Project 2 Scientific Computing 2021

Kimi Cardoso Kreilgaard (twn176)

September 29, 2021

Questions for Week 3

a(1)

The function `gershgorin` in the attached .ipynb file locates the eigenvalues of a matrix using Gershgorin's theorem ([1] p. 164). It tells us that the eigenvalues of an $n \times n$ matrix $\mathbf{A}$ are all contained within the union of n disks, with the kth disk centered at a_{kk} and having radius $\sum_{j \neq k} |a_{kj}|$.
The function was tested on the example from p. 165 ([1]) as well as all the example matrices. In figure 1 below we can see the Gershgorin disks for the example matrix $\mathbf{A_6}$ along with the eigenvalues marked with crosses. As expected all eigenvalues are within the disks.

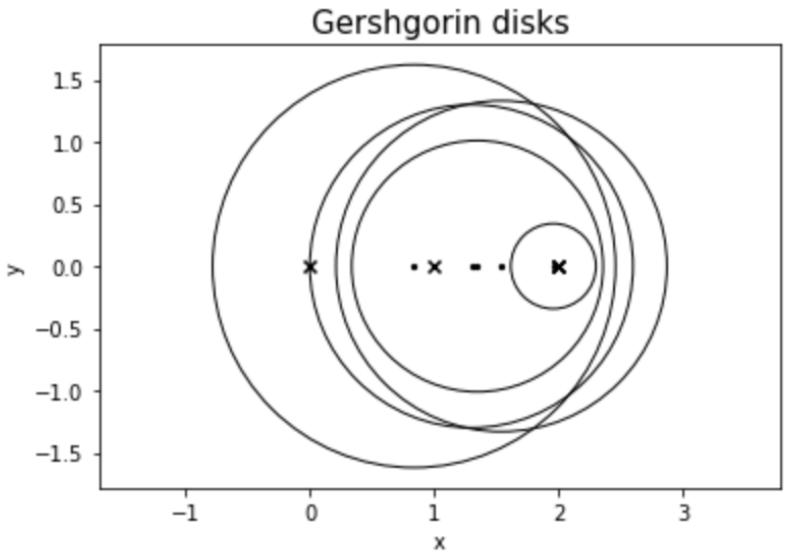

Figure 1: Gershgorin disks for the matrix A6. x represents the real part of the eigenvalue and y the imaginary part. Crosses mark eigenvalues.

a(2)

Using the function from a(1) on the $\mathbf{K}$ matrix we obtain 15 disks. The centers and radii are reported in the table 1 below.

Center	**Radius**	**Center**	**Radius**
129292.22	42231.65	13865.08	5502.97
103041.44	56927.52	5600.55	1195.28
64967.58	31160.67	1173.04	341.99
43612.41	18532.35	1760.77	401.21
36273.75	7870.52	288.42	71.42
37990.10	17854.70	86.90	2.54
24166.97	15414.72	13.89	0.26
11651.16	2512.04		

Table 1: Centers and radii for gershgorin disks of the $\mathbf{K}$ matrix.

b(1)

The function `rayleigh_qt` in the .py file takes a matrix $\mathbf{A}$ and an approximate eigenvector $\mathbf{x}$ and returns the approximate eigenvalue λ given by the Rayleigh quotient, defined as ([1] p. 177):

$$\lambda = \frac{\mathbf{x}^T \mathbf{A}\mathbf{x}}{\mathbf{x}^T \mathbf{x}} \tag{1}$$

The function was tested on all the example matrices. Using `numpy`'s linalg library I calculated the exact eigenvectors of the example matrix, and gave it as input to the function. The rayleigh quotient was equivalent with the actual eigenvalue in all cases. The calculations for $\mathbf{A_6}$ can be seen in the code file.

b(2)

I implemented the function `power_iterate`, which performs normalised power iteration by taking a matrix $\mathbf{A}$ and an initial vector $\mathbf{x_0}$ and returns an eigenvector $\mathbf{x}$ together with the number k of iterations used. I used the Rayleigh residual as the convergence criterion, defined as the residual of the Rayleigh quotient least squares problem. The least squares system is thus $\lambda\mathbf{x} \cong \mathbf{A}\mathbf{x}$, and the residual is: $||\mathbf{A}\mathbf{x} - \lambda\mathbf{x}||_2$. The residual is a measure of how close to obtaining an actual eigenvector we are, I therefor let the convergence criterion be:

$$||\mathbf{A}\mathbf{x} - \lambda\mathbf{x}||_2 < \epsilon \tag{2}$$

where ϵ is the error we allow in the eigenvector. If our eigenvector is a good approximation, we can expect the eigenvalue to be so to, as we have established when testing the `rayleigh_qt` function.

b(3)

We test the function `power_iterate` on the example matrices $\mathbf{A_1}$ to $\mathbf{A_6}$. In table 2 the largest eigenvalue found, the Rayleigh residual along with the number k of iterations used are reported for each of the matrices. In the code file a random seed was introduced so the results are reproducible when running the `.ipynb` file.

Matrix	Exact Largest Eigenvalue	Largest Eigenvalue (from power iteration)	Rayleigh Residual	Number of Iterations
$\mathbf{A_1}$	4.000000000000000	3.999999999999999	$9.19 \cdot 10^{-11}$	33
$\mathbf{A_2}$	4.000000000000000	4.000000000000000	$9.39 \cdot 10^{-11}$	30
$\mathbf{A_3}$	12.298958390970709	12.298958390959081	$5.41 \cdot 10^{-11}$	25
$\mathbf{A_4}$	16.116843969807043	16.116843969815530	$2.99 \cdot 10^{-11}$	10
$\mathbf{A_5}$	68.642080737002402	68.642080737005685	$8.42 \cdot 10^{-12}$	10
$\mathbf{A_6}$	2.000000000000000	2.000000000000000	$8.02 \cdot 10^{-11}$	29

Table 2: The real largest eigenvalue reported along with the results from the power iteration routine performed on each of the six example matrices. In all cases the bound was chosen to be $\epsilon = 10^{-10}$.

b(4)

The largest eigenvalue for the matrix $\mathbf{K}$ is found from power iteration to be $\lambda_k = 151362.6664880076$ with 66 iterations and a Rayleigh residual of $8.748 \cdot 10^{-9}$. Again the bound was chosen to be $\epsilon = 10^{-10}$ and a random seed was introduced for reproducible results. The obtained eigenfunction is visualized using the supplied library `chadni_show.py`. In the figure below respectively the waves and the nodes of the eigenfunction is visualized for the largest eigenvalue of the matrix $\mathbf{K}$. As the assignment predicted the nodes form an 8 by 8 grid.

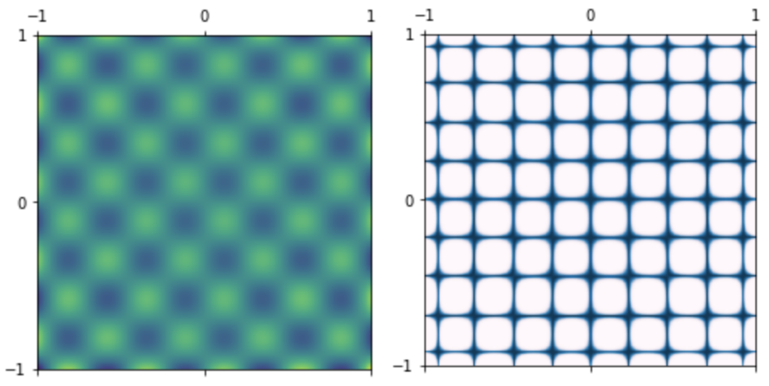

Figure 2: Visualising the eigenfunction's wave and its nodes.

c(1)

I implemented two Rayleigh quotient iteration functions by the names `lu_rayleigh_iterate` and `qr_rayleigh_iterate` which uses respectively LU- or QR-factorization to check which implementation is the most robust. Both functions follows algorithm 4.4 (p. 177 [1]), but differs in their way of solving the linear system to generate the next vector. Furthermore both functions utilizes the same convergence criterion as the power iteration due to the same reasoning as before.
After testing we see that the LU implementation imediately breaks down for singular matrices, and the QR implementation is therefor deemed the more robust one which we will use continuing. The QR implementation theoretically should break down at singular matrices too since it uses back substitution

where no diagonal element can be zero. To get around this we could use pivoting in both of the methods, but this is not implemented for now.

c(2)

We test the `qr_rayleigh_iterate` on the example matrices $\mathbf{A_1}$ to $\mathbf{A_6}$. Since the rayleigh iteration routine will find the eigenvalue closest to the shift we provide it with, I test it by randomly selecting a eigenvalue I aim to find, and then letting the input shift be a random number between -1 and 1 added to that eigenvalue. These values are reported in table 3 for each of the example matrices along with the results from the rayleigh quotient iteration: an eigenvalue, a rayleigh residual and the number of iterations k used.

Matrix	**Random Exact Eigenvalue Chosen to Aim for**	**Initial Eigenvalue Guess (Shift0)**	**Eigenvalue (from Rayleigh Quotient Iteration)**	**Rayleigh Residual**	**Number of Iterations**
$\mathbf{A_1}$	4.00000000000000	4.18568923645004	3.99999999999999	$3.53 \cdot 10^{-11}$	1
$\mathbf{A_2}$	2.00000000000000	1.84730959867781	2.00000000000000	$2.22 \cdot 10^{-16}$	2
$\mathbf{A_3}$	-0.95851013858639	-1.84508418395151	-0.95851013858632	$1.20 \cdot 10^{-12}$	3
$\mathbf{A_4}$	0.00000000000000	0.13608912218787	-0.00000000000000	$7.95 \cdot 10^{-16}$	2
$\mathbf{A_5}$	0.00000000000000	-0.26351692031890	-0.00000000000000	$5.83 \cdot 10^{-15}$	2
$\mathbf{A_6}$	1.00000000000000	1.35775906023792	1.00000000000000	$2.12 \cdot 10^{-16}$	4

Table 3: The eigenvalue we aimed to find with Rayleigh quotient iteration reported along with the shift given to `qr_rayleigh_iterate` and the results from the rayleigh quotient iteration. In all cases the bound was chosen to be $\epsilon = 10^{-10}$.

d(1)

As is described in the book ([1] p. 172-173) power iteration converges to the eigenvector with a corresponding eigenvalue that is of maximum modulus. This eigenvalue is called the dominant eigenvalue and is the only eigenvalue that we can find with the power iteration method. Thus we cannot get all eigenvalues and -vectors with pure power iteration.

d(2)

Since $\mathbf{K}$ is a 15×15 matrix we can expect to find 15 eigenvalues even though they might not all be unique. To see how many unique eigenvalues we can expect I use `numpy.eig` and `np.unique` on the matrix and find that there are 15 unique eigenvectors.
As could be expected we encounter problems with singular matrices for the matrix $\mathbf{K}$, I therefor adjust the `rayleigh_iterate` to use `numpy`'s solver instead. Using the centers of the gershgorin disks calculated in a(2) as the initial shifts in the Rayleigh quotient iteration I am able to find 14 unique eigenvalues. To find

the last unique eigenvalue I use the left edge of each disk (center - radius) as the initial shifts. The function `get_unique_vals` is used to identify the unique eigenvalues, since there might be some slight nummerical difference we can't use `np.unique` instead my function `get_unique_vals` is based on `np.isclose` which checks if two values are the same with some error tolerance. In the table 4 the obbtained eigenvalues from this method is listed along with the value we get from `np.eig` and the difference between the two.

Obtained Eigenvalue	**Exact Eigenvalue**	**Difference**
13.892607255232189	13.892607255232187	$1.776 \cdot 10^{-15}$
86.880216418106784	86.880216418106684	$9.948 \cdot 10^{-14}$
286.533306514492608	286.533306514493006	$3.979 \cdot 10^{-13}$
1132.216568065963202	1132.216568065963884	$6.821 \cdot 10^{-13}$
1799.805890216071930	1799.805890216073749	$1.819 \cdot 10^{-12}$
5560.881197518919180	5560.881197518918270	$9.095 \cdot 10^{-13}$
11485.212833611098176	11485.212833611101814	$3.638 \cdot 10^{-12}$
13338.622299303438922	13338.622299303426189	$1.273 \cdot 10^{-11}$
22590.198519599907740	22590.198519599907740	0.000
32779.071083579518017	32779.071083579518017	0.000
36152.369974537374219	36152.369974537410599	$3.638 \cdot 10^{-11}$
50430.027654187193548	50430.027654187164444	$2.910 \cdot 10^{-11}$
52766.288757713831728	52766.288757713795349	$3.638 \cdot 10^{-11}$
93999.614128836488817	93999.614128836488817	0.000
151362.666488007758744	151362.666488008049782	$2.910 \cdot 10^{-10}$

Table 4: Eigenvalues of the matrix $\mathbf{K}$ found with Rayleigh quotient iteration with initial shifts determined from centers and radii of the Gersgorin disks. Eigenvalues found with `numpy` are also reported and the difference between corresponding eigenvalues are reported.

Using the `show_nodes` function we can visualize the the eigenfunction with the lowest eigenvalue and check that it looks like a cross, as is apparent in figure 3 below.

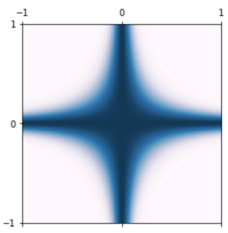

Figure 3: Visualising the nodes of the lowest eigenfunction.

d(3)

We check that $\mathbf{K} = \mathbf{T}\Lambda\mathbf{T}^{-1}$, where $\mathbf{T}$ is the transformation matrix whose columns are eigenvectors in order of ascending eigenvalues and Λ is diagonal matrix with the ascending eigenvalues on the diagonal. I check this by confirming that $\mathbf{K} - \mathbf{T}\Lambda\mathbf{T}^{-1} = \mathbf{0}$, which can be seen in the code. To visualise it here I use `plt.imshow` to visualise the numbers with a colorbar.

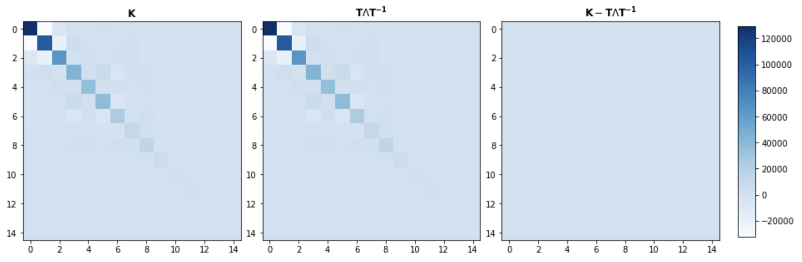

Figure 4: Visualising the nodes of all eigenfunction.

d(4)

We can visualize the nodes of all eigenfunctions found, this is seen in figure 5 below.

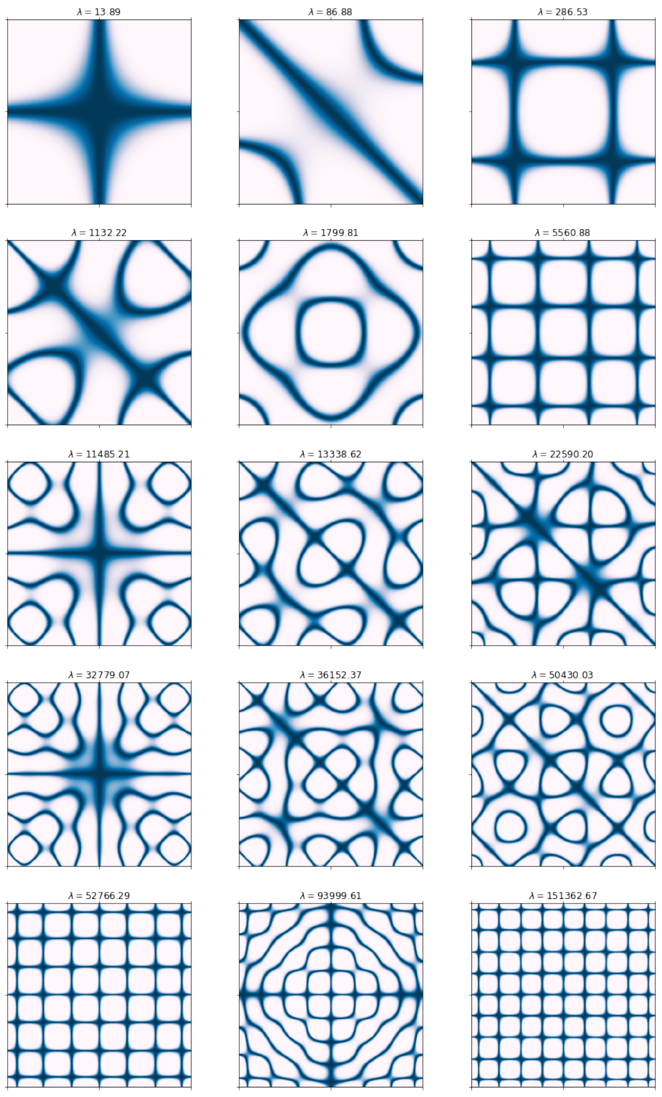

Figure 5: Visualising the nodes of all eigenfunction.

References

[1] Michael T. Heath, *Scientific Computing: An Introductory Survey*, 2nd Ed.