Computational Linear Algebra CW3

nk1719

January 2022

1

Note: test for my code can be ran by pytest cw3/coursework3qi.py where i is the question number.

1.1

We observe that that our outputs follow the same structure as the matrix A itself. The output matrices are also non-zero only on the offset diagonals one below and one above. Further, these two diagonals have the same entries in the same indexes along the diagonal except that the below diagonal is the negative of the above one, also like in the input matrix A. We also notice that in each offset diagonal we see a 0 every second entry, this decreases the number of non-zero entries nearly two fold and means that the resulting output matrix has only one entry per column and one entry per row which is non-zero. The preservation of this structure is due to the complex eigenvalues of the initial matrix meaning for all future matrices we have the same complex eigenvalues and so the same structure, if this structure were not preserved the matrix would have some real eigenvalue which we know is not true. This means A^k never converges to a triangular matrix meaning we cannot read off the eigenvalues from the diagonal.

1.2

We see that in the pure QR algorithm our A^k, after a large number of iterations, converges to a matrix C of form

$$\begin{bmatrix} M_1 & & \\ & \ddots & \\ & & M_n \end{bmatrix} : M_k = \begin{bmatrix} 0 & x_k \\ -x_k & 0 \end{bmatrix}$$

where x_k are some real valued entries. Our method to find the eigenvalues of A is as follows:

- Compute $C = \text{pure\_qr}(A)$

- Compute eigenvalues for each M_k
- Due to M_k structure these eigenvalues are $\pm x_k i$
- All of these eigenvalues together are the eigenvalues of A

First we clearly show step 3

$$det(\begin{bmatrix} 0 & x_k \\ -x_k & 0 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}) = det(\begin{bmatrix} -\lambda & x_k \\ -x_k & -\lambda \end{bmatrix}) = \lambda^2 + {x_k}^2 = 0 \Rightarrow \lambda = \pm x_k i$$

Now. we explain the rest

$$det(\begin{bmatrix} M_1 & & \\ & \ddots & \\ & & M_n \end{bmatrix} - \lambda I) = det(\begin{bmatrix} \begin{bmatrix} -\lambda & x_1 \\ -x_1 & -\lambda \end{bmatrix} & & \\ & \ddots & \\ & & \begin{bmatrix} -\lambda & x_n \\ -x_n & -\lambda \end{bmatrix} \end{bmatrix})$$

by iterative use of the hint

$$= det(\begin{bmatrix} -\lambda & x_1 \\ -x_1 & -\lambda \end{bmatrix}) \dots det(\begin{bmatrix} -\lambda & x_n \\ -x_n & -\lambda \end{bmatrix}) = (\lambda^2 + {x_1}^2) \dots (\lambda^2 + {x_n}^2) = 0$$

$$\Rightarrow \lambda = \pm x_1 i, \dots \pm x_n i$$

We have now successfully shown that we can find the the eigenvalues of C. These are also the eigenvalues of A since we know from the pure QR algorithm that C and A are similar and similar matrices have the same eigenvalues. So in fact we can read off the eigenvalues of A from C straight away. They are simply all of the non-zero entries in C multiplied by i.

1.3

NOTE:The entries in C are rounded to 3 d.p to save space on the paper.
We will demonstarte with the example of $n = 3$. We obtain

$$A = \begin{bmatrix} 0. & 1. & 0. & 0. & 0. & 0. \\ -1. & 0. & 1. & 0. & 0. & 0. \\ 0. & -1. & 0. & 1. & 0. & 0. \\ 0. & 0. & -1. & 0. & 1. & 0. \\ 0. & 0. & 0. & -1. & 0. & 1. \\ 0. & 0. & 0. & 0. & -1. & 0. \end{bmatrix}$$

$$C = \begin{bmatrix} 0. & 1.802 & 0. & -0. & 0. & 0. \\ -1.802 & 0. & 0. & -0. & 0. & 0. \\ 0. & -0. & 0. & -1.247 & 0. & -0. \\ 0. & 0. & 1.247 & 0. & 0. & -0. \\ 0. & 0. & 0. & 0. & 0. & 0.445 \\ 0. & 0. & 0. & 0. & -0.445 & -0. \end{bmatrix}$$

We compute the eigenvalues of A using numpy.linalg.eigvals and receive $[-1.52624731e{-}166{+}1.80193774j, -1.52624731e{-}166{-}1.80193774j, 0.00000000e{+}000{+}0.44504187j, 0.00000000e{+}000{-}0.44504187j, 0.00000000e{+}000{+}1.2469796j, 0.00000000e{+}000 - 1.2469796j]$ which when removing the negligible values is equal to

$$[-0. + 1.802j, -0. - 1.802j, 0. + 0.445j, 0. - 0.445j, 0. + 1.247j, 0. - 1.247j]$$

which we can see corressponds to the non-zero values in C. The output from my implemented method gives

$$[0.{+}1.80193774j, -0.{-}1.2469796j, 0.{+}0.44504187j, -0.{-}1.80193774j, 0.{+}1.2469796j, -0.{-}0.44504187j]$$

1.4

When we apply pure QR to B we now get an upper triangular structure of 2×2 matrices instead of a diagonal one like with A. Further, in each block along the diagonal the top right entry is not the negative of the bottom left entry as with A. This means that the eigenvalues of each 2×2 block is not possible to read off but we have to do a slight computation.

$$det(\begin{bmatrix} -\lambda & a \\ b & -\lambda \end{bmatrix}) = \lambda^2 - ab = 0 \Rightarrow \lambda = \pm\sqrt{ab}$$

Since in each block a and b have opposite sign we again will get only complex eigenvalues. The upper triangular block structure does not affect calculating the eigenvalues of C as we know by iterative use of the hint that the determinant of a upper triangular block structure is also the product of the determinants of the diagonal blocks. And as with A the eigenvalues of B are those of C by similarity transformations in pure QR.
NOTE:The entries in C are rounded to 3 d.p to save space on the paper.
We will demonstrate with the example of $n = 3$. We obtain

$$B = \begin{bmatrix} 0. & 2. & 0. & 0. & 0. & 0. \\ -1. & 0. & 2. & 0. & 0. & 0. \\ 0. & -1. & 0. & 2. & 0. & 0. \\ 0. & 0. & -1. & 0. & 2. & 0. \\ 0. & 0. & 0. & -1. & 0. & 2. \\ 0. & 0. & 0. & 0. & -1. & 0. \end{bmatrix}$$

$$C = \begin{bmatrix} 0. & 2.646 & 0. & -0.475 & 0. & -0.098 \\ -2.454 & 0. & 0.94. & -0. & -0.24 & 0. \\ 0. & -0. & 0. & 2.035 & 0. & 0.643 \\ 0. & 0. & -1.528 & 0. & -1.277 & -0. \\ 0. & 0. & 0. & 0. & 0. & 1.485 \\ 0. & 0. & 0. & 0. & -0.267 & -0. \end{bmatrix}$$

We compute the eigenvalues of B using numpy.linalg.eigvals and receive $[0.0000000e{+}00{+}2.54832478j, 0.0000000e{+}00{-}2.54832478j, 6.9388939e{-}18{+}$

$0.62938425j, 6.9388939e{-}18{-}0.62938425j, 0.0000000e{+}00{+}1.76349547j, 0.0000000e{+}$
$00 - 1.76349547j]$
The output from my implemented method gives

$[0.{+}2.54832478j, 0.{+}1.76349547j, 0.{+}0.62938425j, -0.{-}2.54832478j, -0.{-}1.76349547j, -0.{-}0.62938425j]$

2

2.1

First we show symmetry preservation.
Let $QR = T_k$ be the QR factorisation of a symmetric tridiagonal matrix T_k. Then at the next step of QR we have $T_{k+1} = RQ = Q^TQRQ = Q^TT_kQ \Rightarrow T_{k+1}^T = (Q^TQRQ)^T = (Q^TT_kQ)^T = Q^TT_k^TQ^{T^T} = Q^TT_kQ$ by symmetry of T_k. Since $T_{k+1} = T_{k+1}^T$ for any k we have shown symmetry preservation for the whole QR proccess.
Now we show tridiagonality preservation.
If we have $T_0 = Q_0R_0 \Rightarrow Q_0 = T_0R_0^{-1}$ and then $T_1 = R_0Q_0 = R_0T_0R_0^{-1}$. We know that the inverse of an upper triangular matrix is upper triangular. From the last coursework we know that the lower bandwidth of T_1 will be the sum of the lower bandwidths of R_0, T_0, R_0^{-1} which is $0 + 1 + 0 = 1$. We repeat this idea with $T_1^T = {R_0^{-1}}^TT_0R_0^T$ which is the product of matrices of upper bandwidth $0, 1, 0$ meaning T_1^T has an upper bandwidth of 1. Since we know T_1 is symmetric we know that T_1 must have upper and lower bandwidth of 1 making T_1 tridiagonal. This procedure happens at each step and so this argument can be repeated for any step of the QR process meaning that tridiagonality is always preserved.

2.2

We construct A

$$\begin{bmatrix} 0.33333333 & 0.25 & 0.2 & 0.16666667 & 0.14285714 \\ 0.25 & 0.2 & 0.16666667 & 0.14285714 & 0.125 \\ 0.2 & 0.16666667 & 0.14285714 & 0.125 & 0.11111111 \\ 0.16666667 & 0.14285714 & 0.125 & 0.11111111 & 0.1 \\ 0.14285714 & 0.125 & 0.11111111 & 0.1 & 0.09090909 \end{bmatrix}$$

and receive this output from our modified pure QR

$$\begin{bmatrix} 8.33789775e-01 & -1.22059895e-04 & -1.34807544e-16 & 4.90840102e-17 & -2.44196451e-19 \\ -1.22059895e-04 & 4.30979026e-02 & 6.59994325e-07 & 1.17396517e-17 & -4.89211789e-18 \\ 2.46311560e-25 & 6.59994325e-07 & 1.29982426e-03 & 2.39086846e-09 & 1.60456091e-17 \\ 9.87547041e-31 & 4.37104633e-27 & 2.39086844e-09 & 2.29961954e-05 & 2.07196071e-12 \\ 0.00000000e+00 & -1.84096559e-34 & 3.21011920e-30 & 2.07196864e-12 & 1.79889252e-07 \end{bmatrix}$$

Using numpy.linalg.eigvals we obtain the eigenvalues for A

$[8.33789794e{-}01, 4.30978838e{-}02, 1.29982425e{-}03, 2.29961954e{-}05, 1.79889252e{-}07]$

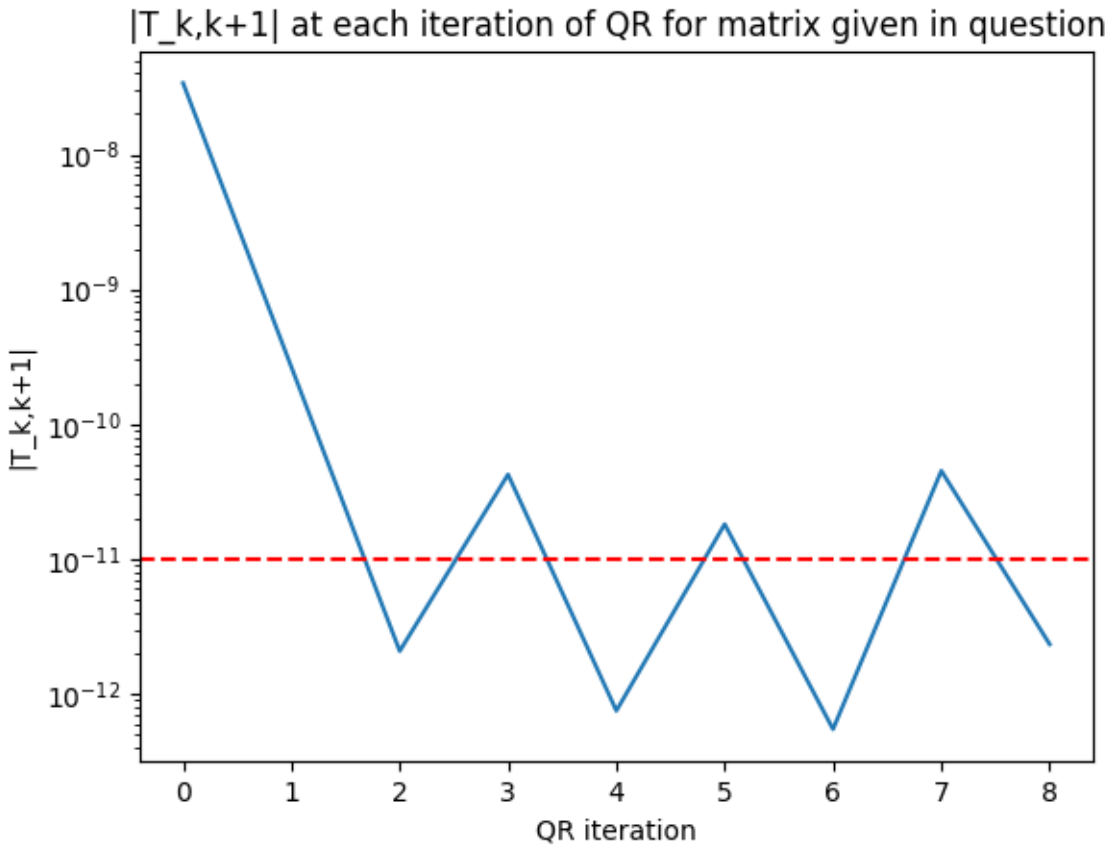
We see the eigenvalues are displayed on the diagonal of C. We calculate the percentage difference between those on the diagonal and those calculated via numpy.linalg to be

$[2.25986315e{-}06\%, 4.36960808e{-}05\%, 8.01403136e{-}07\%, 1.94494881e{-}08\%, 3.71246493e{-}10\%]$

Further, for each eigenvalue we check

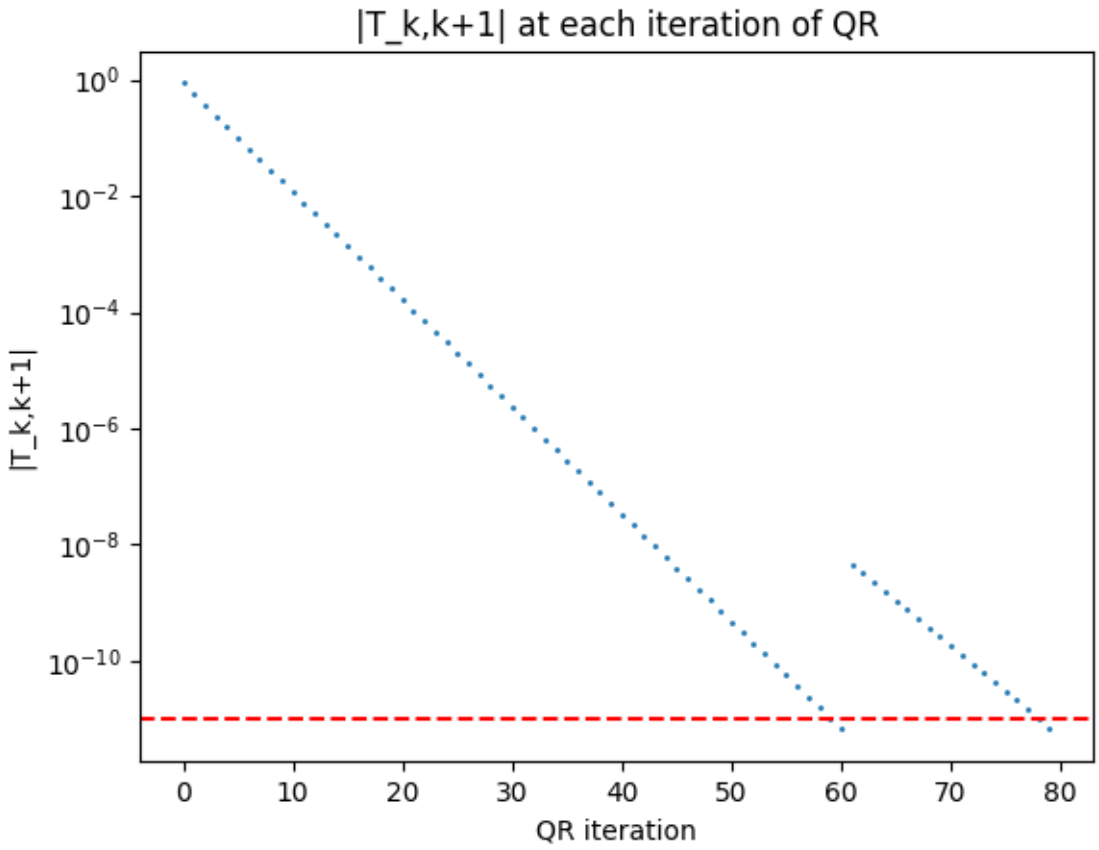
2.3

We will now plot the concatenated arrays of $|T_{k,k-1}|$ for each QR step for some matrices of my choice.
NOTE: since the values in these arrays are going to be incredibly small we use a log scale on the y axis to clearer show the sawtooth pattern.
NOTE: Upon rereading I have noticed in the titles and on the y axis of my graphs I have a typo: should be $k-1$ and not $k+1$.
We start with the matrix, which I name A_0, given in the question.



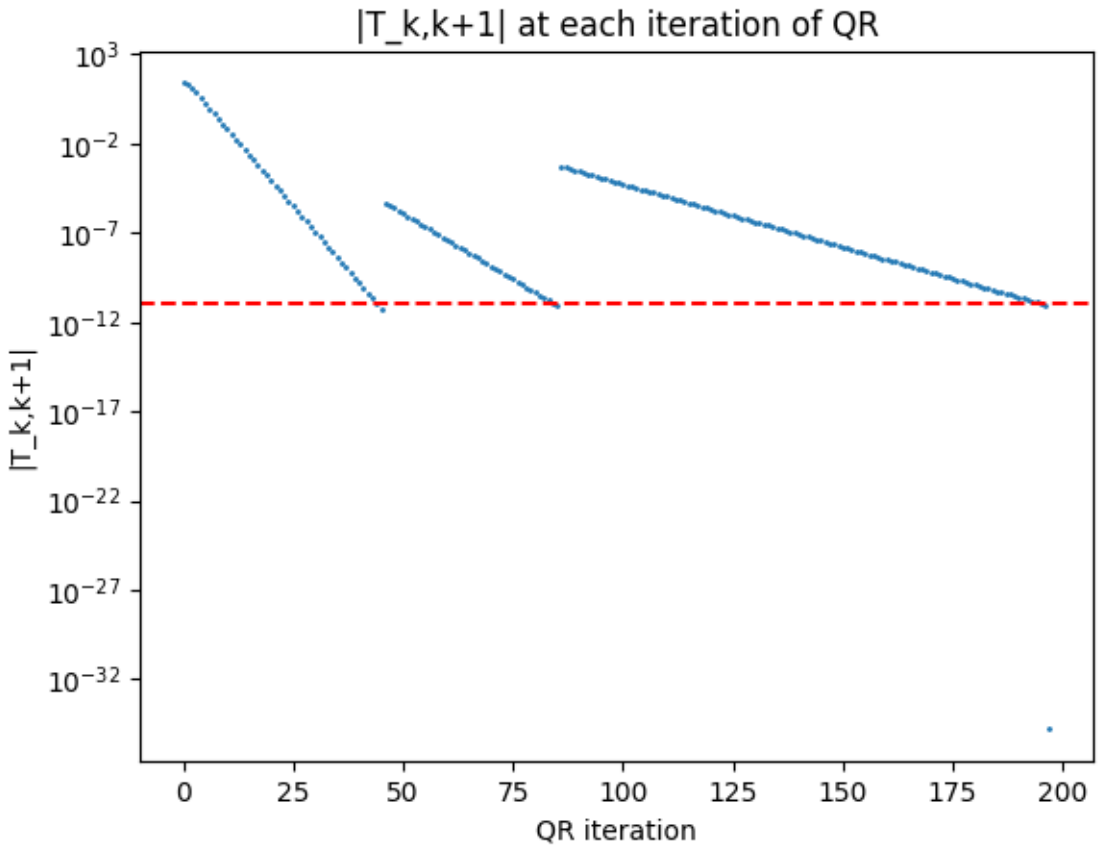
We generate a random symmetric matrix

$$A_1 = \begin{bmatrix} 17. & -9.5 & -15.5 \\ -9.5 & 5. & -10. \\ -15.5 & -10. & -2. \end{bmatrix}$$



We generate another random symmetric matrix but this time of a different size, and one that showcases an interesting exception.

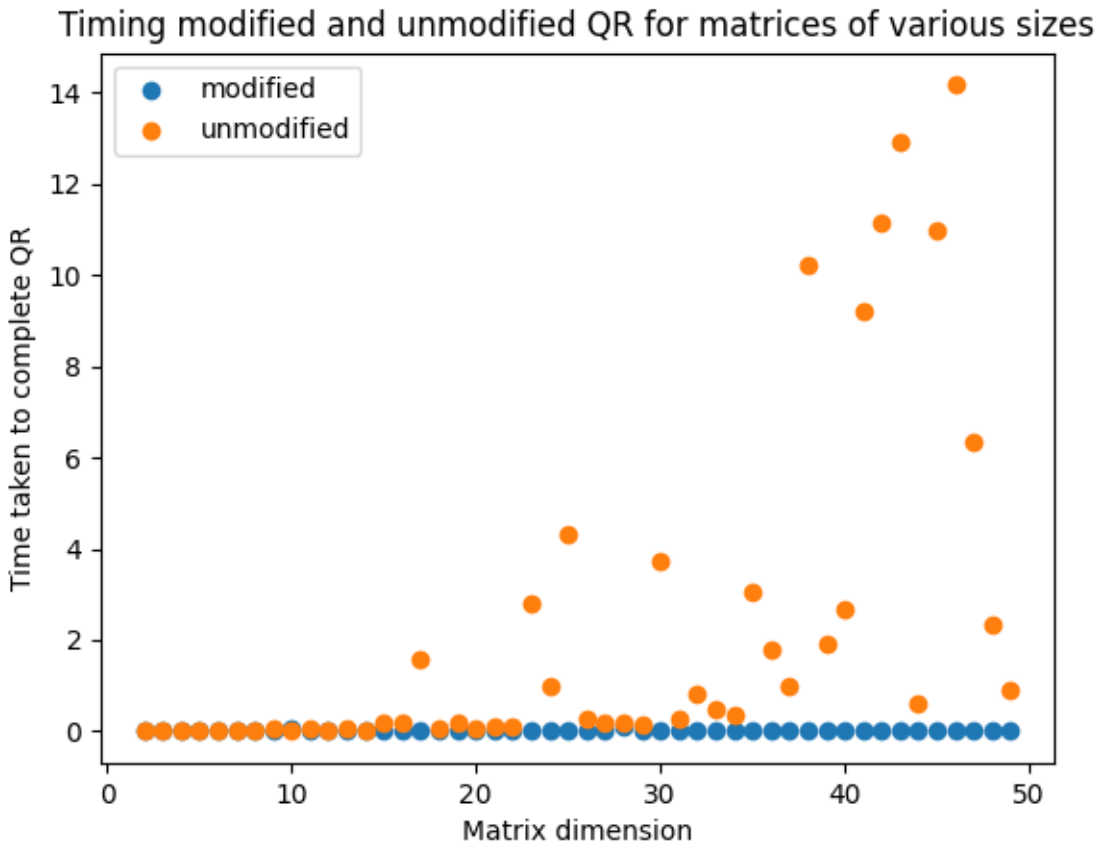
$$A_2 = \begin{bmatrix} 1. & -21. & -38.5 & -59.5 & 45. \\ -21. & 86. & 17.5 & -30. & 14.5 \\ -38.5 & 17.5 & -86. & 40. & 86.5 \\ -59.5 & -30. & 40. & -51. & 47.5 \\ 45. & 14.5 & 86.5 & 47.5 & -37. \end{bmatrix}$$



In all of these plots we see a clear sawtooth pattern. We see that for T_0 that at every QR the entry on the last row and second to last column decreases exponentially (since log scale). Once we reach the termination criterion which is when that entry is less than 10^{-12} as shown by the red line we move on to the sub-block of T_0 which I name T_1. The entry on the last row and second to last column pops back up dramatically instantly and then resumes the same patter of exponential decrease during the QR steps of the sub-matrix. So the length of each tooth shows how many QR iterations of that sub-block of T is necessary to reduce the matrix in each deflation step. Usually the number of teeth is one less than the size of the matrix since we are reducing to a sub-block $m-1$ times. The exception is highlighted by the last matrix which shows that when reduced to the last sub-block that matrix already meets the termination criterion and so we see no tooth.

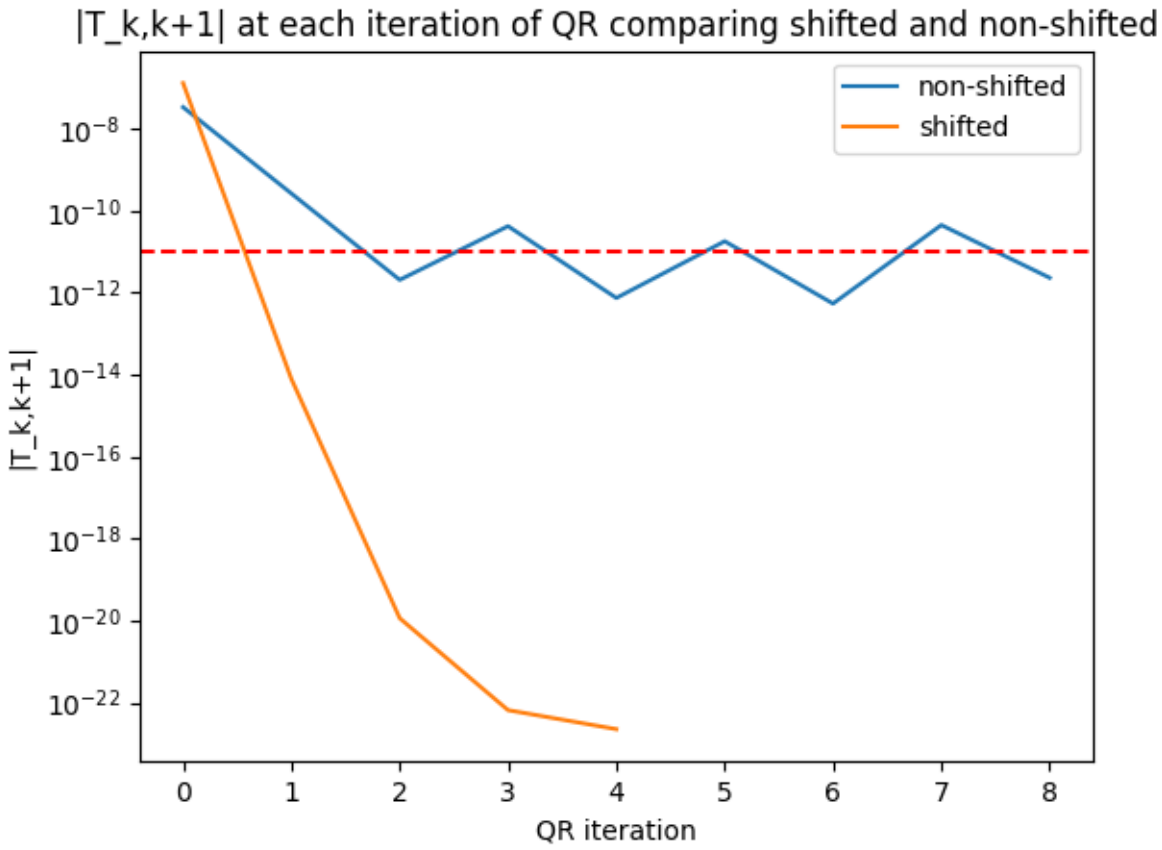
We first time the modified and unmodified QR on A and we obtain times 0.0005588440000003914 and 0.0008131599999998684, so the modified QR is significantly faster. We generate a random symmetric matrix of dimensions

$2\times2\ldots50\times50$ and time the modified and unmodified QR on each one. Plotting the times we clearly observe the modification has decreased the QR processing time.

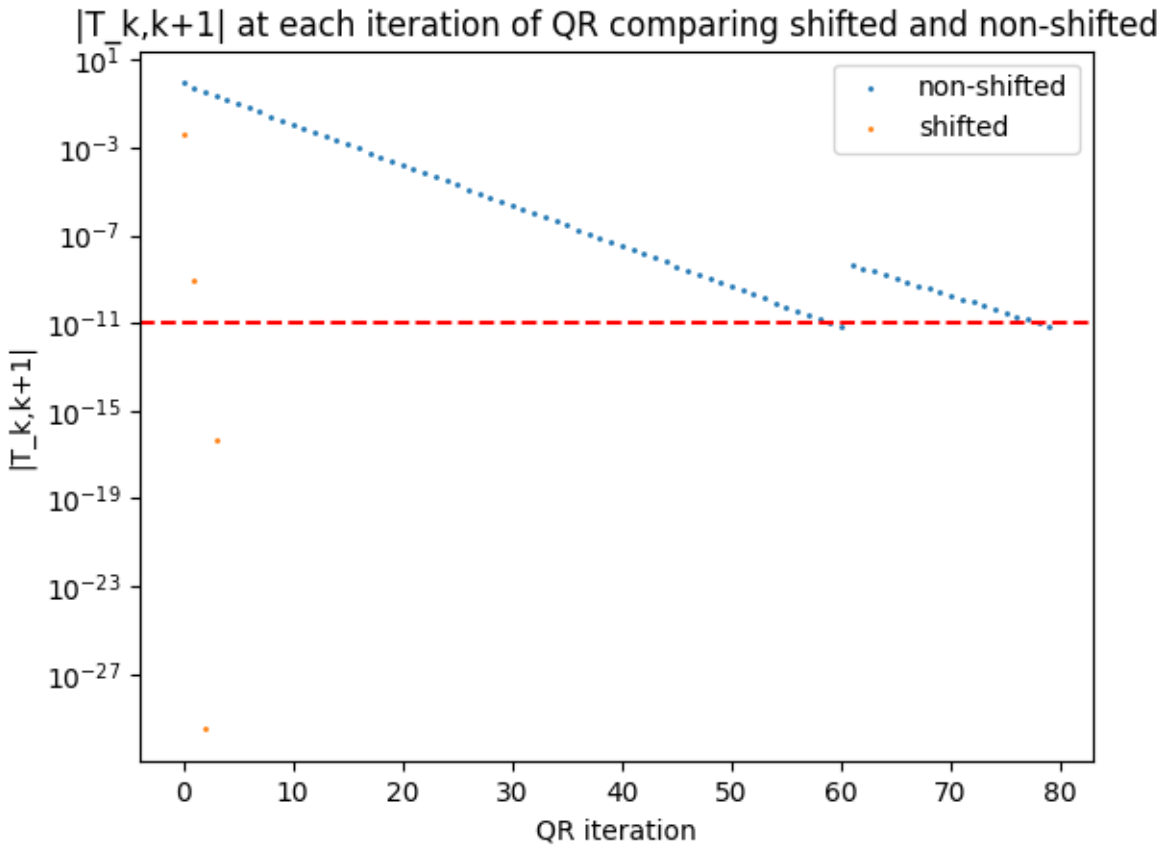


2.4

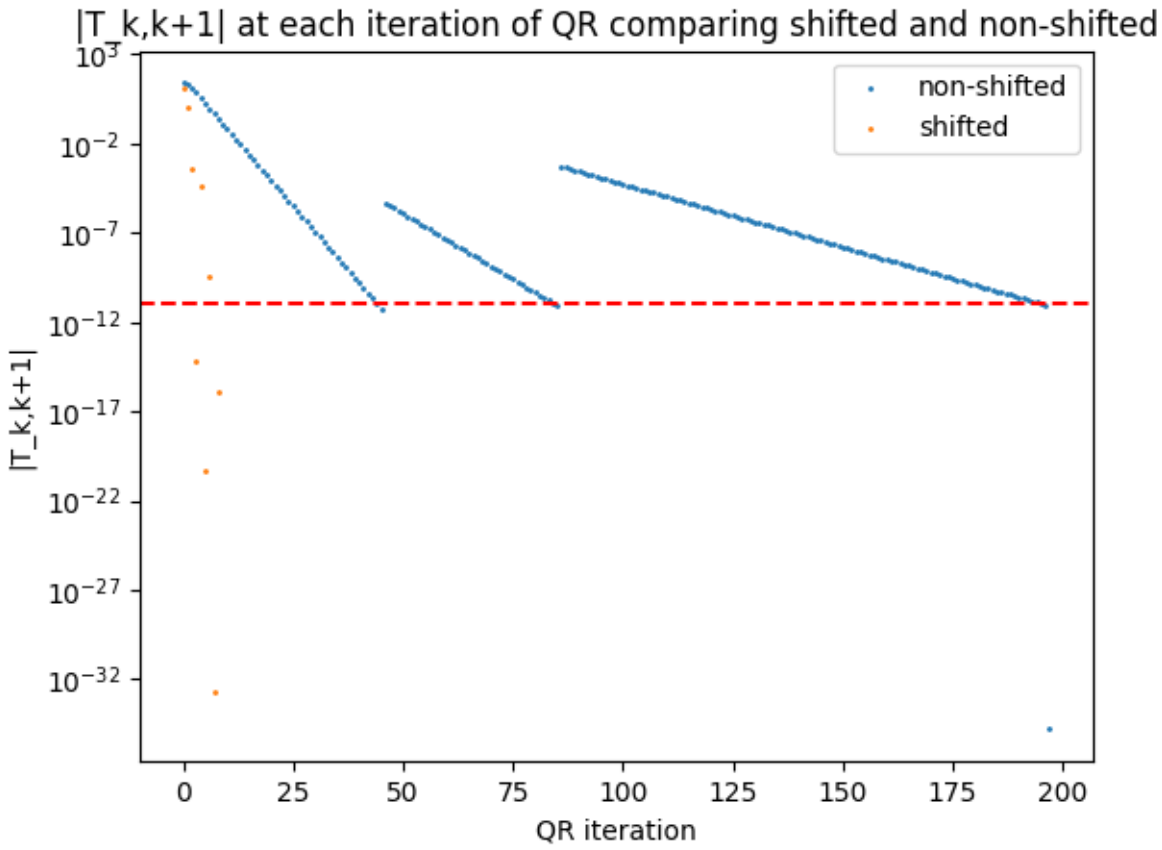
For the matrix from the question we see that we no longer have the sawtooth pattern for the shifted QR since every sub-block, bar the first, the tolerance criterion is fulfilled straight away. This is due to the wilkinson quotient iteration being applied at each step, meaning convergence is achieved much quicker similar to the rayleigh quotient iteration as we know from the notes (cubic convergence rather than linear). The reason the wilkinson quotient speeds up convergence is because when we take μ to be the eigenvalue estimate we are using the non-shifted eigenvalue estimate in the bottom left corner of T_k when solving for the eigenvalue in the bottom left 2×2 block hence getting a better eigenvalue estimate.



We see that for A_1 we only need 4 QR decompositions to completely converge.



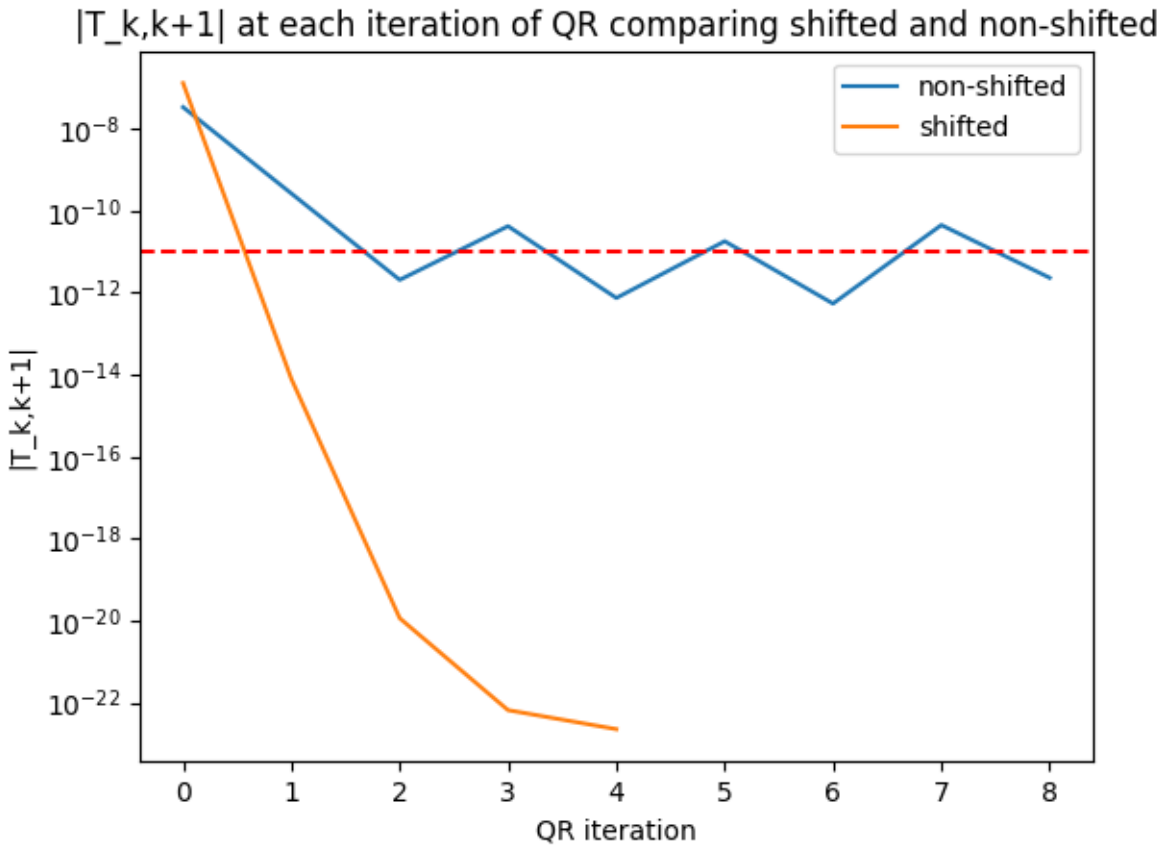
For A_2 we again see a rapid decrease in the number of QR decompositions needed: convergence is achieved for the shifted QR method in half the time that the non-shifted QR reaches the first sub-block stage!



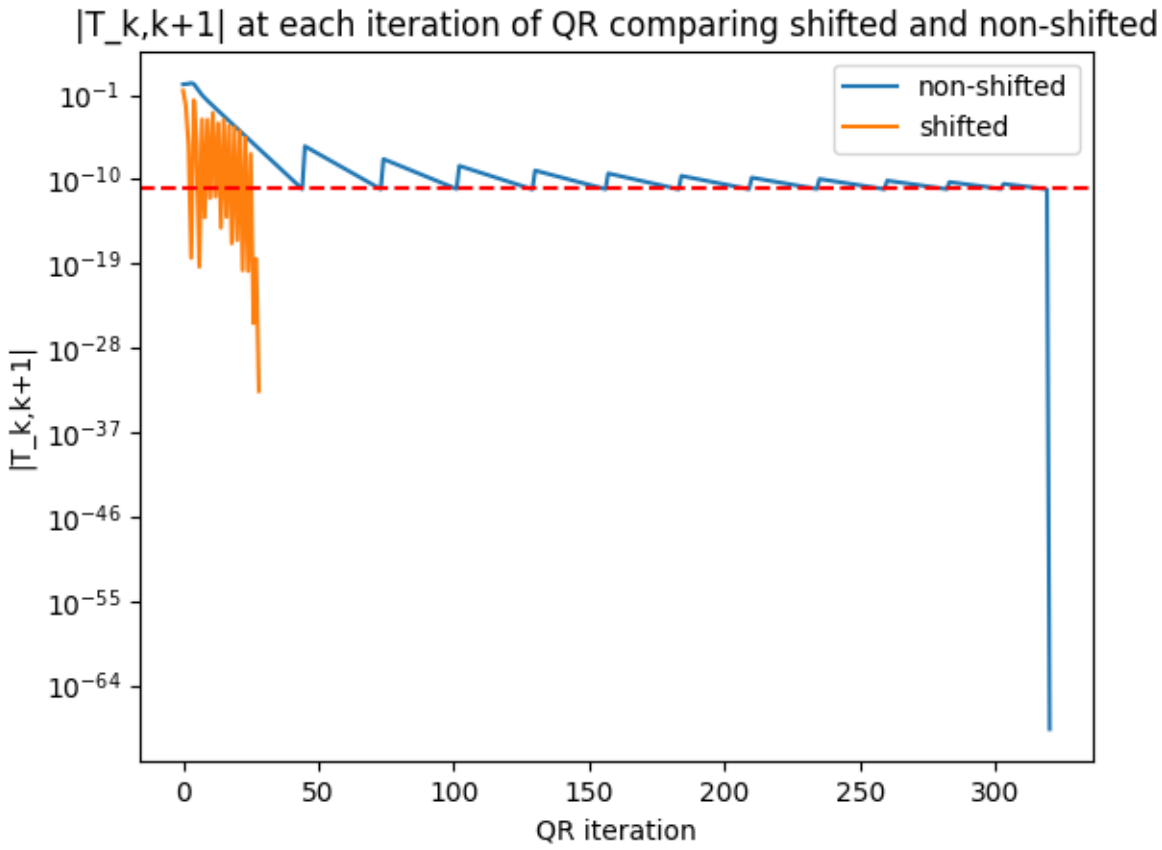
We zoom in on the shifted QR to see that it still follows a sawtooth pattern just much more sharper and less wide than the non-shifted QR.

2.5

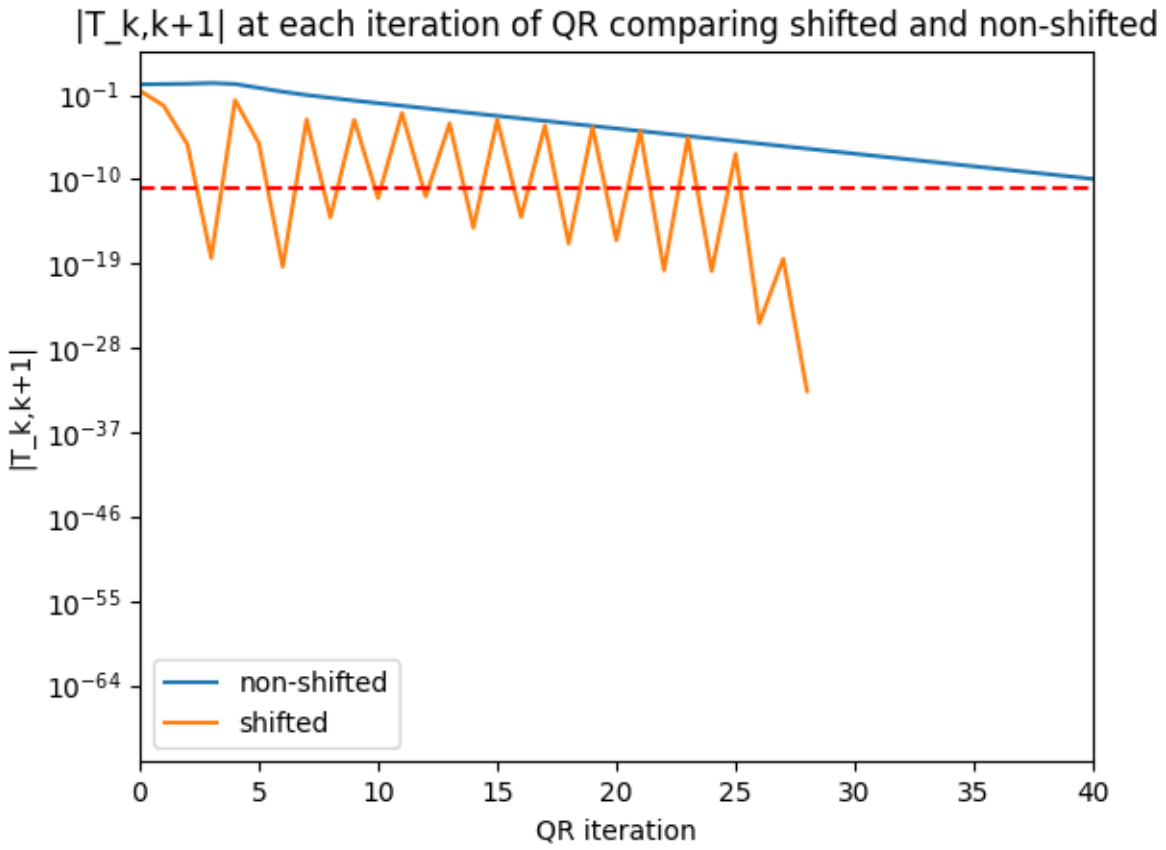
We compare the some plots for A_b given in part b and this new $A_e = D + O$
Plot for A_b


|T_k,k+1| at each iteration of QR comparing shifted and non-shifted
non-shifted
shifted
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
10^{-18}
10^{-20}
10^{-22}
|T_k,k+1|
0
1
2
3
4
5
6
7
8
QR iteration

Plot for A_e



We zoom in on the shifted plot for A_e so we can see what is going on.



Comparing the non-shifted plots we observe that A_e takes many more QR iterations before it converges than A_b. Even though we would usually expect this since A_e is of 3 times larger dimension, and so will of course have more teeth on it's plot. However, when comparing the sizes of each tooth for non-shifted we see a large difference. The number of iterations for each sub-block a.k.a the width of each tooth on A_e is much wider than those on A_b. For example, the first tooth on A_b is of length 3 which is only below it's matrix size but the first tooth on A_e is 28 iterations long, which is 13 more than it's dimension size. So each sub-block in A_e takes longer to converge to the tolerance criterion. The main similarities are that the sawtooth patterns are still very prominent and that the edges of the teeth for both non-shifted plots are very straight. When comparing the shifted plots, we see in both a massive reduction in the number of iterations needed. However, A_e is a much larger reduction percentage (over 90) where as A_b is only a 50 percentage reduction. Another key difference is that for A_b we lose the sawtooth pattern since every deflation step the termination criterion is met after one QR step but for A_e we see a clear sawtooth pattern,

requiring a few QR iterations per each deflation block. We also observe that the A_b shifted plot is monotonic decreasing unlike the one for A_e which makes A_b quite unique since not many matrices will have a plot like that. The last difference we observe is that for A_e the shifted graph is bounded above by the non-shifted graph but for A_b this is not the case as at the first QR iteration the non-shifted value is actually less than the shifted value recorded.