Question 1
Reduction of a symmetric matrix

$$\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix}$$

to tridiagonal form.

Solution: The relevant subroutines for this problem are:

- diagHH
- outerProd

Found in LinAl.f90, the subroutine diagHH calls an outer product routine outerProd. In total diagHH takes the given matrix,

$$\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix}$$

and transforms it into tri diagonal form,

$$\begin{bmatrix} 5.00 & -4.24 & 3.14E-016 & 0.00 \\ -4.24 & 6.00 & 1.41 & -1.11E-016 \\ 3.14E-016 & 1.41 & 5.00 & -1.33E-015 \\ 0.00 & -1.11E-016 & -4.44E-016 & 2.00 \end{bmatrix}$$

This module is written in such a way that it should be able to accept a square matrix of any dimension and transform it into tri diagonal form. The process is remarkably similar to householder QR algorithm. Instead of creating an uppertriangular matrix however, we create a matrix in Hessenberg form. This is done by looping to the second to last row, instead of the last row. In addition, we have a step where we update A from the right, not just from the left. A is updated with the householder matrix H_j, and the cleverness of this algorithm I feel is in the following facts: Applying H_j to the left of A only affects rows $j+1$ and up, while applying it to the right only affects columns $j+1$ and up. This means the ones we've already worked on are left unchanged, and as we iterate through values of j we slowly clear out the necessary entries to arrive at tridiagonal form.

Question 2
Write a program of the QR algorithm (i) without shift and (ii) with shift, to calculate the eigenvalues of

$$A = \begin{bmatrix} 3 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Solution: The relevant subroutines for this problem are:

- eigenvalQR
- shiftQR
- householderQR

All found in LinAl.f90, the first two subroutines answer (i) and (ii) respectively, with both calling householderQR. Note that the given matrix is already in tridiagonal form, and as such the given methods used are applicable. If the matrix was not in tridiagonal form, the previous question's daigHH would be of use.

(i) First and foremost, eigenvalQR when called on A produces

$$\begin{bmatrix} 3.732 & 6.430E-11 & 2.427E-16 \\ 6.430E-11 & 2.000 & 8.479E-17 \\ 3.142E-59 & 8.510E-34 & 0.267 \end{bmatrix}$$

As expected, the eigenvalues are revealed along the main diagonal. The algorithm ceases when the norm of the subdiagonal entries is below a threshold. For the above matrix, the threshold was set to be $1E-10$. To be explicit, $v = [6.43E-011, 8.510E-034]^T$ and the algorithm terminated when $||v||_2 < 1E-10$. This was accomplished this in 30 iterations, tracked by adding a counter to my while loop. Within those 30 iterations, the following was performed: first compute the QR factorization of A. This was done with the householder routines of old. Then compute the reverse product, setting $A = RQ$ for the next pass.

This algorithm simultaneously reveals eigenvalues along the main diagonal, which is a little surprising to consider. As an added bit of neatness, the eigenvectors are easily recoverable as repeated applications of Q to the identity converges to eigenvector matrix, much how repeated applications of the above steps converge to the eigenvalue matrix D_λ. This eigenvalue matrix has all nondiagonal entries trend towards zero as the "revealing" along the main diagonal occurs. This is why a reasonable criteria for convergence is to check that the entries underneath the main diagonal are sufficiently close to zero.

(ii) Next, shiftQR when called on A produces

$$\begin{bmatrix} 3.731 & 2.490E-2 & -2.346E-16 \\ 2.490E-2 & 2.000 & 2.626E-16 \\ -9.891E-25 & -6.274E-22 & 0.268 \end{bmatrix}$$

which again reveals eigenvalues along the main diagonal. This algorithm seeks to speed up the convergence of the same algorithm without shifts. This is done with an intelligent choice of μ, and selecting $\mu = a_{mm}$ effectively uses the Rayleigh quotient shift. This algorithm performs a QR factorization of the shifted matrix A, corresponding to one step of shifted inverse simultaneous power iteration, then reconstructs A as $RQ + \mu I$ to cancel the shift.

The above matrix, which is remarkably similar to that reached in (i) took 5 iterations. However, iteration count is not a perfect metric in this particular comparison as the error to exit the while loop is computed differently here. In this subroutine, the very last subdiagonal entry (e.g., $-6.274E-22$) is checked against thresh ($1E-10$) to determine when to exit the loop. This is a reasonable criteria for convergence as, much like the previous part, all nondiagonal entries trend towards zero in this process.

Question 3
Consider the matrix

$$\begin{bmatrix} 2 & 1 & 3 & 4 \\ 1 & -3 & 1 & 5 \\ 4 & 1 & 6 & -2 \\ 4 & 5 & -2 & -1 \end{bmatrix}$$

Write a program of the inverse iteration to calculate the corresponding eigenvectors.

Solution: The relevant subroutines for this problem are:

- eigenvecIter
- LU_decomp
- backsubLU

Found in Linal.f90, the subroutine eigenvecIter incorrectly produces eigenvectors. More accurately, it creates vectors with no relevance to the situation at hand. It performs this tremendous feat by calling LU_decomp, producing an LU decomposition of the given matrix shifted by μ, a real number ideally close to an eigenvalue. The closer μ is to an eigenvalue, the quicker the algorithm should converge to the eigenvector associated with said eigenvalue. With the LU decomposition made, it is time to repeatedly back substitute using backsubLU. This is done to compute an inverse mapping without actually computing an inverse.

The error in my code lays within the backsubLU, despite it working on previous assignments. I'm not sure why it wouldn't hold up for me here, and lack the time to debug/ write it anew unfortunately. I'm fairly confident the entries I'm calling it on are correct, and the issue lays deep in the way it handles this particular problem. I couldn't compute the residual vector to exit the loop as my vectors never got close enough for this to be relevant, but if I could have I would have ensured the norm of the residual vector is less than $1E-10$ before stopping the algorithm.