Computational Linear Algebra CW1

JIAQI LI

11th January 2023

1 Question2

1.1 question a

Without loss of generality for example, I take n = 3 for example and take 2 decimal places:

$$A^{(k)} = \begin{pmatrix} 0 & 1.8 & 0 & 0 & 0 & 0 \\ -1.8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.25 & 0 & 0 \\ 0 & 0 & -1.25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.45 \\ 0 & 0 & 0 & 0 & 0.45 & 0 \end{pmatrix}$$

We can find that this $A^{(k)}$ is not a upper triangular matrix, but in fact QR algorithm will turn out to be an iterative algorithm that converges to the diagonal matrix (upper triangular matrix for the general non-symmetric case) that A is similar to, and the eigenvalues can be found in the diagonal of $A^{(k)}$ which is similar to A.
By computing the eigenvalues in several cases A in different size, these $2n \times 2n$ matrix always has n groups of eigenvalues, each group has 2 eigenvalue with the same absolute value, by this I can find that not all the absolute values of eigenvalues are distinct, so $A^{(k)}$ does not converges to an upper triangular form. So we can not read off the eigenvalues using our original pure_QR method directly.

1.2 question b

Without loss of generality for example, I take n = 3 for example and take 2 decimal places:

$$A^{(k)} = \begin{pmatrix} 0 & 1.8 & 0 & 0 & 0 & 0 \\ -1.8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.25 & 0 & 0 \\ 0 & 0 & -1.25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.45 \\ 0 & 0 & 0 & 0 & 0.45 & 0 \end{pmatrix}$$

And we know that $A^{(k)}$ is similar with A because $A^{(k)}$ is the result after applying pure QR, so $A^{(k)}$ has the same eigenvalues with A, then I will compute the eigenvalues of $A^{(k)}$.

Use the characteristic polynomial method to calculate the eigenvalues which means by $\det(A-\lambda I) = 0$ I can compute the eigenvalues.

$$A^{(k)} - \lambda I = \begin{pmatrix} -\lambda & 1.8 & 0 & 0 & 0 & 0 \\ -1.8 & -\lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & -\lambda & 1.25 & 0 & 0 \\ 0 & 0 & -1.25 & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & -\lambda & 0.45 \\ 0 & 0 & 0 & 0 & 0.45 & -\lambda \end{pmatrix}$$

So we can find that this is a block diagonal then I can compute the eigenvalues by the following method:

$$\det(\begin{pmatrix} A_1 & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & A_2 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & A_3 & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & A_n \end{pmatrix}) = \det(A1)\det(A2)\det(A3)...\det(An)$$

So I can get that

$$\begin{aligned} \det(A^{(k)} - \lambda I(&= \det(\begin{pmatrix} -\lambda & 1.8 & 0 & 0 & 0 & 0 \\ -1.8 & -\lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & -\lambda & 1.25 & 0 & 0 \\ 0 & 0 & -1.25 & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & -\lambda & 0.45 \\ 0 & 0 & 0 & 0 & 0.45 & -\lambda \end{pmatrix}) \\ &= \det(\begin{pmatrix} -\lambda & 1.8 \\ -1.8 & -\lambda \end{pmatrix})\det(\begin{pmatrix} -\lambda & 1.25 \\ -1.25 & -\lambda \end{pmatrix})\det(\begin{pmatrix} -\lambda & 0.45 \\ 0.45 & -\lambda \end{pmatrix}) \\ &= (\lambda^2 + 1.8^2)(\lambda^2 + 1.25^2)(\lambda^2 + 0.45^2) \\ &= 0 \end{aligned} \tag{1}$$

So $\lambda = 1.8i, -1.8i, 1.25i, -1.25i, 0.45i, -0.45i$. which is equal to 1i $\times$ the nonzero value of upper and lower secondary diagonals of $A^{(k)}$.
By this in general n, $A^{(k)}$ is in size $2n \times 2n$, we can get that:

$$A^{(k)} = \begin{bmatrix} \boxed{\begin{matrix} 0 & a_1 \\ a_1 & 0 \end{matrix}} & \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} & & \\ \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} & \boxed{\begin{matrix} 0 & a_2 \\ a_2 & 0 \end{matrix}} & & \\ & & \ddots & \\ & & & \boxed{\begin{matrix} 0 & a_n \\ a_n & 0 \end{matrix}} \end{bmatrix} \longrightarrow A^{(k)} - \lambda I = \begin{bmatrix} \boxed{\begin{matrix} -\lambda & a_1 \\ a_1 & \lambda \end{matrix}} & \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} & & \\ \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} & \boxed{\begin{matrix} -\lambda & a_2 \\ a_2 & -\lambda \end{matrix}} & & \\ & & \ddots & \\ & & & \boxed{\begin{matrix} -\lambda & a_n \\ a_n & -\lambda \end{matrix}} \end{bmatrix}$$

$$\swarrow$$

$$\det(A^{(k)} - \lambda I) = \begin{vmatrix} -\lambda & a_1 \\ a_1 & -\lambda \end{vmatrix} \begin{vmatrix} -\lambda & a_2 \\ a_2 & -\lambda \end{vmatrix} \cdots \begin{vmatrix} -\lambda & a_n \\ a_n & -\lambda \end{vmatrix}$$

$$\downarrow$$

$$\lambda = a_1 i, -a_1 i, a_2 i, -a_2 i, \cdots, a_n i, -a_n i$$

Algorithm:
1. Take the upper and lower secondary diagonals of $A^{(k)}$ and store them as 2 arrays.
2. Add a zero element into the array of lower secondary diagonal in order to collect the element in upper secondary diagonal.
3. Put the i^{th} element in the array of lower secondary diagonal into the $i+1^{th}$ position in the array of lower secondary diagonal by for loop.

1.3 question c

Function and Automated testing:
The code are shown in cw3.CW3.py by function pure_QR_A_ev.
The automated testing are shown in test_cw3.test_coursework3.py by function testpure_QR_A_ev.
Numerical experiments for different values of n 1. When n = 2 (in 2 decimal places):

$$A^{(k)} = \begin{pmatrix} 0 & 1.62 & 0 & 0 \\ -1.62 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.62 \\ 0 & 0 & -0.62 & 0 \end{pmatrix}$$

By the function pure_QR_A_ev that I write in 2(c) I can find that

$$\lambda_{A^{(k)}} = 1.62i, -1.62i, 0.62i, -0.62i.$$

By the function np.linalg.eig in numpy package I can find that

$$\lambda_A = 1.62i, -1.62i, 0.62i, -0.62i.$$

I can find that $\lambda_{A^{(k)}} = \lambda_A$ when take 2 decimal places, so they are in the same values.
2. When n = 3 (in 2 decimal places):

$$A^{(k)} = \begin{pmatrix} 0 & 1.8 & 0 & 0 & 0 & 0 \\ -1.8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.25 & 0 & 0 \\ 0 & 0 & -1.25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.45 \\ 0 & 0 & 0 & 0 & 0.45 & 0 \end{pmatrix}$$

By the function pure_QR_A_ev that I write in 2(c) I can find that

$$\lambda_{A^{(k)}} = 1.8i, -1.8i, 1.25i, -1.25i, 0.45i, -0.45i.$$

By the function np.linalg.eig in numpy package I can find that

$$\lambda_A = 1.8i, -1.8i, 1.25i, -1.25i, 0.45i, -0.45i.$$

I can find that $\lambda_{A^{(k)}} = \lambda_A$ when take 2 decimal places, so they are in the same values.
3. When n = 4 (in 2 decimal places):

$$A^{(k)} = \begin{pmatrix} 0 & 1.88 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1.88 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.53 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.53 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.35 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.35 & 0 \end{pmatrix}$$

By the function pure_QR_A_ev that I write in 2(c) I can find that

$$\lambda_{A^{(k)}} = 1.88i, -1.88i, 1.53i, -1.53i, 1i, -1i, 0.35i, -0.35i.$$

By the function np.linalg.eig in numpy package I can find that

$$\lambda_A = 1.88i, -1.88i, 1.53i, -1.53i, 1i, -1i, 0.35i, -0.35i.$$

I can find that $\lambda_{A^{(k)}} = \lambda_A$ when take 2 decimal places, so they are in the same values.

1.4 question d

Function and Automated testing:
The code are shown in cw3.CW3.py by function pure_QR_A_ev_new.
The automated testing are shown in test_cw3.test_coursework3.py by function testpure_QR_A_ev_new.

Numerical experiments for different values of n Fix $maxit = 5000$ and $tol = 1.0 \times 10^{-5}$. 1. When n = 2 (in 9 decimal places):

$$A^{(k)} = \begin{pmatrix} 0 & 2.45787612 & 0 & -0.55708601 \\ -2.13032217 & 0 & 1.11417203 & 0 \\ 0 & 0 & 0 & 1.62742132 \\ 0 & 0 & -0.46941257 & 0 \end{pmatrix}$$

By the function pure_QR_A_ev_new that I write in 2(d) which write all eigenvalues in an array, I can find that (in 9 decimal places)

$$\lambda_{A^{(k)}} = [2.28824561i, -2.28824561i, 0.87403205i, -0.87403205i]$$

By the function np.linalg.eig in numpy package which write all eigenvalues in an array I can find that (in 9 decimal places)

$$\lambda_A = [2.28824561i, -2.28824561i, 0.87403205i, -0.87403205i]$$

I can find that in 9 decimal places $\lambda_{A^{(k)}} = \lambda_A$, so they are in the same values.
2. When n = 3:
Write $A^{(k)}$ in 3 decimal places:

$$A^{(k)} = \begin{pmatrix} 0 & 2.646 & 0 & 0.475 & 0 & 0.098 \\ -2.454 & 0 & -0.94 & 0 & 0.24 & 0 \\ 0 & 0 & 0 & 2.035 & 0 & 0.643 \\ 0 & 0 & -1.528 & 0 & -1.277 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.485 \\ 0 & 0 & 0 & 0 & -0.267 & 0 \end{pmatrix}$$

By the function pure_QR_A_ev_new that I write in 2(d) which write all eigenvalues in an array, I can find that (in 9 decimal places)

$$\lambda_{A^{(k)}} = [2.54832479i, -2.54832479i, 1.76349547i, -1.76349547i, 0.62938425i, -0.62938425i].$$

By the function np.linalg.eig in numpy package which write all eigenvalues in an array I can find that (in 9 decimal places)

$$\lambda_A = [2.54832479i, -2.54832479i, 1.76349547i, -1.76349547i, 0.62938425i, -0.62938425i].$$

I can find that in 9 decimal places $\lambda_{A^{(k)}} = \lambda_A$, so they are in the same values.
3. When n = 4:
Write $A^{(k)}$ in 3 decimal places:

$$A^{(k)} = \begin{pmatrix} 0 & 2.73 & 0 & -0.427 & 0 & -0.03 & 0 & -0.062 \\ -2.588 & 0 & 0.774 & 0 & 0.342 & 0 & 0.185 & 0 \\ 0 & 0 & 0 & 2.322 & 0 & -0.584 & 0 & 0.196 \\ 0 & 0 & -2.021 & 0 & 1.169 & 0 & 0.14 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.773 & 0 & -0.645 \\ 0 & 0 & 0 & 0 & -1.128 & 0 & 1.335 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.423 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.17 & 0 \end{pmatrix}$$

By the function pure_QR_A_ev_new that I write in 2(d) which write all eigenvalues in an array, I can find that (in 9 decimal places)

$\lambda_{A^{(k)}} = [2.6578521i, -2.6578521i, 2.16670088i, -2.16670088i, 1.41421356i, -1.41421356i, 0.49115122i, -0.49115122i]$.

By the function np.linalg.eig in numpy package which write all eigenvalues in an array I can find that (in 9 decimal places)

$\lambda_A = [2.6578521i, -2.6578521i, 2.16670088i, -2.16670088i, 1.41421356i, -1.41421356i, 0.49115122i, -0.49115122i]$.

I can find that in 9 decimal places $\lambda_{A^{(k)}} = \lambda_A$, so they are in the same values.

2 Question3

2.1 question a

If A is a symmetric and tri-diagonal matrix, then after applying QR factorization Q will be a matrix with lower bandwidth 1, and R will be a matrix with upper bandwidth 2.
Without loss of generality for example, I take 6×6 matrix for example:
We know that A = QR by factorization, and show it by matrix,

$$A = \begin{bmatrix} \bullet & \bullet & & & & \\ \bullet & \bullet & \bullet & & & \\ & \bullet & \bullet & \bullet & & \\ & & \bullet & \bullet & \bullet & \\ & & & \bullet & \bullet & \bullet \\ & & & & \bullet & \bullet \end{bmatrix} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ & \bullet & \bullet & \bullet & \bullet & \bullet \\ & & \bullet & \bullet & \bullet & \bullet \\ & & & \bullet & \bullet & \bullet \\ & & & & \bullet & \bullet \end{bmatrix} \begin{bmatrix} \bullet & \bullet & \bullet & & & \\ & \bullet & \bullet & \bullet & & \\ & & \bullet & \bullet & \bullet & \\ & & & \bullet & \bullet & \bullet \\ & & & & \bullet & \bullet \\ & & & & & \bullet \end{bmatrix} = QR$$

Then I will apply pure_QR, $A^{(k)} = RQ$ should be in Hessenberg form which is shown below:

$$RQ = \begin{bmatrix} \bullet & \bullet & \bullet & & & \\ & \bullet & \bullet & \bullet & & \\ & & \bullet & \bullet & \bullet & \\ & & & \bullet & \bullet & \bullet \\ & & & & \bullet & \bullet \\ & & & & & \bullet \end{bmatrix} \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ & \bullet & \bullet & \bullet & \bullet & \bullet \\ & & \bullet & \bullet & \bullet & \bullet \\ & & & \bullet & \bullet & \bullet \\ & & & & \bullet & \bullet \end{bmatrix} = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ & \bullet & \bullet & \bullet & \bullet & \bullet \\ & & \bullet & \bullet & \bullet & \bullet \\ & & & \bullet & \bullet & \bullet \\ & & & & \bullet & \bullet \end{bmatrix} = A^{(k)}$$

Also we have $A^{(k)} = RQ = IRQ = Q^TQRQ = Q^TAQ$, and A is symmetric, then $A^T = A$, so we can get that:

$$\begin{aligned} {A^{(k)}}^T &= (Q^TAQ)^T \\ {A^{(k)}}^T &= Q^TA^T(Q^T)^T \\ {A^{(k)}}^T &= Q^TA^TQ = A^{(k)} \end{aligned} \tag{2}$$

So I can find that $A^{(k)}$ is also a symmetric matrix, so the step of QR algorithm will preserve the symmetric tri-diagonal matrix.

2.2 question b

Modification of pure_QR code and auto-test
I modify my code of function pure_QR in cla_utils.exercise9.
To create matrix A which satisfies the condition $A_{i,j} = \frac{1}{i+j+1}$ I write the function create_Aij in cw3.CW3.py.
To test my code I create a function test_pure_QR_cw3 in test_cw3.test_coursework3.py.
Apply to an example
Apply my program to the 5×5 matrix $A_{ij} = \frac{1}{i+j+1}$, I find that the approximate eigenvalues I can read off by the diagonal of matrix after applying pure_QR and I can show the approximate eigenvalues in an array:

$$ev1 = [1.56705061e, 2.08534301 \times 10^{-1}, 1.14074916 \times 10^{-2}, 3.05898040 \times 10^{-4}, 3.28792877 \times 10^{-6}]$$

and I can also find that the exact eigenvalues of matrix A which satisfies the property $A_{i,j} = \frac{1}{i+j+1}$, and I also show the exact eigenvalues in an array:

$$ev2 = [1.56705069, 2.08534219 \times 10^{-1}, 1.14074916 \times 10^{-2}, 3.05898040 \times 10^{-4}, 3.28792877 \times 10^{-6}]$$

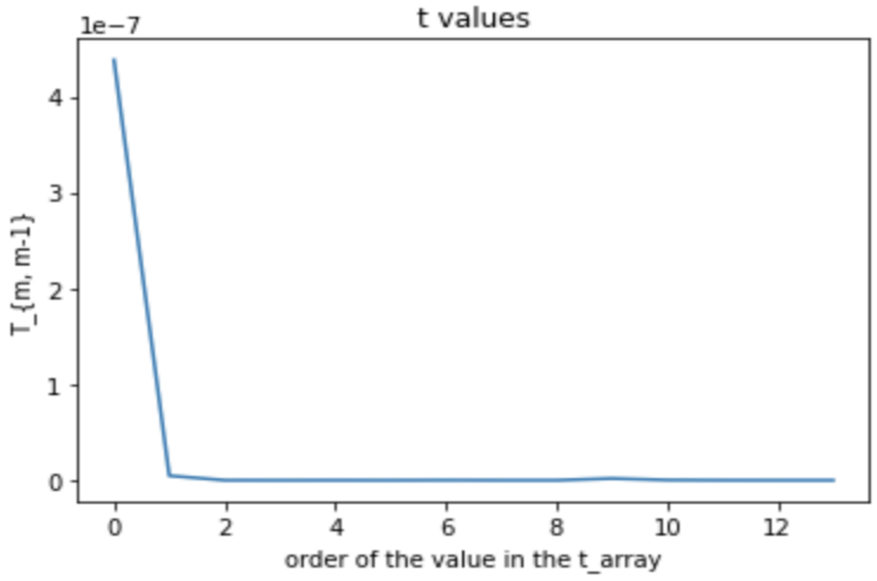
So the error between them is

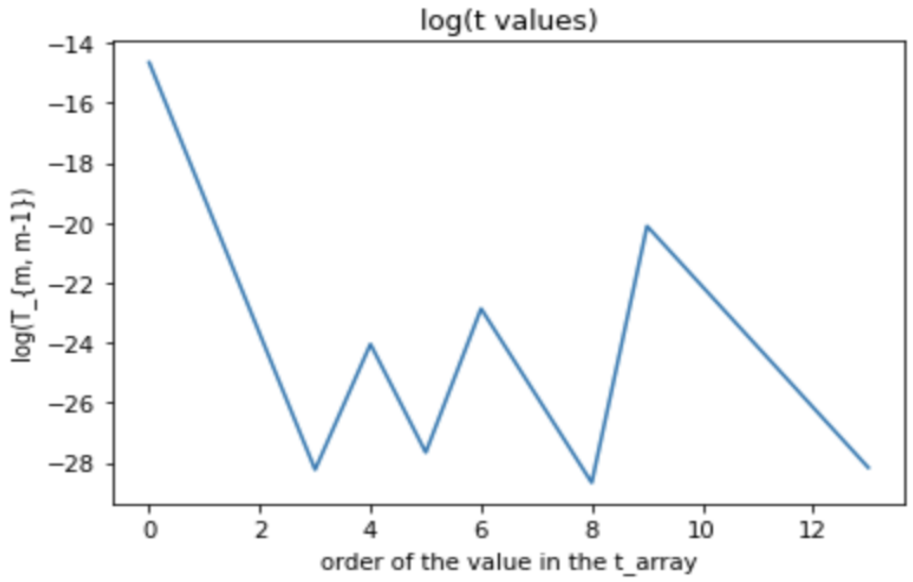
$$error = [8.20858967 \times 10^{-8}, -8.20832709 \times 10^{-8}, -2.6254085 \times 10^{-12}, -1.62792956 \times 10^{-16}, -8.47032947 \times 10^{-22}]$$

By this I can find that all the eigenvalues are converge in this case.

2.3 question c

Code
I write the function ev_3c_check in cw3.CW3.py to write down an array of eigenvalues and an array of t values.
Modified pure_QR: I modify my code of function pure_QR in cla_utils.exercise9.
Test: To test my code I create a function test_pure_QR_cw3c in test_cw3.test_coursework3.py.
Graph I use the plt.plot to draw the graph and output the graph by using jupyter qtconsole in visual studio.



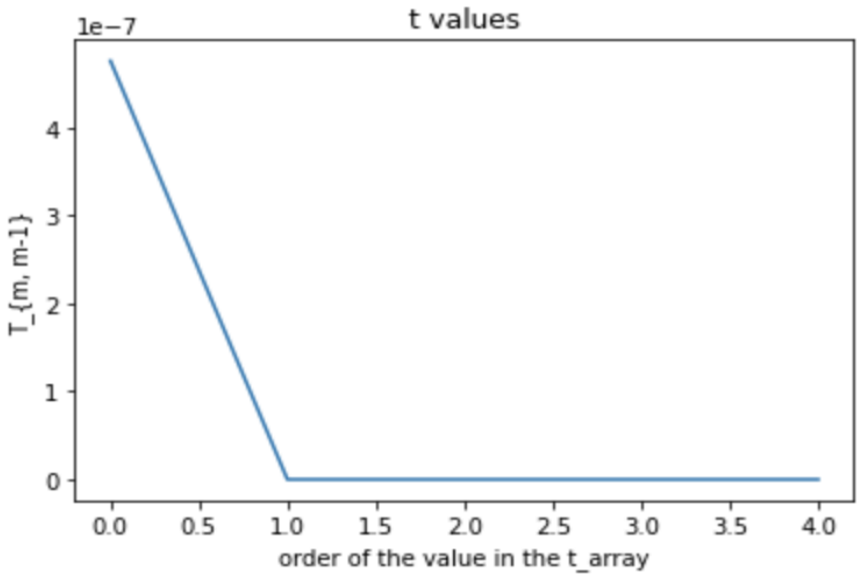


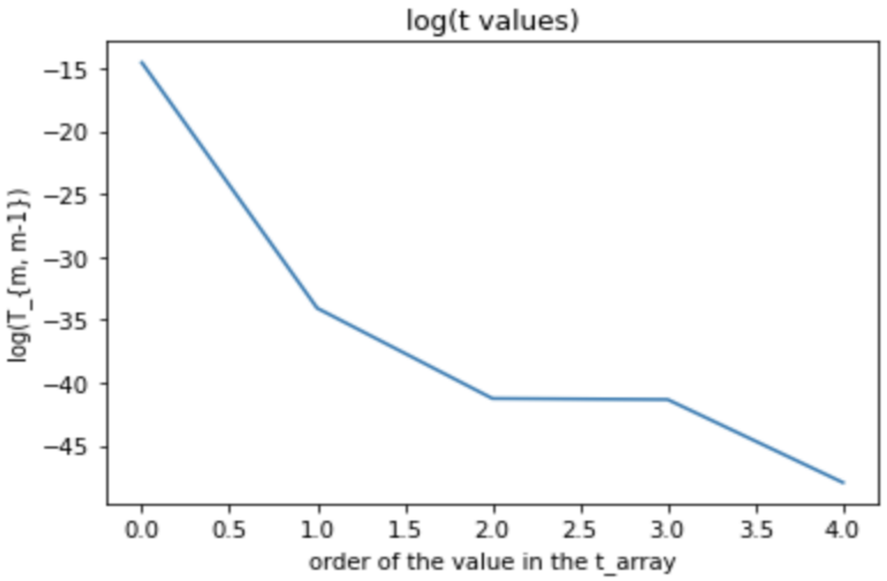
Describe, Comparison and Explain
t values represent the errors, in the graph I can find that the error drops quickly, because this method makes $|T_{m,m-1}|$ converges to 0 quickly.
So compared with the unmodified QR algorithm, it is more efficient, and because in method in c part t values converge to 0 quickly, I can find that the eigenvalues calculated in this way is more accurate than that calculated by unmodified QR algorithm.
We use the modified pure_QR which is introduced in (b), so we can do the symmetric matrix QR

in its special case, which makes the algorithm faster and more accurate. In addition, in each loop in c algorithm, I replace T with the $(k-1) \times (k-1)$ submatrix of T which consisting of the first k 1 rows and columns of T, by this we there will be less operation count and we can avoid more error of the other entries.

2.4 question d

Code
I also use the function ev_3c_check in cw3.CW3.py that I defined in 3(c) to write down an array of eigenvalues and an array of t values.
Modified pure_QR: I modify my code of function pure_QR in cla_utils.exercise9.
Test: To test my code I create a function test_pure_QR_cw3d in test_cw3.test_coursework3.py.
Graph I use the plt.plot to draw the graph and output the graph by using jupyter qtconsole in visual studio.





Comparison(Compare my plots of values of $|T_{m,m-1}|$, including for the 5×5 matrix from previous steps):
I can find that the t value which is calculated with shifts is always smaller than the t value which is calculated without shift.
And in the third and the fourth positions, t value is similar which means the t values in the same position is too close, but t value which is calculated with shifts is still smaller than the t value which is calculated without shift.
Besides, I can also find by the graph that the length of array of t in (d) which is corresponding to shifts is 5, but the length of array of t in (c) which is corresponding to no shift is 14, by this I can get that more eigenvalues are convergence when I use the modified pure_QR which is corresponding to shifts, and the eigenvalues can get into convergence more quickly when I use the modified pure_QR which is corresponding to shifts.
By taking the log of T, I can draw a graph and find that the log(t value) of (c) has more obvious fluctuation than the log(t value) of (d), while there is almost no fluctuation in the picture of (d), showing a trend of steady decline until they converge.
In addition, At the beginning, before we doing pure_QR algorithm the t value which is calculated with shifts is larger than the t value which is calculated without shift which is the first t value in each array.
Reason
We get μ when we modify the pure_QR algorithm with shifts, which can help me to predict the eigenvalue, which means we can make it to a matrix closer to eigenvalues to A, so this method with shifts will be more efficient and more accurate.

2.5 question e

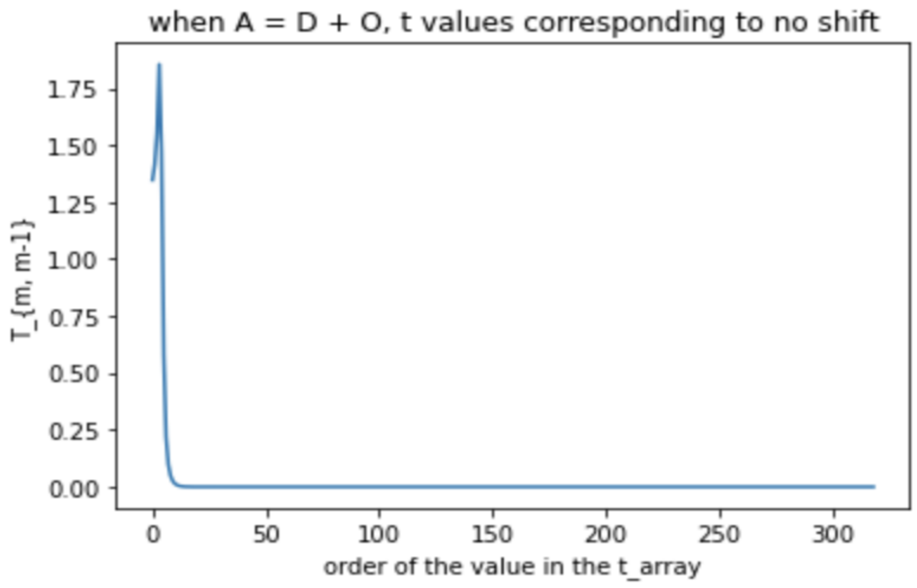
Code
I write the function ev_3e_check in cw3.CW3.py to write down an array of eigenvalues and an array

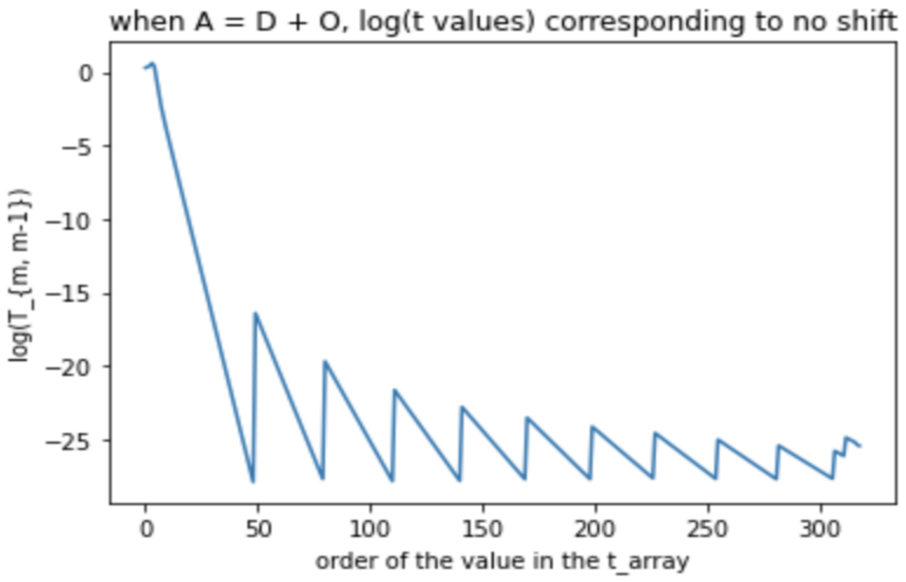
of t values.

Graph

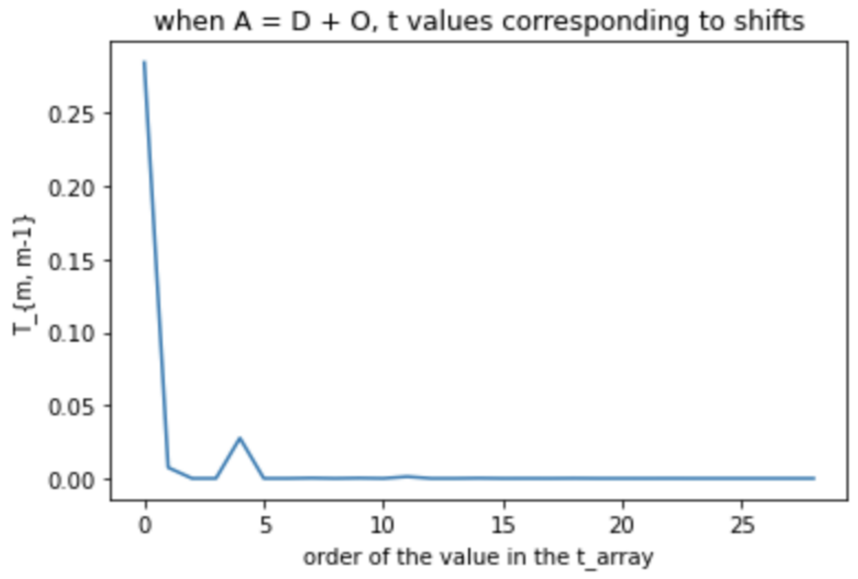
I use the plt.plot to draw the graph and output the graph by using jupyter qtconsole in visual studio.

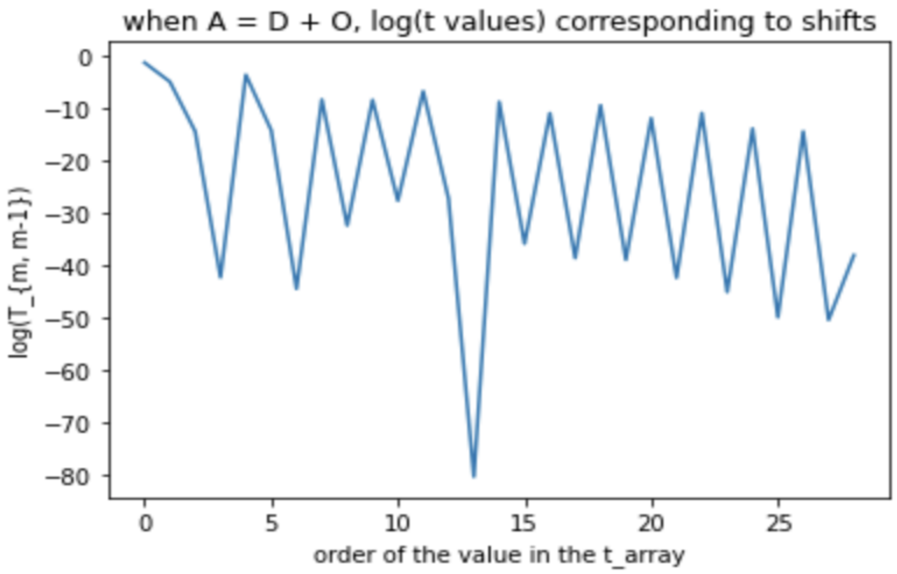
1. Graph corresponding to no shift





2. graph corresponding to shifts





Observation of comparison
Compare with the 5 matrix in (c) and (d), I can find that there are more t values here and the error is larger here. Because the size of matrix is larger in this pattern, so there will be more iterations to get into convergence, and in the same position of t array, the t value will be larger in the larger size of matrix, because the error that need to be decreased is larger here.
The sawtooth graph is very clear for the normal plot axis. The valley of the line shows the step when I can find $|T_{k,k-1}| < tol$, then I can proceed to the next step. In addition, the peak of the

line shows every time that I begin doing with a new $|T_{k,k-1}|$ after I replacing T with the submatrix of T which is in $(k-1) \times (k-1)$ size.
Then look at the graph of log(t values), as we do more iterations, the values of t keeps decreasing each time, and after several iterations, the value of the top left entry of sub T will be very small during other iterations, although I have not reached it. And this is the reason why I will find a flat line at the end of the normal graph, and also I can find a plummet at the ending of the log plot. This means that before applying the pure_QR, $|T_{k,k-1}|$ is already be decreased to a very small value which is smaller than tolerance. In addition the operation count here will also be very small, so this is a efficient and accurate method.

3 Question4

3.1 question a

I write the function u_to_v_wb in cw3.CW3.py to serialise normal two-dimensional matrix u row by row.
To test my code I create a function test_u_to_v_wb in test_cw3.test_coursework3.py.
I write the function dx_to_v in cw3.CW3.py to serialise 2d matrix dx with size $(n+1) \times n$ (different dimensions) into 1d array row by row.
To test my code I create a function test_dx_to_v in test_cw3.test_coursework3.py.
I write the function dy_to_v in cw3.CW3.py to serialise 2d matrix dx with size $n \times (n+1)$ (different dimensions) into 1d array row by row.
To test my code I create a function test_dy_to_v in test_cw3.test_coursework3.py.
I write the function v_to_u_wb in cw3.CW3.py to inverse from 1d array into the normal two-dimensional array of u.
To test my code I create a function test_v_to_u_wb in test_cw3.test_coursework3.py.
I write the function v_to_d in cw3.CW3.py to inverse from 1d array into the 2d matrix with different sizes (different dimensions dx or dy be $(n+1) \times n$ or $n \times (n+1)$).
To test my code I create a function test_v_to_d in test_cw3.test_coursework3.py.

3.2 question b

I write the function H_apply in cw3.CW3.py.
To test my code I create a function test_H_apply in test_cw3.test_coursework3.py.

3.3 question c

I modify my code of function GMRES in cla_utils.exercise10.
To test my code I create a function test_GMRES_Afunction in test_cw3.test_coursework3.py.

3.4 question d