constructed. For this reason, partial pivoting is added.

For each cell on the diagonal, the program looks at the cells beneath it and finds the one with the greatest absolute value. If it exceeds the absolute value of the current cell, the corresponding two rows are swapped. This ensures that the divisor making up the reductor is as far from zero as possible.

Because the input matrix changes while being decomposed, a permutation matrix P is introduced to enable its recomposition from L and U , satisfying the equation:

$$LU = PA \tag{2.1}$$

Finally, the matrix L is also permuted to reflect changes in the layout of the system.

2.2.3 Back-substitution

A triangular matrix supplied with a constant vector b can be easily solved by traveling along its diagonal and eliminating the factors directly above the visited cell. In the end, the matrix is transformed into an identity matrix and the vector alongside it is transformed into x .

2.2.4 Residual correction

Residual correction is a method of improving the accuracy of the results obtained from Gaussian elimination. By reusing the already obtained matrices L , U and P , the system is solved once again, this time with b replaced with the vector of errors of the solution. The resulting vector δx is then subtracted from the previous solution. This procedure may be repeated as needed.

2.3 Program output

2.3.1 Error plots

2.3.2 Solutions for $n = 10$

Solution to subtask a:

i	x_i	error
1	0.8953	0
2	0.71881	-8.8818e-16
3	0.82946	0
4	0.86335	0
5	0.91714	0
6	0.9681	0
7	1.0105	0
8	1.0901	0
9	1.0293	0
10	1.4927	-8.8818e-16

Error: 1.2561e-15

With residual correction:

i	x_i	error
1	0.8953	0
2	0.71881	0
3	0.82946	0
4	0.86335	0
5	0.91714	0
6	0.9681	0
7	1.0105	0
8	1.0901	0
9	1.0293	0
10	1.4927	0

Error: 0

Solution to subtask b:

i	x_i	error
1	-2.7253e+08	7.6294e-06
2	9.5052e+09	-6.3578e-05
3	-1.2041e+11	-2.2888e-05
4	7.6994e+11	-8.138e-05
5	-2.8351e+12	5.3406e-05
6	6.3794e+12	-5.171e-05
7	-8.9152e+12	-1.5259e-05
8	7.5508e+12	-2.5431e-05
9	-3.5489e+12	8.3923e-05
10	7.1024e+11	7.1208e-06

Error: 0.00015731

With residual correction:

i	x_i	error
1	-2.7292e+08	-5.3406e-05
2	9.5191e+09	-7.8837e-05
3	-1.2059e+11	-6.8665e-05
4	7.7108e+11	-7.3751e-05
5	-2.8392e+12	-6.8665e-05
6	6.3888e+12	-7.4599e-05
7	-8.9283e+12	-8.7738e-05
8	7.562e+12	1.2716e-05
9	-3.5542e+12	-5.722e-05
10	7.1129e+11	3.3061e-06

Error: 0.00020161

2.4 Observations

Matrices from both subtasks a and b lent themselves nicely to being solved using Gaussian elimination. Their absolute errors, when divided by the solutions themselves, resulted in very small relative errors, nearing the machine epsilon.

The computation time became prohibitive beyond $n = 320$, showing that Gaussian elimination is an effective method of solving large systems of equations.

Residual correction was very effective for the matrix from subtask a, reducing the error to a perfect 0. For subtask b, on the other hand, no matter the amount of iterations, the error could not be reduced any further. This might be related to a high condition number of the matrix, $2.4225 \cdot 10^{14}$, prohibiting it from producing accurate enough results.

3 Task 3: Solving systems of linear equations using the Jacobi and Gauss-Seidel algorithms

3.1 Overview

In this task, a predefined, fixed-size system of equations was to be solved using the Jacobi and Gauss-Seidel iterative algorithms. The algorithms were to be iterated until the accuracy $\|Ax_k - b\| < 10^{-10}$ was achieved. The error of each iteration was to be noted.

The system to be solved:

$$\begin{aligned}18x_1 + 2x_2 - 3x_3 + x_4 &= 7 \\2x_1 - 25x_2 + 5x_3 - 18x_4 &= 12 \\x_1 + 3x_2 + 13x_3 - 8x_4 &= 24 \\x_1 + x_2 - 2x_3 - 10x_4 &= 20\end{aligned}$$

Afterwards, one of the algorithms was to be applied to matrices from task 2 a and b, for $n = 10$.

3.2 Implementation

The general formula describing iterative methods of solving linear equation systems is:

$$x^{(i+1)} = Mx^{(i)} + w \quad (3.1)$$

Where the matrix M and the vector w are specific to the chosen algorithm.

3.2.1 The Jacobi algorithm

The first step of the Jacobi algorithm is to split the matrix A into three matrices: the lower triangular matrix, the upper triangular matrix and the diagonal matrix.

Because the diagonal matrix is only ever used in its inverse form, the function doing the splitting only returns the inverse diagonal matrix. It utilizes the fact that the inverse of a diagonal matrix is a matrix of the inverse of its diagonal elements.

Once the matrices L , U and D^{-1} have been obtained, the parameters M

and w are given by:

$$\begin{aligned} M &= -D^{-1}(L + U) \\ w &= D^{-1}b \end{aligned}$$

The step described in (3.1) is then performed until the desired accuracy is achieved.

3.2.2 The Gauss-Seidel algorithm

The Gauss-Seidel algorithm is similar to the Jacobi algorithm in that it utilizes the lower, upper and inverse diagonal matrices to calculate subsequent iterations. An important difference is that M and w are not constant and they cannot be trivially calculated.

An iteration of the Gauss-Seidel algorithm can be written as:

$$\begin{aligned} Dx^{(i+1)} &= -Lx^{(i+1)} - (Ux^{(i)} - b) \\ &= -Lx^{(i+1)} - w^{(i)} \end{aligned}$$

The presence of $x^{(i+1)}$ on both sides of the equation might appear problematic, however, it's crucial to observe that individual elements of the vector $x_n^{(i+1)}$ can be calculated based solely on the knowledge of their preceding elements, with $x_1^{(i+1)}$ being completely independent:

$$\begin{aligned} x_1^{(i+1)} &= (-w_1^{(i)}) \cdot D_{11}^{-1} \\ x_2^{(i+1)} &= (-L_{21} \cdot x_1^{(i+1)} - w_2^{(i)}) \cdot D_{22}^{-1} \\ x_3^{(i+1)} &= (-L_{31} \cdot x_1^{(i+1)} - L_{32} \cdot x_2^{(i+1)} - w_3^{(i)}) \cdot D_{33}^{-1} \\ &\dots \end{aligned}$$

And so, on each step, the vector $w^{(i)}$ is calculated first, after which the elements of $x_n^{(i+1)}$ are computed in sequence.

3.3 Program output

3.3.1 Error plots

3.3.2 Solutions

Solution using Jacobi:

i	x_i
1	0.44705
2	1.0232
3	0.38763
4	-1.9305

Error: 6.1548e-11

Solution using Gauss-Seidel:

i	x_i
1	0.44705
2	1.0232
3	0.38763
4	-1.9305

Error: 4.6134e-11

Solution to task 2a using Gauss-Seidel:

i	x_i
1	0.8953
2	0.71881
3	0.82946
4	0.86335
5	0.91714
6	0.9681
7	1.0105
8	1.0901
9	1.0293
10	1.4927

Error: 3.01e-11

3.4 Observations

In both algorithms, the error of the solution decreased exponentially in relation to the number of iterations.

The Gauss-Seidel method proved to be faster than the Jacobi method, with the desired accuracy achieved in just 32 iterations compared to 43.

The matrix from task 2a was solved correctly. The 2b variant, however, could not achieve the desired accuracy and the algorithm became stuck in a loop, iterating infinitely in an attempt to reduce the error. This again may be related to the high condition number of the matrix, or to the fact that the measured error is absolute rather than relative.

4 Task 4: Finding the eigenvalues of a symmetric matrix using the QR method

4.1 Overview

The problem presented in this task was to find the eigenvalues of a symmetric matrix by utilizing the properties of its QR decomposition. Two variants of the algorithm were to be used, one with and one without shifting the diagonal to accelerate the convergence. The algorithms were to be iterated until the off-diagonal elements decreased below the threshold of 10^{-6} . The resultant final matrices were to be printed.

The matrix of choice:

$$\begin{bmatrix} 1 & 1 & 7 & 5 & 2 \\ 1 & 8 & 5 & 4 & 4 \\ 7 & 5 & 0 & 8 & 8 \\ 5 & 4 & 8 & 0 & 8 \\ 2 & 4 & 8 & 8 & 1 \end{bmatrix}$$

4.2 Implementation

The QR method is based on the observation that by repeatedly decomposing the matrix into its orthogonal matrix Q and triangular matrix R and recomposing it as $R * Q$, the matrix will converge into a diagonal matrix of its eigenvalues by means of similarity.

4.2.1 QR decomposition using the Gram-Schmidt algorithm

The columns of the input matrix A are orthogonalized with the modified Gram-Schmidt algorithm, producing matrices Q and R , which can later be multiplied together to reverse the process.

The algorithm iterates over the columns of A and orthogonalizes every column in front of it with relation to the currently visited column. This is done by calculating the dot product of the two columns and using it to subtract the “common part” from the second column. The matrix Q is produced in the process, as well as R as a byproduct.

4.2.2 QR method without shifts

The QR method for finding the eigenvalues of a matrix without shifts is a naive implementation of the observation described at the beginning of this section. The matrix is repeatedly decomposed and recomposed until its off-diagonal elements converge below the prescribed threshold.

4.2.3 QR method with shifts

The QR method may be accelerated by shifting the diagonal of the matrix towards an approximation of the expected eigenvalues. The required shift can be calculated by finding the eigenvalues of the lower right 2×2 sub-matrix and choosing the one closer to the lower right corner of said sub-matrix.

The eigenvalues of the sub-matrix can be found by solving the characteristic equation of the sub-matrix, $\det(A - \lambda I) = 0$. This will result in a quadratic equation whose solutions are guaranteed to be real for a symmetric matrix.

This shift is applied on every iteration of the algorithm. Once all non-diagonal values of the lowermost row of the matrix have decreased below the threshold, the matrix is deflated, i.e. its size is reduced by one in both dimensions. The algorithm stops once the matrix can't be deflated any further.

4.3 Program output

Solution using QR without shifts:

23.3571	-9.3695	-7.8773	5.0307	-1.1410
---------	---------	---------	--------	---------

Final matrix:

23.3571	0.0000	-0.0000	-0.0000	0.0000
0.0000	-9.3695	-0.0000	-0.0000	0.0000
0.0000	-0.0000	-7.8773	0.0000	-0.0000
0.0000	-0.0000	0.0000	5.0307	0.0000
0.0000	0.0000	0.0000	0.0000	-1.1410

Iteration count: 75

Solution using QR with shifts:

5.0307	-1.1410	-7.8773	-9.3695	23.3571
--------	---------	---------	---------	---------

Final matrix:

23.3571	-0.0445	0.0000	0.0000	0.0000
0.0000	-9.3695	0.0000	-0.0000	0.0000
0.0000	-0.0000	-7.8773	0.0000	-0.0000
0.0000	-0.0000	0.0000	-1.1410	0.0000
0.0000	0.0000	0.0000	0.0000	5.0307

Iteration count: 9

4.4 Observations

The addition of shifting greatly improved the convergence ratio and the same values could be obtained in just 9 iterations instead of 75.