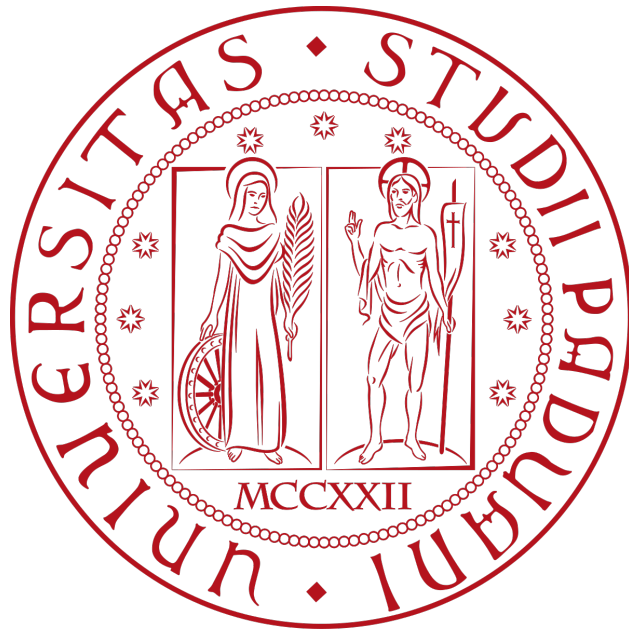# Università degli Studi di Padova

Report about:
2nd Matlab project

Realized by:
Riccardo Berengan, matriculation number 2080041
Michele Dioli, matriculation number 2077629
Gabriele Di Pietro, matriculation number 2010000

1   Introduction

The eigenvalues are the solutions to the equation $det(A - \lambda I) = 0$. The eigenspace is the vectorial subspace generated by all the vectors and eigenvalues λ, together with the null vector.
Example:

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

$$det(A - \lambda I) = det \begin{bmatrix} 4 - \lambda & 2 \\ 1 & 3 - \lambda \end{bmatrix} \iff (\lambda - 5)(\lambda - 2) = 0 \iff \lambda_1 = 5 \lambda_2 = 2$$

The geometric multiplicity is the dimension of the eigenspace, associated with an eigenvalue then $|n - \text{rk}(A - \lambda I)|$. Our purpose is to calculate the geometric and the algebraic multiplicities to define if a matrix is diagonalizable or not.

2   Calculation

2.1   Multigeo Function

For the calculation, count the "null" values *(smaller than a tolerance)* in the matrix U obtained using the factorization LU. It can be done because: A blank row in U means that the linear system of $(A - \lambda I)x = 0$ has a free solution, a blank row does not depend from the other rows of the matrix, the blank rows indicate that there are linear dependencies between the rows. The approach is to count the elements in the diagonal line that are larger than a tolerance, thus counting the matrix rank. Then subtract this number from the matrix dimension to obtain the eigenspace dimension of the eigenvalue that coincides with the geometric multiplicity.

2.2   Myobjective Function

The function `myobjective` is used to calculate the funcion f and the function g from a scalar value z and a squared matrix A.
The first step is to calculate the matrix $B = A - zI$, and then, by using the LU factorization using `lu(B)` in Matlab that returns U, upper matrix, and the L lower matrix. Then we calculate P, the permutation matrix.
$P = \pm 1$ based on the number of permutations that occurred during this process, from which we calculate the determinant matrix P. Once we found it we calculate the B determinant as $det(P) \cdot \prod_{i=1}^{n} U_{ii}$, where U_{ii} are the elements on the U diagonal. $f = det(B)$
After this the function calculates the inverse of the matrix B and calculates g as the inverse of the track *(it sums the elements on the diagonal)* of B.

1

2.3 Multialg Funcion

It consider the polynomial with m as the algebraic multiplicity $f(x) = (x - \lambda)^m g(x)$ and the Newton method $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ to find the roots of $g(x)$. We analyze the speed of convergence $f(x) = (x - \lambda)^{m-1} g(x) + (x - \lambda)^m g'(x)$
$f'(x) = 0 \iff x = \lambda$, $m \geq 1$ isn't immediately the root $m(x - \lambda)^{m-1}$ and it's responsible for the speed of convergence, so the Newton method applied to a root that has multiplicity $m \geq 1$ has linear speed of convergence instead of a quadratic one.
The error estimate is reduced, depending on the root multiplicity $|x_{n+1} - \lambda| \approx |x_n - \lambda|^{\frac{m-1}{m}}$
$\frac{m-1}{m}$ is the speed of convergence
(Newton method initialized near multiple roots $\Rightarrow |e_{n+1}| \approx C|e_n|^{\frac{m-1}{m}}$)
We calculate the eigenvalue with the newton method, iterating until a specified number of iterations and/or until we reach a sensible tolerance. The function then applies the Newton method to the characteristic polynomial $f(\lambda) = det(A - \lambda I) \Rightarrow \lambda_{n+1} = \lambda_n - \frac{f(\lambda_n)}{f'(\lambda_n)}$. With this process, we find the eigenvalue. To calculate the algebraic multiplicity we divide the eigenvalue by the characteristic polynomial, and we count the times that we can do this division having a result different from zero.

3 Test

3.1 File `main.m`

The main file allows to define a test matrix by modifying the `lambda` vector, it allows to change the starting point for the newton method `l0` and also to change the tolerance parameter `toll`.
The matrix used in tests is created through the modification of the `lambda` parameter. After the definition of lambda we call the `creaJacob` function that uses `lambda` as input and it returns the Jordan block matrix as output. Then the main creates Q through `Q = orth(randn(n))`, that creates an orthogonal matrix. After that, the matrix A is calculated by using the following structure: $A = Q^t J Q$

3.2 File test.m

The test.m file runs some tests to verify correctness of the multialg and multigeo functions. These function's purpose is to calculate the eigenvalues of the matrix and their algebraic and geometric multiplicities. The program defines a group of test cases, each with a matrix of eigenvalues that are specified by the lambda value, the starting point l0 of the Newton method, and a tolerance toll.

```
TestCases = {
    struct('lambda', [1,1,1,1,1,4,4,4,4,4,4,4], 'l0', 3.8, '
    toll', 1e-4),
    struct('lambda', [1, 1.01, 1.02, 5, 5.01, 5.02, 10,
    10.001, 10.002, 10.003], 'l0', 10.1, 'toll', 1e-4),
    struct('lambda', [1, 1.01, 1.02, 5, 5.01, 5.02, 10,
    10.001, 10.002, 10.003], 'l0', 1.0125, 'toll', 1e-4),
    struct('lambda', [1,1,1,4,5,6], 'l0', 3.5, 'toll', 1e-4),
    struct('lambda', [1,1,1,4,5,6], 'l0', 3.9, 'toll', 1e-4),
    struct('lambda', [1,1,1,4,5,6,4], 'l0', 3.9, 'toll', 1e
    -4),
    struct('lambda', [1,1,1,4,5,6,4,4], 'l0', 3.5, 'toll', 1e
    -4),
    struct('lambda', [1,1,1,4,5,6,4,4,4], 'l0', 3.7, 'toll',
    1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4,4], 'l0', 3.3, 'toll'
    , 1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4], 'l0', 3.7, 'toll',
    1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4,4], 'l0', 3.5, 'toll'
    , 1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4,4,4], 'l0', 4.3, '
    toll', 1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4,4,4], 'l0', 3.5, '
    toll', 1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4,4,4], 'l0', 3.4, '
    toll', 1e-4),
    struct('lambda', [4,4,4,4,4,4,4,4], 'l0', 0., 'toll', 1e
    -4),
};
```

1: Test Cases

3.2.1 Test execution

For every test case the program runs the following steps:

1. Print the number of the test and the values of `lambda`, `l0`, and `toll`.

2. Creates the diagonal matrix `J` with the `lambda` values and adds 1 to the elements on the diagonal to manage multiplicities.

3. It generates a `Q` orthogonal matrix and calculates the `A` matrix as `A = Q' * J * Q`.

4. It uses the `multialg` function to calculate the eigenvalue `l` and his algebraic multiplicity named `m`.

5. If the Newton method converges, it calculates the geometric multiplicity, named `k`, of the eigenvalues by using the `multigeo` function.

6. It prints the test results, including the calculated eigenvalue, the algebraic and geometric multiplicities and a message that confirms the convergence.

The program gives a detailed analysis of the results for each test, permitting a verification on the accuracy and the efficiency of the implemented functions.

```matlab
for testID = 1:length(TestCases)
    fprintf('\nExecuting Test %d...\n', testID);
    lambda = TestCases{testID}.lambda;
    fprintf('[%s]\n', sprintf('%g ', lambda));
    l0 = TestCases{testID}.l0;
    fprintf('l0 = %d\n', TestCases{testID}.l0);
    toll = TestCases{testID}.toll;
    fprintf('toll = %d\n', TestCases{testID}.toll);
    J = diag(lambda);
    for i = 2:length(lambda)
        if J(i,i) == J(i-1,i-1)
            J(i-1,i) = 1;
        end
    end
    n = length(lambda);
    Q = orth(randn(n));
    A = Q' * J * Q;
    A1 = A;
    disp(J);
    it = 5;
    maxit = 50;
    [l, m, flag] = multialg(A, l0, toll, it, maxit);
    if flag == 1
        fprintf('Newton convergence.\nCalculated eigenvalue:
%f\nEstimated algebraic multiplicity: %d\n', l, m);
        l1 = round(l);
        toll = 1e-6;
        k = multigeo(A1, l1, toll);
        fprintf('Geometric multiplicity of %f: %d\n', l1, k);
    else
        fprintf('Non-convergent Method.\n');
    end
    linea = 50;
    fprintf('%s\n', repmat('-', 1, linea));
end
```

2: Esecuzione dei test

3.2.2 Terminal output

For convenience we'll print just one test case, the other ones are similar.

```
Executing Test 14...
[1 1 1 4 5 6 4 4 4 4 4 4 ]
l0 = 3.400000e+00
toll = 1.000000e-04
      1      1      0      0      0      0      0      0      0      0      0      0
      0      1      1      0      0      0      0      0      0      0      0      0
      0      0      1      0      0      0      0      0      0      0      0      0
      0      0      0      4      0      0      0      0      0      0      0      0
      0      0      0      0      5      0      0      0      0      0      0      0
      0      0      0      0      0      6      0      0      0      0      0      0
      0      0      0      0      0      0      4      1      0      0      0      0
      0      0      0      0      0      0      0      4      1      0      0      0
      0      0      0      0      0      0      0      0      4      1      0      0
      0      0      0      0      0      0      0      0      0      4      1      0
      0      0      0      0      0      0      0      0      0      0      4      1
      0      0      0      0      0      0      0      0      0      0      0      4

Newton convergence.
Calculated eigenvalue: 4.000005
Estimated algebraic multiplicity: 7
Geometric multiplicity of 4.000000: 2
----------------------------------------------------
```
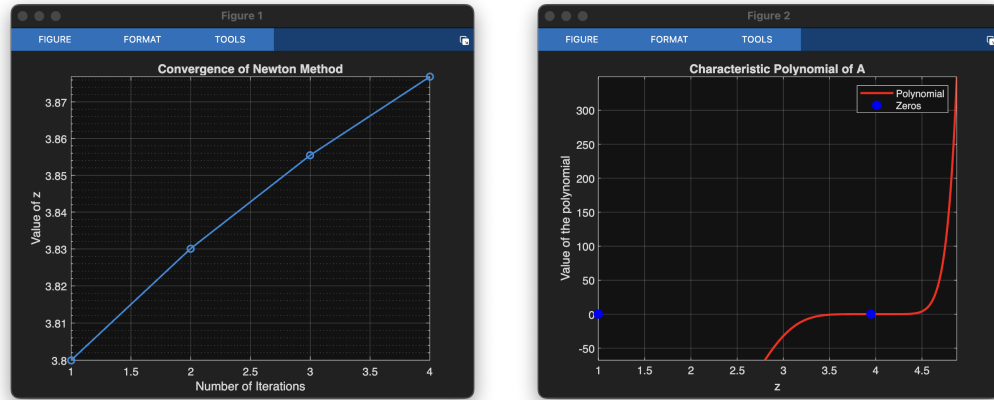
3: Tests case output

Figure 1: *Newton method's convergence and A polynomial*

On some separate graphs are shown both the Newton convergence and the A polynomial.

4   Conclusions

We can say that the software manages the problem well and that it can find the eigenvalues also with far initial estimations, and the Newton method will still converge (*for example*, `10`= 3.5 for $\lambda = 4$).

The software has been tested with matrices of various dimensions and with different algebraic and geometric multiplicities. In every case the Newton method is able to converge, unless the choice of the starting point is very far from the eigenvalue that we want to find.