Final Project

**Introduction**

For this project, I will be exploring the topic of finding eigenvalues by using the QR algorithm. There are many interesting real world applications of the QR algorithm, including Google’s page rank algorithm to traverse between web pages which features a stochastic matrix where each web page is assigned to a row and column, and also in various fields of data science, like in principal component analysis, in which factorization can be utilized to analyze the vibration levels in industrial machines. In the fundamental sense, in order to calculate the eigenvalue of a matrix, one has to compute the characteristic equation, which requires you to calculate the determinant. This can be a relatively expensive computation. Through Gaussian Elimination, you will find the determinant by performing a sequence of row operations. Row reducing an n × n matrix of integers alone already takes O(n3). Another issue associated with calculating eigenvalues is that solving the characteristic equation involves computing the roots of a polynomial, which can be a difficult problem. Finding an efficient root finding algorithm becomes part of the problem then. The technique of Newton’s method can be used, but the challenge comes from having to choose a set of proper initial points, which could be complex, in order to influence the algorithm to converge to all the roots.

With all of this in mind, how are we to remedy these setbacks for finding the eigenvalues of a matrix? This is where the QR algorithm comes in.

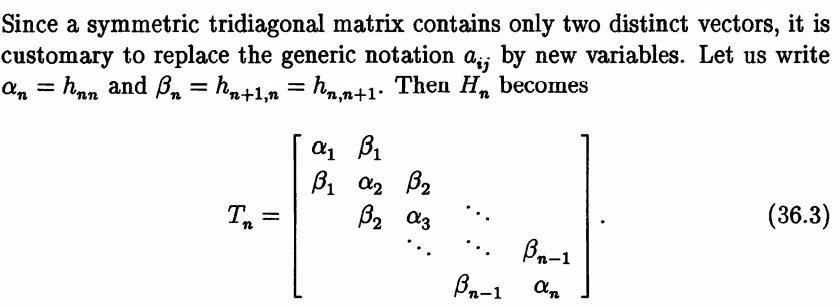
The QR algorithm negates the issue of having to calculate the determinant of the matrix by implementing a Schur decomposition of the original matrix instead. You are also not required to solve for the roots of a polynomial. The process of factoring the original matrix and reverse multiplying the components will in turn convert the original matrix into one where the diagonal entries are the eigenvalues. The reason why the QR algorithm is desired is that if you reduce the original matrix into, say, a Hessenberg matrix, you can achieve a time complexity of O(n2) operations per iteration as opposed to O(n3) from before.

**Theory**  
 By definition, an eigenvector of an n × n matrix A is a *nonzero* vector x in Rn where some scalar value **λ** exists such that Ax = **λ**x. The corresponding eigenvalue of A is the scalar multiple **λ**. We then have the eigenpair, denoted as (**λ**, x), which is an ordered pair of an eigenvalue and its corresponding eigenvector. In essence, eigenvalues are the *roots* of polynomials of degree n, so there are n eigenvalues for an n × n matrix (though some roots may be repeated). Thus, the eigenvectors are the diagonals of the reduced matrix. Some issues that we might find when calculating eigenvectors numerically would be the fact that there are infinitely many eigenvalues for the system. Note that if the matrix is above 4 × 4 dimensions, then the problem of finding eigenvalues and their eigenvectors becomes nearly impossible.

A special case of the finding eigenvalues and eigenvectors problem is when the matrix A is called a symmetric tridiagonal matrix. Let us define what that is.

**Definition:**

Let A be our n × n matrix. A is a symmetric tridiagonal matrix if it is in the form

,

where An,n = αn and An+1,n = An,n+1 = βn and the rest of the entries are zeroes.

**Definition:**

Let A and B be a matrix, then A and B are similar if there exists some kind of transformation from A to B such that

A = Q-1BQ, (1)

where Q is an invertible matrix.

This specific transformation is called a similarity transformation. Thus, if A and B are similar, then they will have the same eigenvalues.

**Definition:**

A square n × n matrix A with real numbers or elements is an orthogonal matrix if its transpose AT is equal to its inverse matrix I,

i.e. AT = A-1.

Or we can say when the product of a square matrix and its transpose gives an identity matrix, then the square matrix is known as an orthogonal matrix,

i.e. AAT = ATA = I.

For the QR method, we will be decomposing our matrix A into two separate matrices, Q and R, where Q is an *orthogonal* matrix and R is an *upper triangular* matrix. So,

A = QR.

Essentially, the QR method will decompose our matrix A further by a number of iterations.

Let Ak be our matrix A during the kth iteration of our QR method. Then A can be decomposed into

Ak = QkRk.

Now, we are going to have to calculate the matrix for the next iteration Ak+1. Since we want to preserve the eigenvalues of the original matrix A throughout our iterations, we can utilize a nice property of orthogonal matrices, which says that

Q-1 = QT.

So, we can use the first definition (1) and substitute to decompose our matrix A into

Ak+1 = RkQk = Qk-1QkRkQk = Qk-1AkQk

By using a similarity transformation, matrix Ak and Ak+1 will have the same eigenvalues. Eventually, by using the QR method, you will converge to the n × n matrix Ak where

Ak = RkQk = [ **λ** X …. X

0 **λ** …. X

0 0 ….

0 0 …. **λ**n ]

All in all, QR decomposition is an expensive method to find the eigenvalues of a matrix, which runs for O(n3). The off diagonal entries will converge to 0 at a rate of O(|**λ**j/**λ**j-1|).

The most integral aspect about the QR algorithm is the QR factorization, which will be done using *rotational matrices*.

**Definition:**

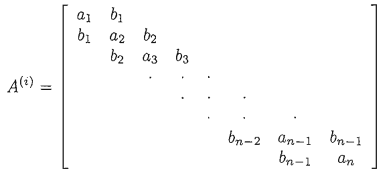
Let P(i, j), where i < j, denote an orthogonal matrix that is structured identically to the identity matrix except the elements where

pi,i = pj,j = cos θ and pi,j = -pj,i = sin θ,

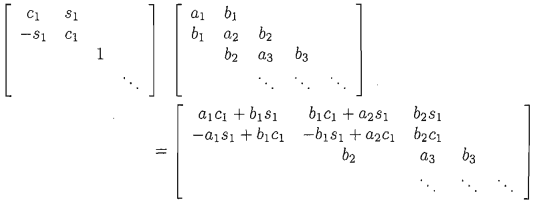
for some angle θ.

P(i, j) is a rotational matrix.

Thus, our factorization of matrix A will be the premultiplication of A by the rotational matrix P(i,j) where A is of the form



and the equation P(i,j)A = factorized(A) is of the form

,

where cj and sj are the shorthand notation for cosine and sine respectively for the rotational matrix P(j, j+1).

The goal is to make the element under each diagonal element equal to 0 in the factored form of A. In each iteration, we can sub in these values of cj and sj.

For j = 2, 3, 4,…., n - 1, we have

,

where t is the old value of element bj.

By doing this process, knowing these values will help us derive the next iteration of Ai (i.e. Ai+1) in the QR method without directly computing the subsequent Ri and Qi matrices.

Remember that R in the QR factorization is an upper triangular matrix. This triangular matrix R(i) of the matrix A(i) is

R(i) = P(n-1,n)**•••**P(3,4)P(2,3)P(1,2)A(i) (2)

If we have that

Q(i)TA(i) = R(i), (3)

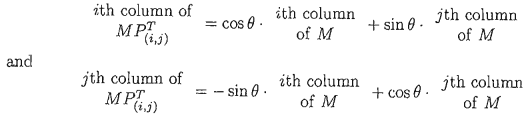
then if we substitute (2) into (3),

Q(i)TA(i) = P(n-1,n)**•••**P(3,4)P(2,3)P(1,2)A(i)

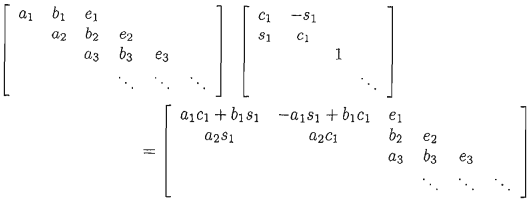
Q(i)T = P(n-1,n)**•••**P(3,4)P(2,3)P(1,2)

Q(i) = PT(1,2)PT(2,3)PT(3,4)**•••**PT(n-1,n).

It is not necessary to directly compute the matrix Q(i), but it is sufficient to keep track of the sj and cj entries for each of the rotation matrices P(j,j+1) based on the relations given by

.

This leads to the matrix multiplication of RQ, which takes the form

.

Lastly, we will set aj, bj, and aj+1 as follows:

aj = ajcj + bjsj,

bj = aj+1sj,

aj+1 = aj+1cj

for j = 1, 2, 3, n - 1. The ej term represents the values that we did not save from R(i) during the factorization process. As we can see, it is actually not used in our computations.

For our QR algorithm, the convergence rate of the subdiagonal entries to zero without using a shift are

|λp+1 / λp|,

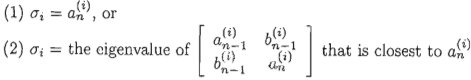
where λp is the pth largest eigenvalue of the matrix. This can be an issue when eigenvalues have magnitudes that are close to one another.

However, if we were to apply a shift on the original matrix, then we can effectively accelerate the convergence. After integrating a shift σ into the QR algorithm, the convergence rate becomes

|λp+1 - σ| / |λp - σ|.

This has it so that if σ is close to an eigenvalue, then convergence of the subdiagonal entry is much more rapid.

We may choose the shift however we wish. Though, the two most commonly used shifts for σi are

,

where i is the iteration in the algorithm, denotes the nth diagonal element of n × n matrix A, and is the n-1th subdiagonal element.

The reason why we apply this shift is to make the subdiagonal element converge to zero the fastest. We will be checking this specific element’s value each time at the end of every iteration of the algorithm. If its value is lower than some specified *convergence tolerance*, then the next diagonal element is determined to be an eigenvalue of the original matrix A(0). Furthermore, we can calculate the value of that eigenvalue as

+ Σ,

where Σ is the total number of shifts accumulated from the first iteration to the current one. This process is repeated until , in which + Σ and + Σ are the last two eigenvalues of A(0).

All in all, the QR algorithm is one of the most important algorithms for eigenvalue computation. However, one limitation of the QR algorithm is that it can only be applied to dense or full matrices only, which is a matrix where most elements are nonzero. Another limitation of the algorithm is that there is a transformation step before the algorithm itself, to transform the matrix into the correct form, if the matrix we are evaluating is not already in either one of two forms: the Hessenberg form, which is a variant of triangular matrices, and the Hermitian or symmetric form, which is called a tridiagonal matrix.

**Homework Problems**

1. Let our symmetric tridiagonal matrix A be

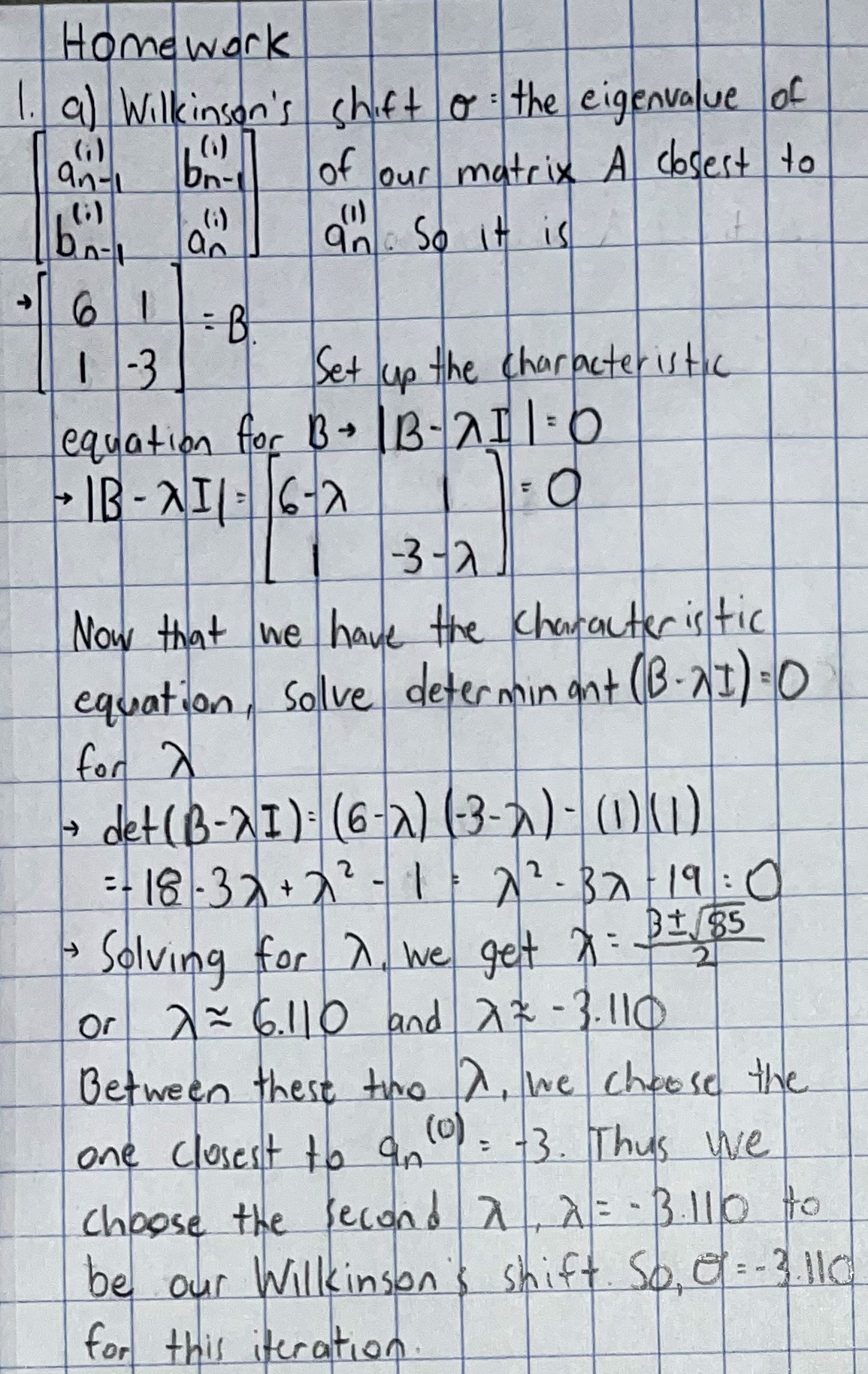
|  | 2 -1 0 | -1 6 1 | 0 1 -3 |  |
| --- | --- | --- | --- | --- |

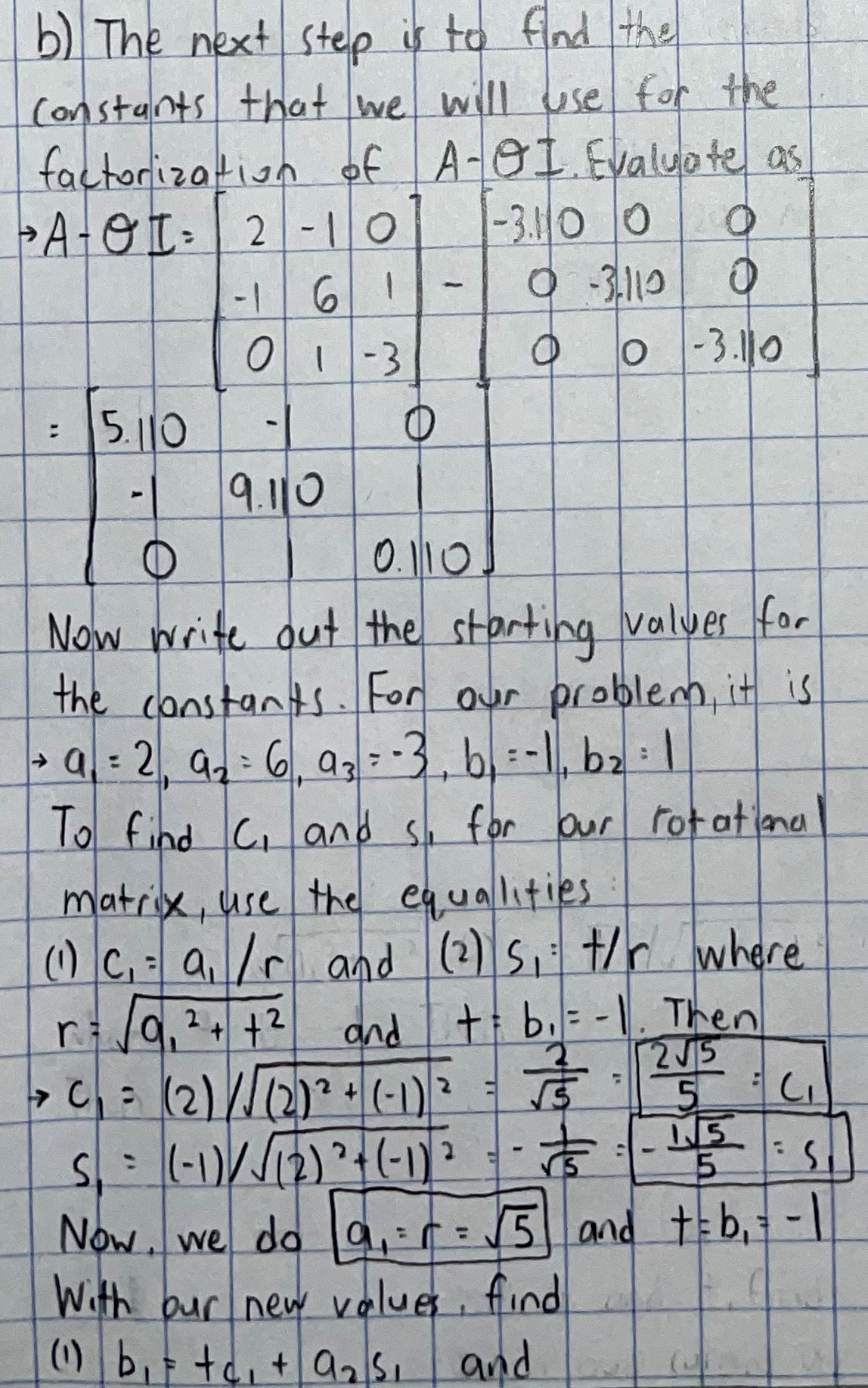
Perform one iteration of the QR algorithm on A to find A(1) using Wilkinson’s shift. Let the convergence tolerance TOL be 5 \* 10^-14 and our shift σ to be Wilkinson’s shift. In order to do so

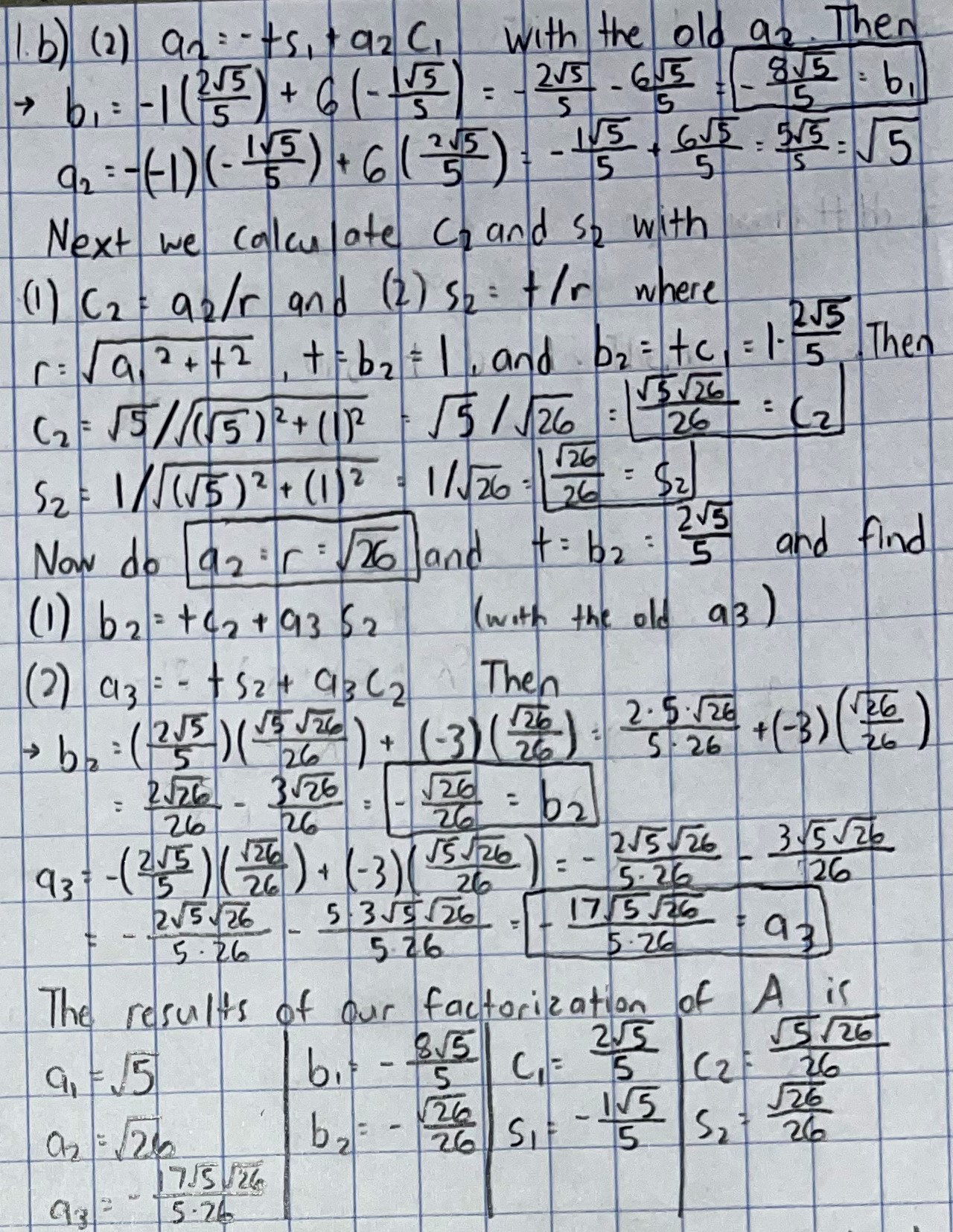
* 1. Calculate the Wilkinson shift σ.
  2. Find the constants of the factorization of A - σI: a1, a2, a3, b1, b2, c1, c2, s1, s2, where I is the identity matrix and ai, bj, corresponds to the entries in matrix A, and cj corresponds to the entries in the rotational matrix R for i = 1, 2, 3 and j = 1, 2.
  3. Then use the results from the factorization of A to compute the product R(0)Q(0).

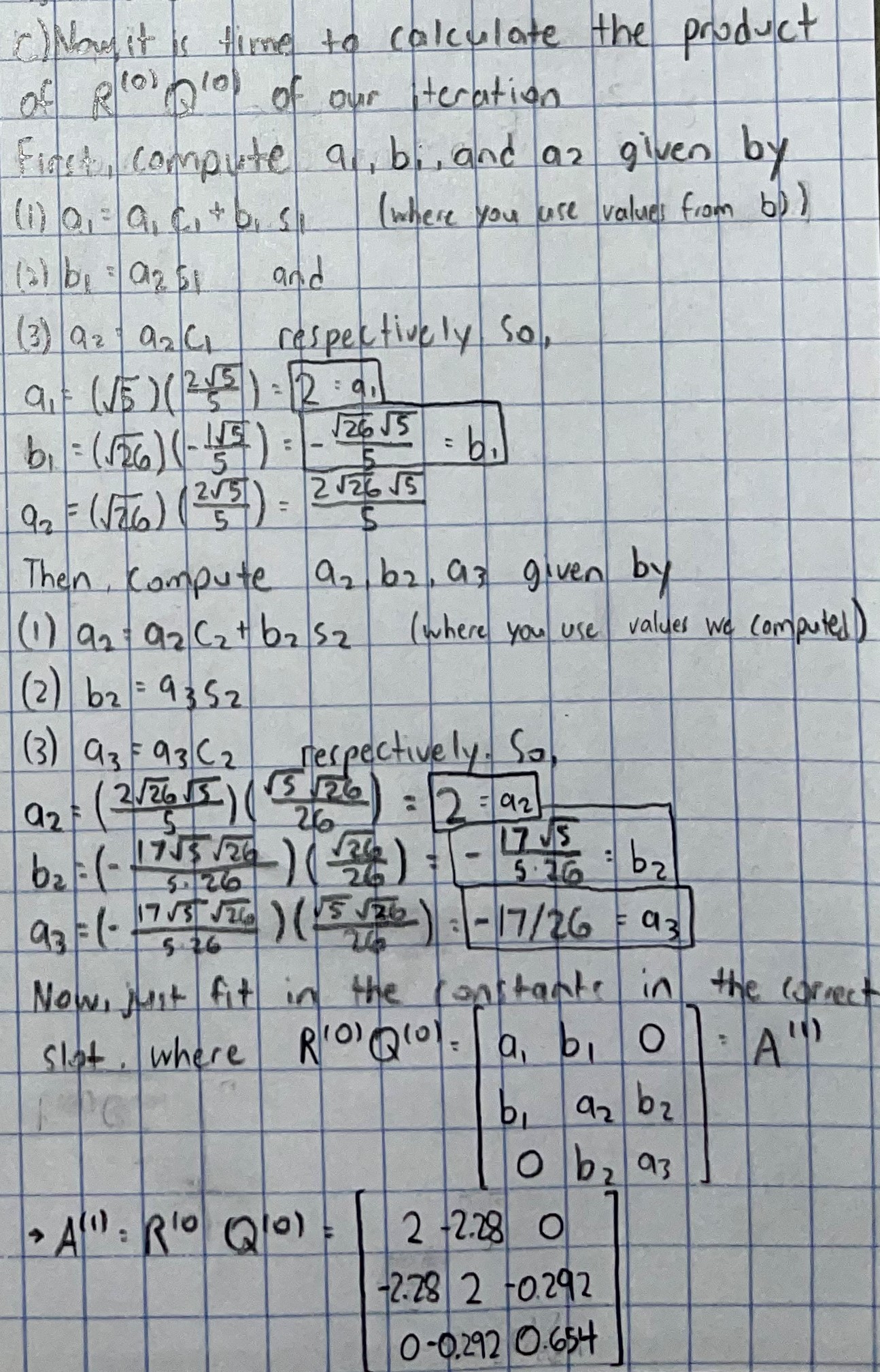
1. Create a program that implements the QR algorithm to find the eigenvalues of the matrix from problem 1, both without a shift and then with Wilkinson’s shift. Compare the number of iterations for each case. Let the convergence tolerance be 5 \* 10^-14.

**Solutions**

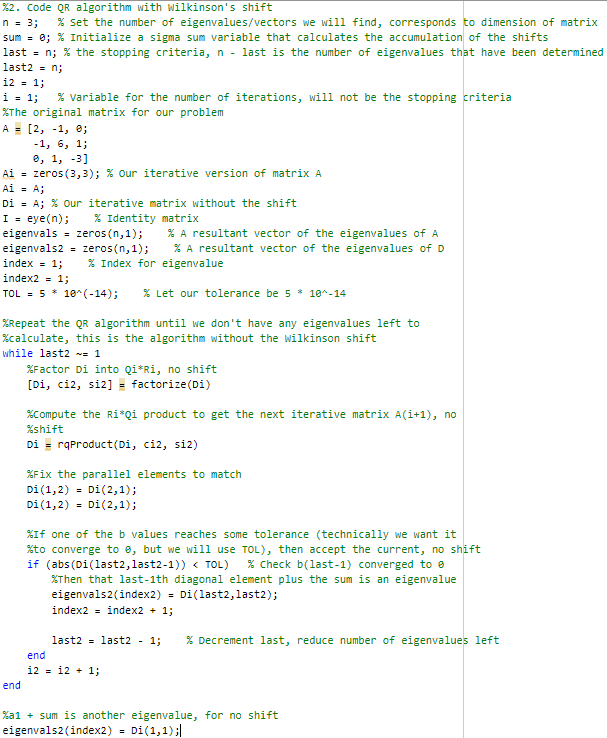


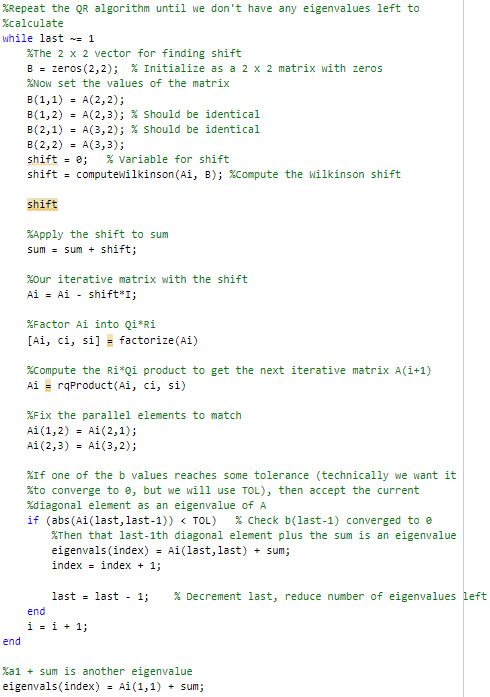




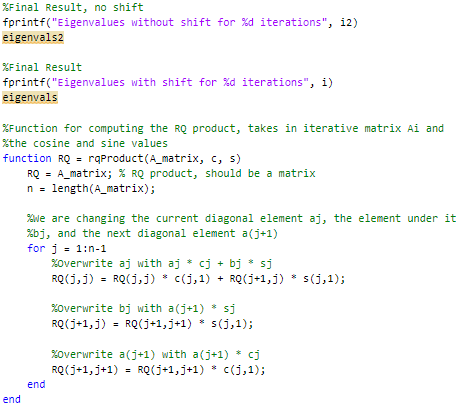


1. First, do the QR algorithm without shift

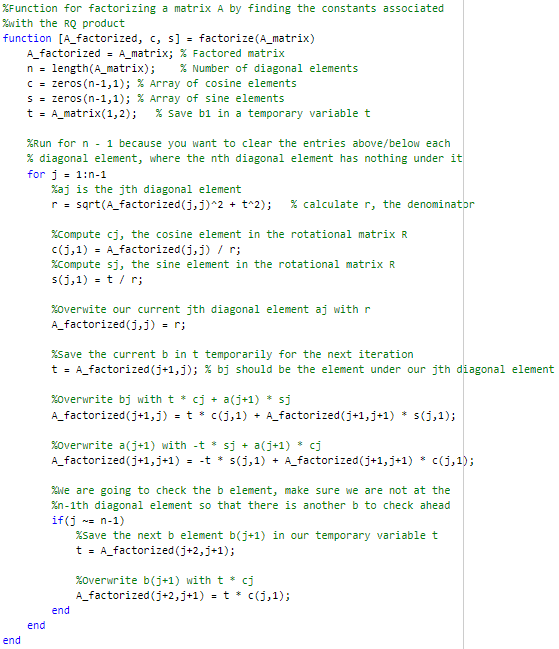
  
  
  
  
  
Now do the algorithm with Wilkinson’s shift



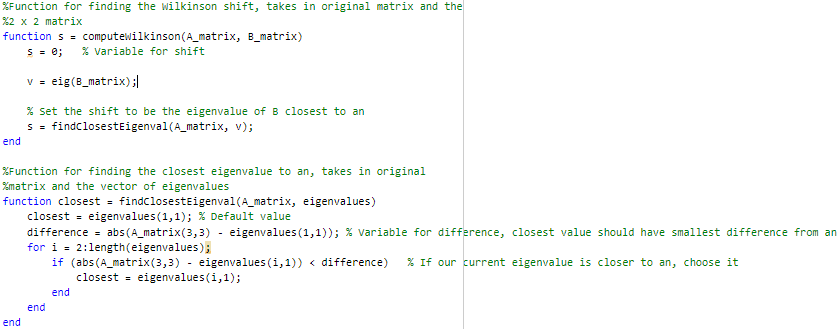
Print the eigenvalues along with the number of iterations for each version  
The function finds the product of RQ after the matrix A is factorized



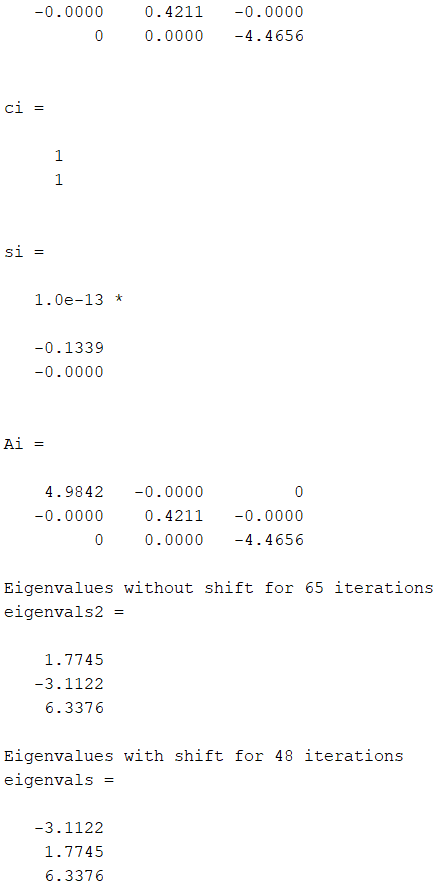
This function will factorize A into Q and R



The first function computes the Wilkinson shift, and the second is a helper function for determining the closest eigenvalue



Output



**Homework Narrative Reflection**

For the first problem, I wanted to give the student a good understanding of the QR algorithm, and what better way to do it than actually implementing the algorithm by hand yourself. This question is meant to help the student understand the application aspect more, rather than the theory. This problem will be giving them a hands-on implementation by asking them to perform an iteration of it. However, this algorithm can get complicated and tedious if done by hand, for there are a lot of terms that you have to keep track of. That is why I only asked for one iteration of the algorithm, and not to find all of the eigenvalues or eigenvectors of the given matrix. I decided to split this problem up into separate steps to better help the student and lead them along the algorithm. I chose to ask them to do a Wilkinson’s shift, because I wanted them to learn the more preferred variant of the QR algorithm. So, for this question, I think that the work is reasonably achievable.

Now, for the second problem, I wanted to have the student work on the same matrix to at least let them confirm parts of their answer to the first question. Then, we will add onto that by asking to code the rest of the QR algorithm. The coding portion was a little more involved than the hand written one. I still thought that it wouldn’t be an issue to also ask to implement the algorithm without the shift, since that would be a small change that could be easily implemented in code without too much added work. Plus, the student will be able to see the strength of having a shift in the first place, and hopefully the difference that it makes without having one. Since a coded program can easily execute an algorithm, this coding problem allows the student to examine the theory aspect of the QR algorithm.

I opted out from asking to find the eigenvectors of the matrix, because it is a standard process that is not necessarily unique to the QR algorithm. The second question was already getting on the lengthy side, so I didn’t ask them to code a program to find eigenvectors either.

**Conclusion**

Overall, I enjoyed learning about the QR algorithm. The QR algorithm provides some advantages, such as reducing the number of operations per iteration. Also, the QR algorithm provides more complete answers than other ones. For example, a limitation of another algorithm, the power method, is that it converges to the eigenvector of the *largest* eigenvalue of the matrix for almost all chosen initial vectors. This is not the case for the QR algorithm.

We can further improve the convergence speed by shifting our original matrix. In our case, we used Wilkinson’s shift. Without some kind of shift, the convergence for the QR algorithm can be painfully slow and may possibly require thousands of iterations.

**Bibliography**:

Bradie, Brian. *A Friendly Introduction to Numerical Analysis*. Pearson Prentice Hall, 2006.

Used for limitation →

https://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter4.pdf

For the image of tridiagonal symmetric matrix →

https://math.stackexchange.com/questions/3986722/since-a-symmetric-tridiagonal-matrix-contains-only-two-distinct-vectors

https://www2.math.upenn.edu/~deturck/m320/text/part8.pdf

Good source → https://pythonnumericalmethods.berkeley.edu/notebooks/chapter15.03-The-QR-Method.html#:~:text=The%20QR%20method%20is%20a%20way%20to%20decompose%20a%20matrix,%3DQTQ%3DI.

<https://lpsa.swarthmore.edu/MtrxVibe/EigMat/MatrixEigen.html>

<https://math.nyu.edu/~stadler/num1/material/num1_eigenvalues>

<https://www.quora.com/How-many-eigenvalues-does-an-n-x-n-matrix-have>

http://madrury.github.io/jekyll/update/statistics/2017/10/04/qr-algorithm.html

Help with deciding a topic:

<https://math.stackexchange.com/questions/1555357/numerical-methods-for-finding-eigenvalues-of-large-matrices>