Coursework 3

Lucjano Muhametaj

January 13, 2022

Abstract

The hash is 5eef71cef61cdc317131f88d18331271f5c39156

1 Exercise 1

The weekly exercises can be found on GITHUB https://github.com/Imperial-MATH96023/clacourse-2021-lm1621

2 Exercise 2

2.1 Part a

Running the code on the script `ex2.py` it is quite evident that applying `pure_QR` to the matrix we are not converging to a diagonal (or upper triangular matrix). In fact we can empirically observe that we are getting a tridiagonal matrix. This can be even more evident if we use the `matplotlib` library to do a sort of a 'plot' of some of the matrices we obtain.

So we cannot read the eigenvalues off. The reason why we have not convergence to a diagonal (or

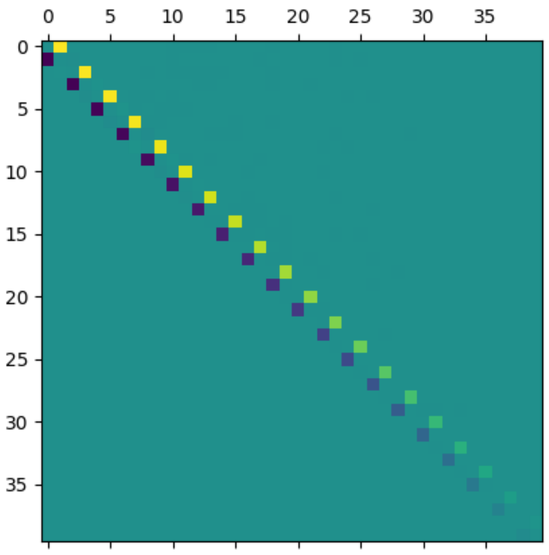

Figure 1: graphic representation ex 2a

tridiagonal) matrix is that (as we can see printing the eigenvalues of the starting matrix), we will have couple of conjugates. So independently from the order of the eigenvalues the condition

$$|\lambda_1| > |\lambda_2| > |\lambda_3|... \tag{1}$$

discussed in the notes (section 5.9) is not true.

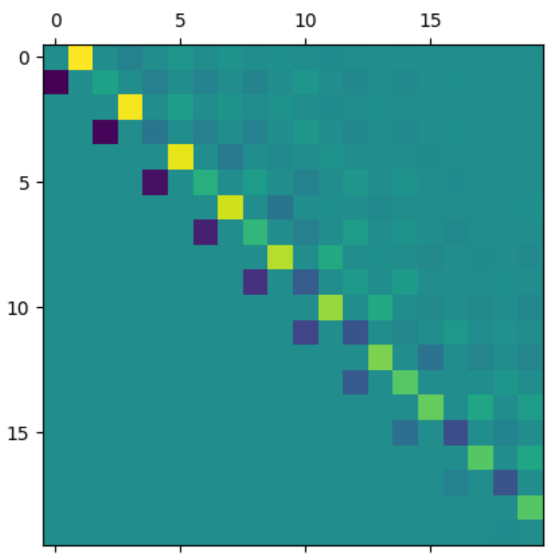

Figure 2: graphic representaion of ex 2d

2.2 Part b

The output of `pure_QR` is an antisymetric block matrix composed by n $2x2$ blocks of the form $\begin{pmatrix} 0 & a_i \\ -a_i & 0 \end{pmatrix}$. The eigenvalues of the matrix are the eigenvalues of the blocks and each block has characteristic polynomial $\lambda^2 + a_i^2$, and so eigenvalues $\pm i \cdot a_i$.
So a method to find the eigenvalues given the output of the `pure_QR` is to check the $2x2$ block matrices and print the eigenvalues (multiplying the nonzero entries per i, obtaining as said before $\pm i \cdot a_i$). (A way to see why all this happens is that applying `pure_QR` we divide in 2 by 2 subspaces.

2.3 Part c

The code for this part can be found on `ex2c.py` and the test on `test_exex2c.py`, where I have tested the correctness of the eigenvalue function provider for different dimensions of the starting matrix.

2.4 Part d

Applying `pure_QR` to B we have something similar to what we obtained for A, except from the fact that we are going to have 'less almost equal to zero entries' in the upper triangular part. So it is not going to look as a tridiagonal matrix. As before we plot it to have a graphical view of what is happening. Moreover the $2x2$ block are not antysimmetric as before, but we can still use them since we know that the determinant of $\begin{pmatrix} A & B \\ 0 & C \end{pmatrix}$ is equal to $det(A)det(C)$. So in our case when we calculate the characteristic polynomial of our matrix after applying `pure_QR` we multiply the characteristic polynomials of the $2x2$ blocks on the diagonal, which are of the form $\begin{pmatrix} 0 & a_i \\ b_i & 0 \end{pmatrix}$. So at the end we will have (taking them from each block) eigenvalues of the form $\pm\sqrt{a_i b_i}$.
This gives us also an algorithm that we can use to obtain the eigenvalues. We just need to take the $2x2$ block matrices and calculate 2 eigenvalues for each of them.
The code for this point can be found on `ex2d.py`. In particular `matrixconstructionB` creates the matrix, `eigenvaluescalculatorB` finds the eigenvalues and finally `test_eigenchecker` tests for several dimensions the algorithm.

3 Exercise 3

3.1 Part a

We want to show that combining the QR with the tridiagonal form reduction, in the context of symmetric matrices the symmetric tridiagonal structure is preserved through the steps. For what regards the symmetry we proceed by induction. We know that the starting matrix is symmetric and we assume that at the step k, the matrix $A^{(n)}$ is symmetric. We want to prove that also the $A^{(n+1)}$ is symmetric. We will have $Q^{(n+1)}R^{(n+1)} = A^{(n)}$ and $A^{(n+1)} = R^{(n+1)}Q^{(n+1)}$. So we can write $A^{(n+1)} = Q^{(n+1)*}A^{(n)}Q^{(n+1)}$. We are dealing with real matrix, and being Q orthonormal

$$A^{(n+1)T} = (Q^{(n+1)T}A^{(n)}Q^{(n+1)})^T = Q^{(n+1)T}A^{(n)T}Q^{(n+1)} = A^{(n+1)} \tag{2}$$

Now we need to prove that it is tridiagonal.
When A non-singular (so that in the first step of gram-schmidt we can divide for r_{11}, since it's different from 0) working through the columns of Q using Gram-Schmidt, since the starting A is tridiagonal we get that Q is an upper Hessemberg.
Using QR at each step we get $\langle e_i, R^{(n)}Q^{(n)}e_j\rangle = 0$ for $i \geq j+2$, and so we have that lower bandwith equal to 1 for A for the n+1 iteration. We can proceed in the same way due to symmetry to state that A is tridiagonal. We conclude saying that this construction works also for A being singular (is $r_{ii} = 0$, we just see that the i-th columns of Q has all zeros after the entry i).

3.2 Part b

I have modified the `pure_QR` as requested. To use the new stopping condition we need to take `case=True`. On the script `ex3b.py` in the folder `cw3` we construct the matrix A as requested, we apply hessenberg and after that the new version of `pure_QR`. From the matrix we get, that is almost upper triangular (we check this with an automatic test in `testex3b.py`), we can read the eigenvalues and check if they are correct. We can see that the 'smallest' eigenvalues obtained with this method, are the ones that converge. On the other hand for the largest one for example we get an error of the order of $1.0e-6$. (To measure the error we calculate for each eigenvalue λ the determinant of $(A-\lambda I)$. A test that checks the eigenvalues (confronting them with the one we get with numpy) could be found in `testex3b.py`

3.3 Part c

The code for this section can be found on `ex3c.py`. The function `scriptc(A)`, just follows the instructions first given in c, while `scriptcmodified`, computes also the concatenation of the arrays as asked (I stored the $|T_{kk-1}$ in lists though, to make the concatenations easier.) Plotting what I get for the 5x5 matrix A generated previously I get the image (figure 3). For 4x4 matrix A, generated in the same way I get the image in figure 4. Finally in figure 6 we have the case for the same type of matrix (so with the entries again equal to $a_{ij} = \frac{1}{1+j+i}$) we have the figure 5. We are using a log scale in the plotting. If we don't consider a logarithmic scale (for example for a matrix A with the same structure and dimension 7x7) we get figure 6.
Observing all this figures we can observe that the plot seems a sort of sawtooth. This is why the 'tolerance' $|T_{kk-1}|$ keeps decreasing until we reach $1e-12$ and then the loop in QR stops, that's why the plot stops decreasing, and then starts decreasing again and we have this sawtooth form.

Also a test `testex3c.py`, that checks the correctness of eigenvalues is provided.
If we compare this algorithm with the one of the previous section we can see using timing and trying to do some checks that the previous algorithm is much faster. However it is quite more precise than the previous one, even if for the largest eigenvalue the error is still larger than the one for the small one

3.4 Part d

The script to consider is `ex3d.py`, we have to put `case3 = True`, to use the shifted version. For the same matrix A 5x5 we had before we get the figure. We provide also the 4x4 (both those are in log scale)

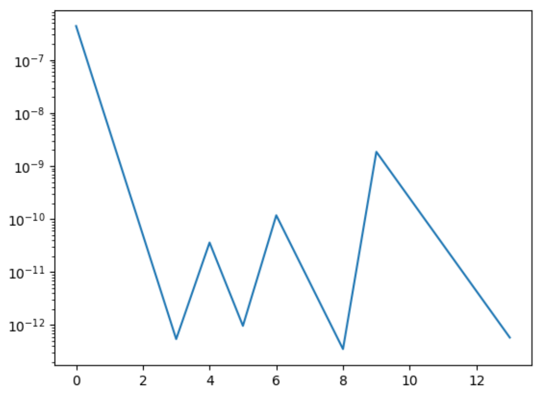

Figure 3: Matrix A 5x5 (ex 3c)

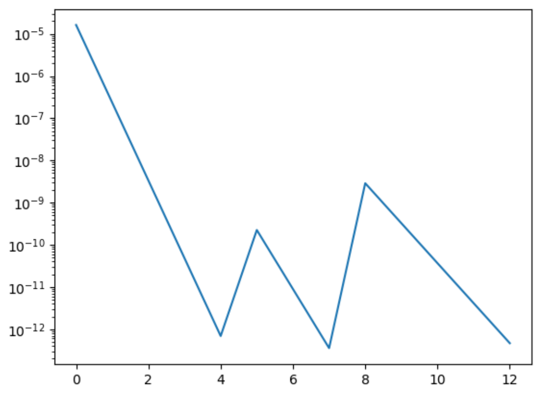

Figure 4: Matrix A 4x4 (ex 3c)

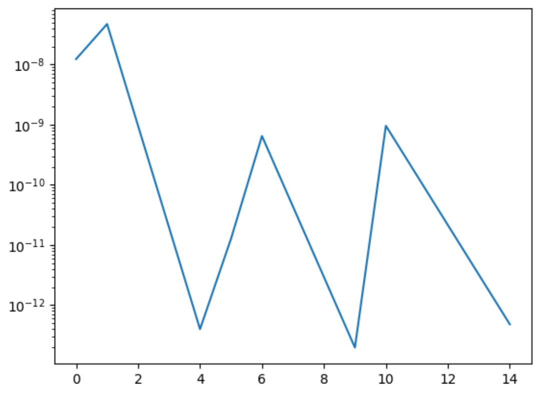

Figure 5: Matrix A 6x6 (ex 3c)

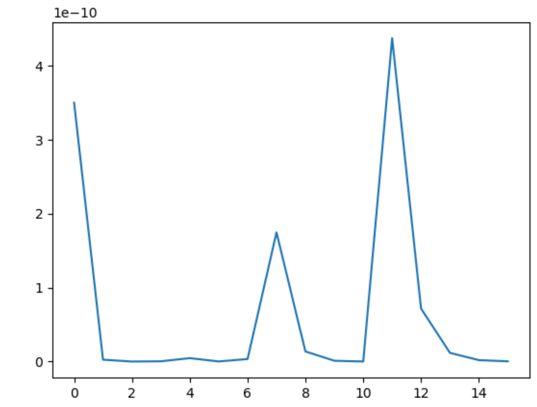

Figure 6: Matrix A 7x7 (not log scale) (ex 3c)

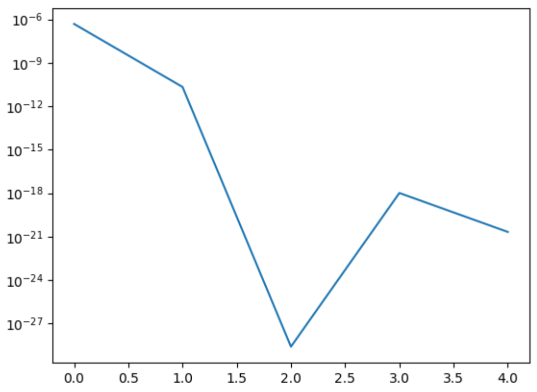

Figure 7: ex 3d A 5x5

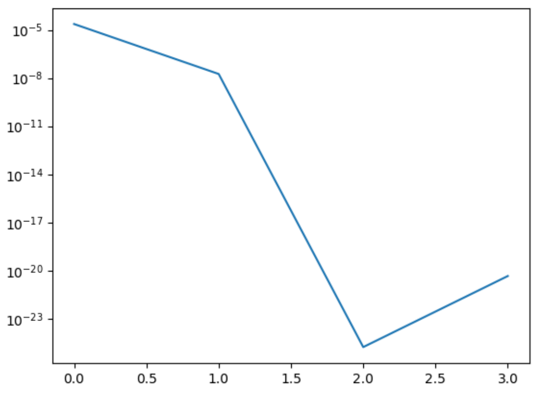

Figure 8: ex 3d A 4x4

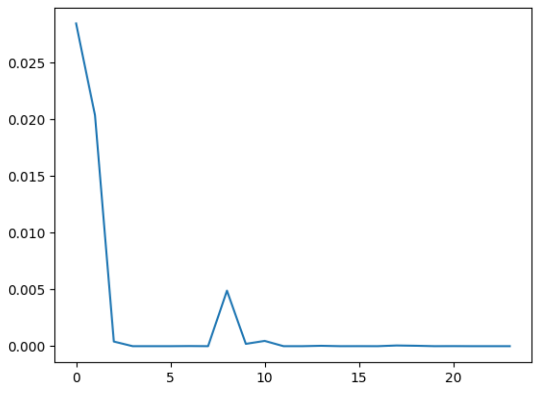

Figure 9: ex 3d 15x15(This is not for the matrix of exercise 3e)

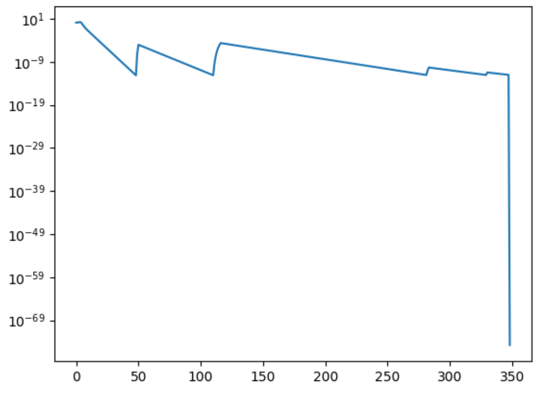

Figure 10: ex 3e 15x15 non shifted

We can see that this algorithm is way more efficient and takes less steps.
Just for curiosity we add another plot to see what happens for a 15x15 matrix in a not logged scale. A test to check the eigenvalues i provided in `testex3d.py`

3.5 Part e

The code for this is in `ex3e.py`. The plots we get are in figure 10 and 11, respectively for the non shifted and the shifted version. Is quite evident that the shifted version is way better and takes less iteration. (Both the plots are in log scale). Moreover comparing with the 5x5 matrix we can see that in both cases the shifted algorithm is better, but the difference is more substantial in the 15x15 one. This is due to the structure A = D + O, which is more diagonally dominant than the 5x5 one. (Running the code of this exercise, more than 2 plots are drawn, consider the last 2)

4 Exercise 4

4.1 Part a

The matrices may have different dimensions but the procedure is the same. The code can be found on `ex4a.py` and the test on `testex4a.py`

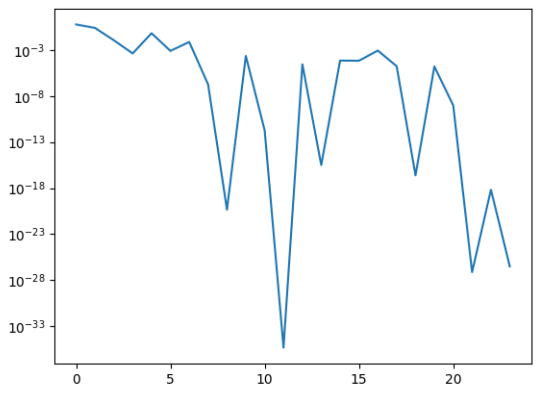

Figure 11: ex 3e 15x15 shifted

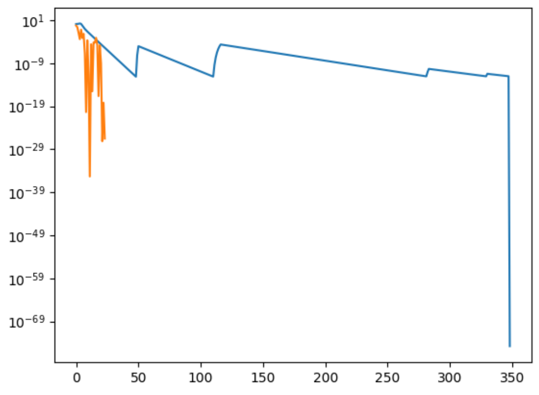

Figure 12: ex 3e 15x15 shifted and non shifted comparison

4.2 Part b

The code for this exercise can be found on `ex4b.py` and a test (which checks the sum over the indices of the negative Laplacian we get) can be found on `testex4b.py`

4.3 Part c

The modified GMRES code could be found on the script of `exercises10.py` (see `situation = None`. An automated test could be found on `testex4c.py`.

4.4 Part d

The code for this exercise could be found on `ex4d.py`

4.5 Part e

The code for this exercise could be found on `exercises10.py` (see function GMRES, check the option (`preconditoning = None`))