1) Given matrix

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

Without the calculation of eigenvalues, please answer the following questions.

a) What is the sum of the eigenvalues of A?

The sum of the eigenvalues of a matrix is equal to the trace of A:

$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = \mathrm{Tr}(A) = 0 + 1 + 1 + -1 = 1. \tag{2}$$

b) What is the product of the eigenvalues of A?

The product of eigenvalues is just the determinant of A, which was determined using wolfram:

$$\lambda_1\lambda_2\lambda_3\lambda_4 = \det(A) = -2. \tag{3}$$

c) What's your observation of the eigenvalues of AA^{T}?

The eigenvalues of AA^{T} multiply to be the square of the product of eigenvalues of A: $\det\big(AA^{\mathrm{T}}\big) = \det^2(A)$

2) For $n \times n$ real symmetric matrices A and B, prove AB and BA always have the same eigenvalues.

Let λ be an eigenvalue of the matrix AB. Then, by definition, λ solves the equation $\det(AB - \lambda\mathbb{1}) = 0$. Note that $\det(AB - \lambda\mathbb{1}) = \det\big[(AB - \lambda\mathbb{1})^{\mathrm{T}}\big] = \det\big((AB)^{\mathrm{T}} - \lambda\mathbb{1}\big) = \det(BA - \lambda\mathbb{1}) = 0$. That is, AB and BA have the same characteristic equation, implying that their eigenvalues are the same.

3) For $n \times n$ matrices A and B prove AB and BA always have the same eigenvalues if B is invertible.

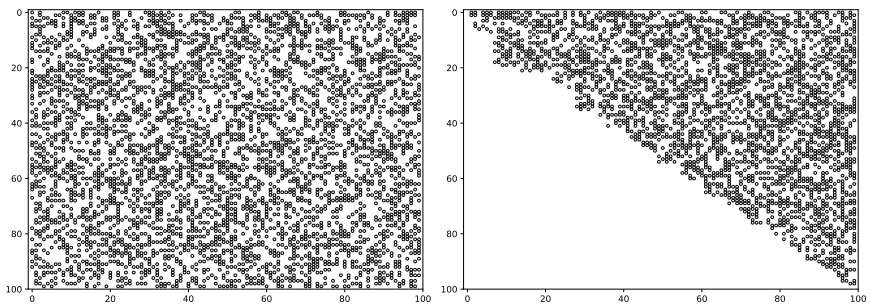
Note that similar matrices have the same eigenvalues, and since B is invertible AB is similar to $B(AB)B^{-1} = BA$, which implies that AB and BA have the same eigenvalues if B is invertible.

4) Starting with a random matrix $A \in \mathbb{R}^{100\times100}$, perform QR iteration 200 times. Plot the sparsity pattern for initial matrix A and final output matrix.

The code is shown below with the result. Note that we initialized our random matrix with integers between -5 and 5. For the sparsity pattern, the dots represent nonzero entries while

white space represents an effectively zero entry (i.e. $A_{ij} < 10^{-10}$). It is clear that the QR iteration transforms A into an upper triangular matrix.

```
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt

def my_spy(ax,A,marker='o'):
    n,m = np.shape(A)
    x = []
    y = []
    for i in range(n):
        for j in range(m):
            if A[j,i] < 1.0e-10:
                continue
            else:
                x.append(i)
                y.append(j)

    ax.scatter(x,y,color='k',marker='o',facecolor='none',s=10)
    ax.set_ylim(n,-1)
    ax.set_xlim(-1,n)


nrows=1;ncols=2
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*ncols,5*nrows))

n = 100
A = np.random.randint(-5,5,(n,n))

my_spy(ax[0],A)

N = 1000
for i in range(N):
    Q,R = np.linalg.qr(A,mode='complete')
    A = R@Q

my_spy(ax[1],A)

fig.tight_layout()
fig.savefig('prob4.pdf',bbox_inches='tight')
```



5) Consider $A \in \mathbb{R}^{10\times 10}$ of the form

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}. \tag{4}$$

a) Implement the power method to compute an approximation to the eigenvalue of largest absolute value and its corresponding eigenvector, starting with a random initial vector and stop the iteration with relative error 10^{-6}.

The code an results for this part are shown below:

```
#!/usr/bin/env python3

import numpy as np

n = 10
A = np.zeros([n,n])
for i in range(n):
    if i != 0:
        A[i,i-1] = -1.0
    if i != n-1:
        A[i,i+1] = -1.0
    A[i,i] = 2.0

v = np.random.randint(-10,10,n)
v = v/np.linalg.norm(v,2)
lam_old = None
iters = 0
while True:
    u = A@v
    lam = np.dot(u,v)
    u = u/np.linalg.norm(u)

    if lam_old == None:
        lam_old = lam
    elif np.abs((lam_old - lam)/lam_old) < 1e-6 and np.all(v - u) < 1e-6:
        break
    elif iters > 1e5:
        break

    lam_old = lam
    v = u
    iters += 1

np.set_printoptions(precision=3)
print()
print('\tlamda = {:.3f}'.format(lam))
print('\teigenvector:',v)
```

```
print()
```

```
lamda = 3.919
eigenvector: [-0.12   0.231 -0.322  0.388 -0.422  0.422 -0.388  0.322 -0.231  0.12 ]
```

b) Implement the QR algorithm (without shifts) to convert A to diagonal form and find out all the eigenvalues.

The code implementing the "basic" QR algorithm without shifts for calculation of the eigenvalues is shown below along with a screenshot of the resultant eigenvalues.

```
#!/usr/bin/env

import numpy as np

n = 10
A = np.zeros([n,n])
for i in range(n):
    if i != 0:
        A[i,i-1] = -1.0
    if i != n-1:
        A[i,i+1] = -1.0
    A[i,i] = 2.0

A_old = np.copy(A)
iters = 0
while True:
    Q,R = np.linalg.qr(A_old,mode='complete')
    A1 = R@Q

    if np.all(np.abs(A1 - A_old) < 1e-6):
        break
    elif iters > 1e6:
        break

    A_old = A1
    iters += 1

np.set_printoptions(precision=3)
print()
print('\teigenvalues:',np.array([A1[i,i] for i in range(n)]))
print()
```

```
eigenvalues: [3.919 3.683 3.31  2.831 2.285 1.715 1.169 0.69  0.317 0.081]
```

c) Take the 5th eigenvalue computed in your QR algorithm and, using it as a fixed shift in inverse iteration, compute the corresponding eigenvector.

The code implementing the inverse power method to calculate the eigenvector corresponding

to the the 5th eigenvalue calculated in part (b) is shown below along with a screenshot of the output.

```
#!/usr/bin/env python3

import numpy as np

n = 10
A = np.zeros([n,n])
for i in range(n):
    if i != 0:
        A[i,i-1] = -1.0
    if i != n-1:
        A[i,i+1] = -1.0
    A[i,i] = 2.0

shift = 2.28562968
I = np.eye(n)
v = np.random.randint(-10,10,n)
v = v/np.linalg.norm(v,2)
iters = 0
while True:
    u = np.linalg.solve((A-shift*I),v)
    lam = np.dot(u,v)
    u = u/np.linalg.norm(u)

    if np.all(v - u) < 1e-6:
        break
    elif iters > 1e5:
        break

    v = u
    iters += 1

np.set_printoptions(precision=3)
print()
print('\teigenvector:',v)
print()
```

```
eigenvector: [ 0.422 -0.12  -0.388  0.231  0.322 -0.322 -0.231  0.388  0.12  -0.422]
```