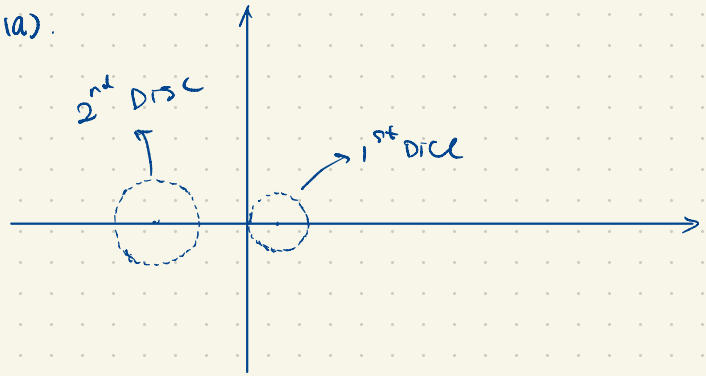
1. **[Gerschgorin, 2+1+1pt]** Gerschgorin's second theorem states that if the union of k Gerschgorin discs is disjoint from the other $n-k$ discs, it must contain exactly k eigenvalues. Now let

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

for some $z \in \mathbb{C}$. Here i is the imaginary unit.

(a) Sketch the first two Gerschgorin discs for A.

(a).



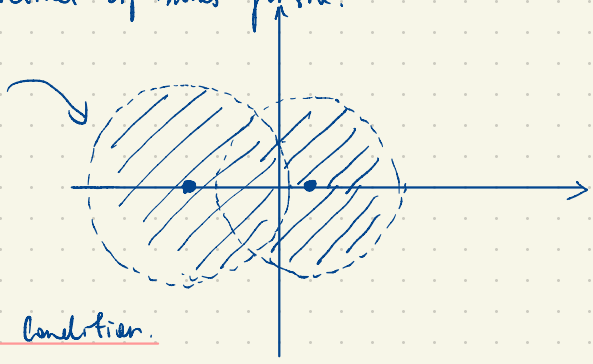
(b) Suppose we know that at least two of the three eigenvalues are equal. Using Gerschgorin's theorems, what can we conclude about the value of z? (Find the largest subset of $\mathbb{C}$ that you know z cannot be in)

At least 2 eigenvalues Equal $\Leftrightarrow$ Not the case of 3 distinct Eigenvalues.

The radius of the third disc is $|2i| = 2$

The third disk should at least touches an disk.
otherwise it would has 3 distinct eigenvalues for sure.

$$\begin{cases} dist(z, 1) \leq 1+2 \\ \text{or} \\ dist(z, -3) \leq 2+\sqrt{2} \end{cases}$$



$\Rightarrow$ This is a necessary Condition.

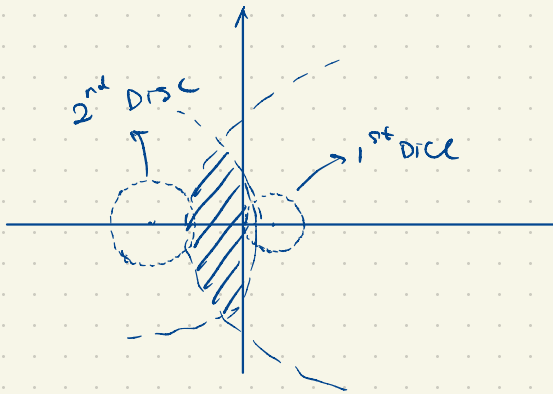
(c) Suppose we know that all eigenvalues are equal. What can we conclude about z?

Similarly, A sufficient condition for 3 eigenvalues are equal is

the 3 discs are not mutually disjoint.

That is to say,

$$\begin{cases} \operatorname{dist}(z, 1) \le 1+2 \\ \text{and} \\ \operatorname{dist}(|z|, -3) \le 2+\sqrt{2} \end{cases}$$



This is a necessary condition.

2. **[Power method and inverse iteration, 2+2+2+2pt]** Power Method and Inverse Iteration.

   (a) Implement the Power Method for an arbitrary symmetric matrix $A \in \mathbb{R}^{n\times n}$ and an initial vector $\boldsymbol{x}_0 \in \mathbb{R}^n$.

(a) See Code file.

   (b) Use your code to find an eigenvector of

$$A = \begin{bmatrix} -2 & 1 & 4 \\ 1 & 1 & 1 \\ 4 & 1 & -2 \end{bmatrix},$$

   starting with $x_0 = (1, 2, -1)^T$ and $x_0 = (1, 2, 1)^T$. Report the first 5 iterates for each of the two initial vectors. Then use the build-in function in MATLAB/Python/Julia (e.g., `eig(A)` in MATLAB) to examine the eigenvalues and eigenvectors of A. Where do the sequences converge to? Why do the limits not seem to be the same?

   (c) Implement the Inverse Power Method for an arbitrary matrix $A \in \mathbb{R}^{n\times n}$, an initial vector $\boldsymbol{x}_0 \in \mathbb{R}^n$ and an initial eigenvalue guess $\theta \in \mathbb{R}$.

   (d) Use your code from (c) to calculate *all* eigenvectors of A. You may pick appropriate values for θ and the initial vector as you wish (obviously not the eigenvectors themselves). Always report the first 5 iterates and explain where the sequence converges to and why.

3. **[Tridiagonalization with Householder, 4pt]** Use Householder matrices to transform the matrix A into tridiagonal form, i.e., find an orthogonal matrix Q such that $T = QAQ^T$ is tridiagonal.

$$A = \begin{bmatrix} 2 & 1 & 2 & 2 \\ 1 & -7 & 6 & 5 \\ 2 & 6 & 2 & -5 \\ 2 & 5 & -5 & 1 \end{bmatrix}.$$

1° Goal of H_1 – Convert $\begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$ into $\begin{bmatrix} * \\ 0 \\ 0 \end{bmatrix}$

Check $v = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 2 \end{bmatrix} \Rightarrow H_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\frac{1}{3} & -\frac{2}{3} & -\frac{2}{3} \\ 0 & -\frac{2}{3} & \frac{2}{3} & -\frac{1}{3} \\ 0 & -\frac{2}{3} & -\frac{1}{3} & \frac{2}{3} \end{bmatrix}$

$$A = \begin{bmatrix} 2 & -3 & 0 & 0 \\ -3 & 1 & 3 & 4 \\ 0 & 3 & -3 & -9 \\ 0 & 4 & -9 & -2 \end{bmatrix}$$

2° Goal of H_2 – Convert $\begin{bmatrix} 3 \\ 4 \end{bmatrix}$ into $\begin{bmatrix} * \\ 0 \end{bmatrix}$

$$v = \begin{bmatrix} 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \end{bmatrix}$$

$$\Rightarrow H_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{3}{5} & -\frac{4}{5} \\ 0 & 0 & -\frac{4}{5} & \frac{3}{5} \end{bmatrix}$$

$$\Rightarrow A = \begin{bmatrix} 2 & -3 & 0 & 0 \\ -3 & 1 & -5 & 0 \\ 0 & -5 & -11 & -3 \\ 0 & 0 & -3 & 6 \end{bmatrix} \leftarrow \text{Tridiagonalized form}$$

$$Q = H_2 H_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{3}{5} & -\frac{4}{5} \\ 0 & 0 & -\frac{4}{5} & \frac{3}{5} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\frac{1}{3} & -\frac{2}{3} & -\frac{2}{3} \\ 0 & -\frac{2}{3} & \frac{2}{3} & -\frac{1}{3} \\ 0 & -\frac{2}{3} & -\frac{1}{3} & \frac{2}{3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\frac{1}{3} & -\frac{2}{3} & -\frac{2}{3} \\ 0 & \frac{14}{15} & -\frac{2}{15} & -\frac{1}{3} \\ 0 & \frac{2}{15} & -\frac{11}{15} & \frac{2}{3} \end{bmatrix}$$

4. **[QR-algorithm, 1+1+2+2pts]** Let A be a symmetric, tridiagonal matrix. You learned that the matrices A_k defined by the QR-algorithm converge to a diagonal matrix that is similar to (and thus has the same eigenvalues as) A. The convergence speed depends on the absolute value of the ratio of consecutive eigenvalues. Let $r \in (0,1)$ and

$$A = \begin{bmatrix} 1 & r \\ r & 1 \end{bmatrix}$$

(a) Calculate the eigenvalues of A as a function of r (by hand).

(a) $\chi(A) = \det\left(\begin{bmatrix} \lambda-1 & -r \\ -r & \lambda-1 \end{bmatrix}\right) = (\lambda-1)^2 - r^2 = (\lambda-(1+r))(\lambda-(1-r))$

$\Rightarrow \lambda_1 = 1+r$

$\lambda_2 = 1-r$

(b) Implement the QR-algorithm using MATLAB's (or Python's) implementation of the QR-factorization, `qr()`. Your code should run for a quadratic matrix of any size.

(c) Now define a tolerance, e.g., $\tau = 10^{-10}$. Introduce a stopping criterion in your code, causing it to stop when the maximal difference between the true eigenvalues of A and the diagonal entries of A_k is smaller than τ.[1]

(d) Use your code with the matrix given for at least five values of $r \in (0,1)$ and make a plot with r versus the number of iterations needed to achieve the given tolerance. Explain your findings by examining the ratio between the eigenvalues of A using (a).

5. **[Google and eigenvectors, extra credit, 2+2+2pts]** The Google page rank algorithm, which is responsible for providing ordering search results, has a lot to do with the eigenvector corresponding to the largest eigenvalue of a so-called stochastic matrix, which describes the links between websites. Before working on this question, read the SIAM Review paper on the Linear Algebra behind Google.[2] Stochastic matrices have non-negative entries and each column sums to 1, and one can show (under a few technical assumptions) that it has the eigenvalues $\lambda_1 = 1 > |\lambda_2| \geq \ldots \geq |\lambda_n|$. Thus, we can use the power method[3] to find the eigenvector $\boldsymbol{v}$ corresponding to λ_1, which can be shown to have either all negative or all positive entries. These entries can be interpreted as the importance of individual websites.

   Let us construct a large stochastic matrices (pick a size $n \geq 100$, the size of our "toy internet") in MATLAB as follows:

```
I = eye(n);
A = 0.5*I(randperm(n),:)  + (max(2,randn(n,n))-2);
A = A - diag(diag(A));
L = A*diag(1./(max(1e-10,sum(A,1))));
```

   (a) Plot the sparsity structure of L (i.e., the nonzero entries in the matrix) using the command spy. Each non-zero entry corresponds to a link between two websites.

   (b) Plot the (complex) eigenvalues of L by plotting the real part of the eigenvalues on the x-axis, and the imaginary part on the y-axis.[4] Additionally, plot the unit circle and check that all eigenvalues are inside the unit circle, but $\lambda_1 = 1$.

   (c) The matrix L contains many zeros. One of the technical assumptions for proving theorems is that all entries in L are positive. As a remedy, one considers the matrices $S = \kappa L + (1 - \kappa)E$, where E is a matrix with entries $1/n$ in every component[5]. Study the influence of κ numerically by visualizing the eigenvalues of S for different values of κ. Why will $\kappa < 1$ improve the speed of convergence of the power method?

Please also hand in your code and output.

[1] You might want to sort the true and numerically computed eigenvalues before comparing them using `sort`.

[2] *The 25,000,000,000 eigenvector. The linear algebra behind Google* by Kurt Bryan and Tanya Leise. It's easy to find—just google it!

[3] We have discussed the power method for symmetric matrices, but it also works for non-symmetric matrices.

[4] Please make sure that the plotted eigenvalues are not connected by lines–that's confusing.

bigger k.
more sym the eigenvalue is

HW_5_code

April 17, 2022

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

0.1 2 (a)(b)

[2]:
```
def power_method(A, x_0, max_iter=49):
    '''
    Input: Symmetric matrix A, initial vector x_0
    Output: Eigenvalue with greates abs value
    '''
    assert np.allclose(A, A.T)
    assert len(x_0) == A.shape[1]

    history = []


    for i in range(max_iter):
        x_0 = A @ x_0
        x_0 = x_0 * (1/np.linalg.norm(x_0))
        history.append(x_0)

    vec = history[-1]
    val = float((vec.T @ A @ vec) / (vec.T @ vec))

    for i in range(5):
        print(f"The # {i+1} iteration x_0 = {history[i].T}^T")


    return vec, val
```

[3]:
```
A = np.array([
    [-2, 1, 4],
    [1, 1, 1],
    [4, 1, -2]
])
```

[4]:
```
x_0 = np.array([1, 2, -1]).reshape(-1, 1)
vec, val = power_method(A, x_0)

print("Eigenvactor: ", vec.T )
print("Eigenvalue: ", val)
```

```
The # 1 iteration x_0 = [[-0.43643578  0.21821789  0.87287156]]^T
The # 2 iteration x_0 = [[ 0.80829038  0.11547005 -0.57735027]]^T
The # 3 iteration x_0 = [[-0.64483142  0.05862104  0.7620735 ]]^T
The # 4 iteration x_0 = [[ 0.73561236  0.02942449 -0.67676337]]^T
The # 5 iteration x_0 = [[-0.69215012  0.0147266   0.72160331]]^T
Eigenvactor:  [[-7.07106781e-01  8.51170986e-16  7.07106781e-01]]
Eigenvalue:  -6.0
```

[5]:
```
x_0 = np.array([1, 2, 1]).reshape(-1, 1)
vec, val = power_method(A, x_0)

print("Eigenvactor: ", vec.T )
print("Eigenvalue: ", val)
```

```
The # 1 iteration x_0 = [[0.57735027 0.57735027 0.57735027]]^T
The # 2 iteration x_0 = [[0.57735027 0.57735027 0.57735027]]^T
The # 3 iteration x_0 = [[0.57735027 0.57735027 0.57735027]]^T
The # 4 iteration x_0 = [[0.57735027 0.57735027 0.57735027]]^T
The # 5 iteration x_0 = [[0.57735027 0.57735027 0.57735027]]^T
Eigenvactor:  [[0.57735027 0.57735027 0.57735027]]
Eigenvalue:  3.0
```

[6]:
```
# Use the built-in function to compute eigenvalues and eigenvectors
vals, vecs = np.linalg.eig(A)
print("Eigenvalues: ", vals)
print("Eigenvectors: \n", vecs)
```

```
Eigenvalues:  [-6.00000000e+00  3.00000000e+00  2.77080206e-16]
Eigenvectors:
 [[ 7.07106781e-01 -5.77350269e-01  4.08248290e-01]
 [-4.70543743e-17 -5.77350269e-01 -8.16496581e-01]
 [-7.07106781e-01 -5.77350269e-01  4.08248290e-01]]
```

- For $x_0 = (1, 2, -1)^T$, the sequences converge to the eigenvector corresponding to the eigenvalue of $\lambda = -6$.(Since the eigenvalue is negative, the eigenvector is oscillating between signs)
- For $x_0 = (1, 2, 1)^T$, the sequences converge to the eigenvector corresponding to the eigenvalue of $\lambda = 3$.
- The limits don't agree. Generally speaking, if everything works correctly, the sequence should converge to the eigenvector with the greatest eigenvalue (absolute value sense). It includes the setup that x_0 is not orthogonal the eigenvector. However, $x_0 = (1, 2, -1)^T$ happens to be orthogonal to the eigenvactor corresponding to $\lambda = 6$, which is $(1, 0, 1)^T$

0.2 2 (c)(d)

[7]:
```
def inverse_method(A, x_0, theta, max_iter=50):
    assert np.allclose(A, A.T)
    assert len(x_0) == A.shape[0]

    # we use "object" to denote the A - \theta Id
    object = A - theta * np.identity(A.shape[0])
    history = []

    for i in range(max_iter):
        x_0 = np.linalg.solve(object, x_0)
        x_0 = x_0 / np.linalg.norm(x_0)
        history.append(x_0)

    vec = history[-1]
    val = float((vec.T @ A @ vec) / (vec.T @ vec))

    for i in range(5):
        print(f"The # {i+1} iteration x_0 = {history[i].T}^T")


    return vec, val
```

[8]:
```
A = np.array([
    [-2, 1, 4],
    [1, 1, 1],
    [4, 1, -2]
])


# Here we pick \theta = {-0.1, 2.5, -7}
theta_space = [-0.1, 2.5, -7]

# x_0 = [5, 7, 11]^{T}
# x_0 is not orthogonal to any eigenvector
x_0 = np.array([5, 7, 11]).reshape(-1, 1)

for i, theta in enumerate(theta_space):
    print(f"#{i+1} REPORT (first 5)")
    vecs, vals = inverse_method(A, x_0, theta)
    print(f"#{i+1} Eigenvalues: ", vals)
    print(f"#{i+1} Eigenvectors: \n", vecs)
    print()
```

```
#1 REPORT (first 5)
The # 1 iteration x_0 = [[ 0.68281363 -0.45343555  0.57285404]]^T
```

```
The # 2 iteration x_0 = [[ 0.41690339 -0.80660941  0.41901411]]^T
The # 3 iteration x_0 = [[ 0.40858131 -0.81618127  0.40854553]]^T
The # 4 iteration x_0 = [[ 0.40825815 -0.81648641  0.40825876]]^T
The # 5 iteration x_0 = [[ 0.40824862 -0.81649625  0.40824861]]^T
#1 Eigenvalues:  0.0
#1 Eigenvectors:
 [[ 0.40824829]
 [-0.81649658]
 [ 0.40824829]]

#2 REPORT (first 5)
The # 1 iteration x_0 = [[0.58547157 0.58724305 0.55889949]]^T
The # 2 iteration x_0 = [[0.57757053 0.57534    0.57913398]]^T
The # 3 iteration x_0 = [[0.57719536 0.57775183 0.5771034 ]]^T
The # 4 iteration x_0 = [[0.57738772 0.57726994 0.57739313]]^T
The # 5 iteration x_0 = [[0.5773424  0.57736633 0.57734208]]^T
#2 Eigenvalues:  3.0
#2 Eigenvectors:
 [[0.57735027]
 [0.57735027]
 [0.57735027]]

#3 REPORT (first 5)
The # 1 iteration x_0 = [[-0.49148906  0.1509803   0.8576966 ]]^T
The # 2 iteration x_0 = [[-0.6870911   0.01485628  0.72641938]]^T
The # 3 iteration x_0 = [[-0.70506711  0.00134892  0.70913931]]^T
The # 4 iteration x_0 = [[-7.06893316e-01  1.15259479e-04  7.07320172e-01]]^T
The # 5 iteration x_0 = [[-7.07084036e-01  8.72113789e-06  7.07129526e-01]]^T
#3 Eigenvalues:  -6.0
#3 Eigenvectors:
 [[-7.07106781e-01]
 [ 3.55840713e-18]
 [ 7.07106781e-01]]
```

- The sequence always converges to the eigenvector corresponding to eigenvalue λ. E.g. if $\theta = 0.1$, it converges to the eigenvector with eigenvalue 0 since 0 is closest to $\theta = 0.1$
- It consistent with the nature of inverse power method, since it converges to the eigenvector whose eigenvalue closet to θ, which, in this case, is $\lambda = 0$. If we dig deeper, the nature of this property of inverse method is guaranteed by the essence of power method. Since the biggest eigenvalue for $(A - \theta I)^{-1}$ is produced by the eigenvalue which is closed to θ.

0.3 4(b)

[9]:
```
def tridiaglize(A):
    assert np.allclose(A, A.T)
    n = A.shape[0]
```

```
    for i in range(n-2):
        x = A[i+1:, i]
        e = np.zeros(x.shape)
        e[0] = 1

        v = (x + np.linalg.norm(x) * e).reshape(-1, 1)
        print("v:", v)
        h = np.identity(x.shape[0])  - (2 / (v.T @ v) ) * (v @ v.T)
        # Augment the householder into the full-size matrix
        H = np.identity(n)

        H[-h.shape[0]:, -h.shape[0]:] = h

        print("H: \n ",H)

        A = H @ A @ H
        print(H)
        print(A)



    return A
```

[10]:
```
def qr_algo(A):
    '''
    Input: A tridiagonal matrix
    Output: A diagonal matrix
    '''

    # convertit to the tridiagonal form

    A = tridiaglize(A)

    flag = True
    iters = 0

    while flag:
        Q, R = np.linalg.qr(A)
        A = R @ Q

        diag_mat = np.diag(A.diagonal())
        # Here we add a tolerance for stopping criterion
        if np.allclose(diag_mat, A):
            flag = False

        iters += 1
```

```
    return A
```

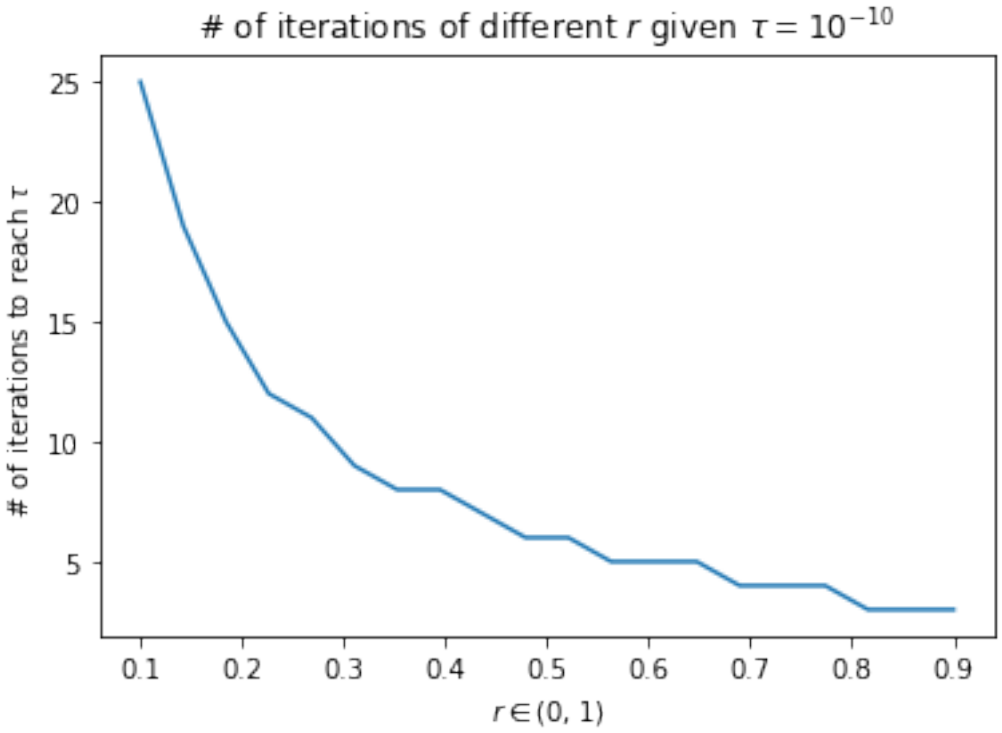
0.4 4(c)

[11]:
```
def qr_algo_tol(A, gt, tol=1e-10, ):
    '''
    Input: A tridiagonal matrix
    Output: A diagonal matrix
    '''

    # convertit to the tridiagonal form

    A = tridiaglize(A)

    flag = True
    iters = 0

    while flag:
        Q, R = np.linalg.qr(A)
        A = R @ Q

        vals = np.sort(A.diagonal())

        if np.allclose(vals, gt, atol=tol):
            flag = False

        iters += 1

    return A, iters
```

0.5 4(d)

[12]:
```
r_space = np.linspace(0.1, 0.9, 20)
iter_hist = []
for r in r_space:
    A = np.array([
        [1, r],
        [r, 1]
    ])

    gt = np.array([1-r, 1+r])

    vals, iters = qr_algo_tol(A, gt)

    iter_hist.append(iters)
```

[13]:
```
plt.plot(r_space, iter_hist)
plt.xlabel(r"$r\in (0, 1)$")
plt.ylabel(r"# of iterations to reach $\tau$")
plt.title(r"# of iterations of different $r$ given $\tau = 10^{-10}$")
plt.show()
```

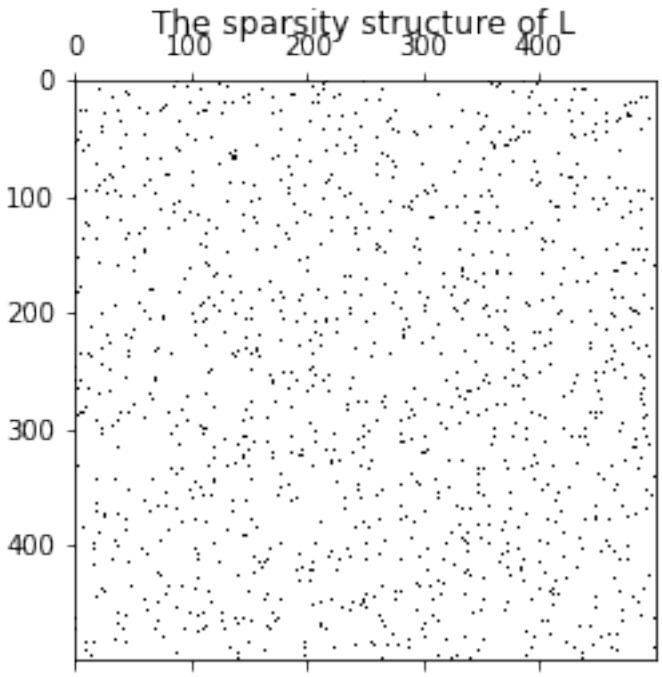


- When r become larger, the ratio between two eigenvalues, $\frac{1}{r}$, become smaller.
- In that case, the numbers of iterations needed to converger for QR-algorithm declines with the decline of ratio.

0.6 5(a)

[14]:
```
# set up a size of my toy internet
n = 500
I = np.identity(500)
A = 0.5 * I[np.random.permutation(n),:] + (np.maximum(2, np.random.
↪normal(size=(n, n))) - 2)
A = A - np.diag(A.diagonal())
L = A @ np.diag(1./ np.maximum(1e-10, np.sum(A, 1)))
```

[15]:
```
plt.spy(L)
plt.title("The sparsity structure of L")
plt.show()
```

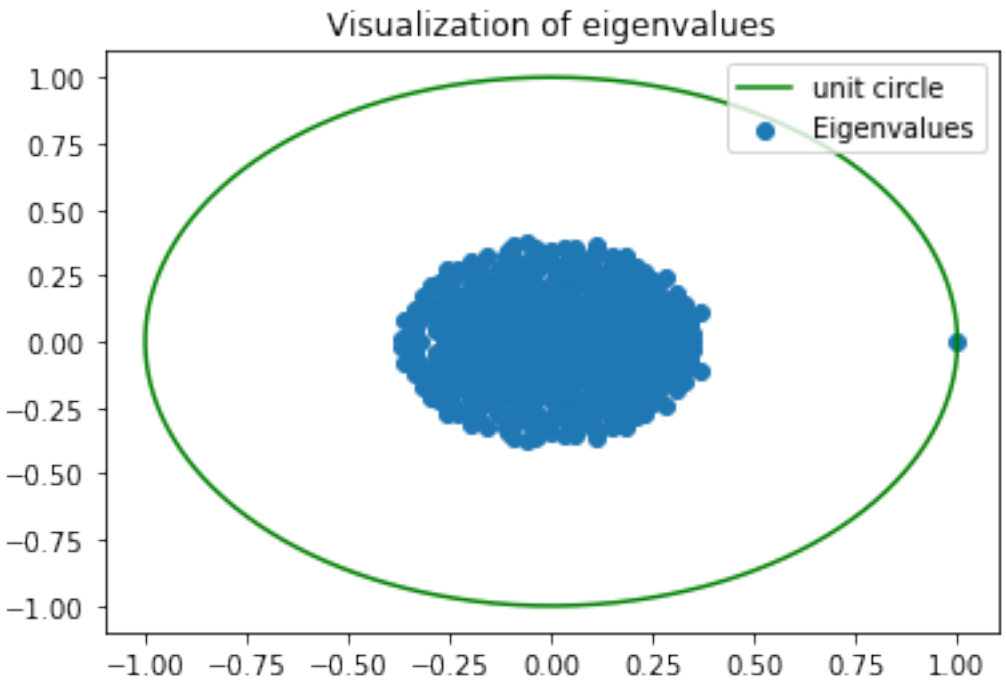


0.7 5(b)

[16]:
```
vals = np.linalg.eigvals(L)
```

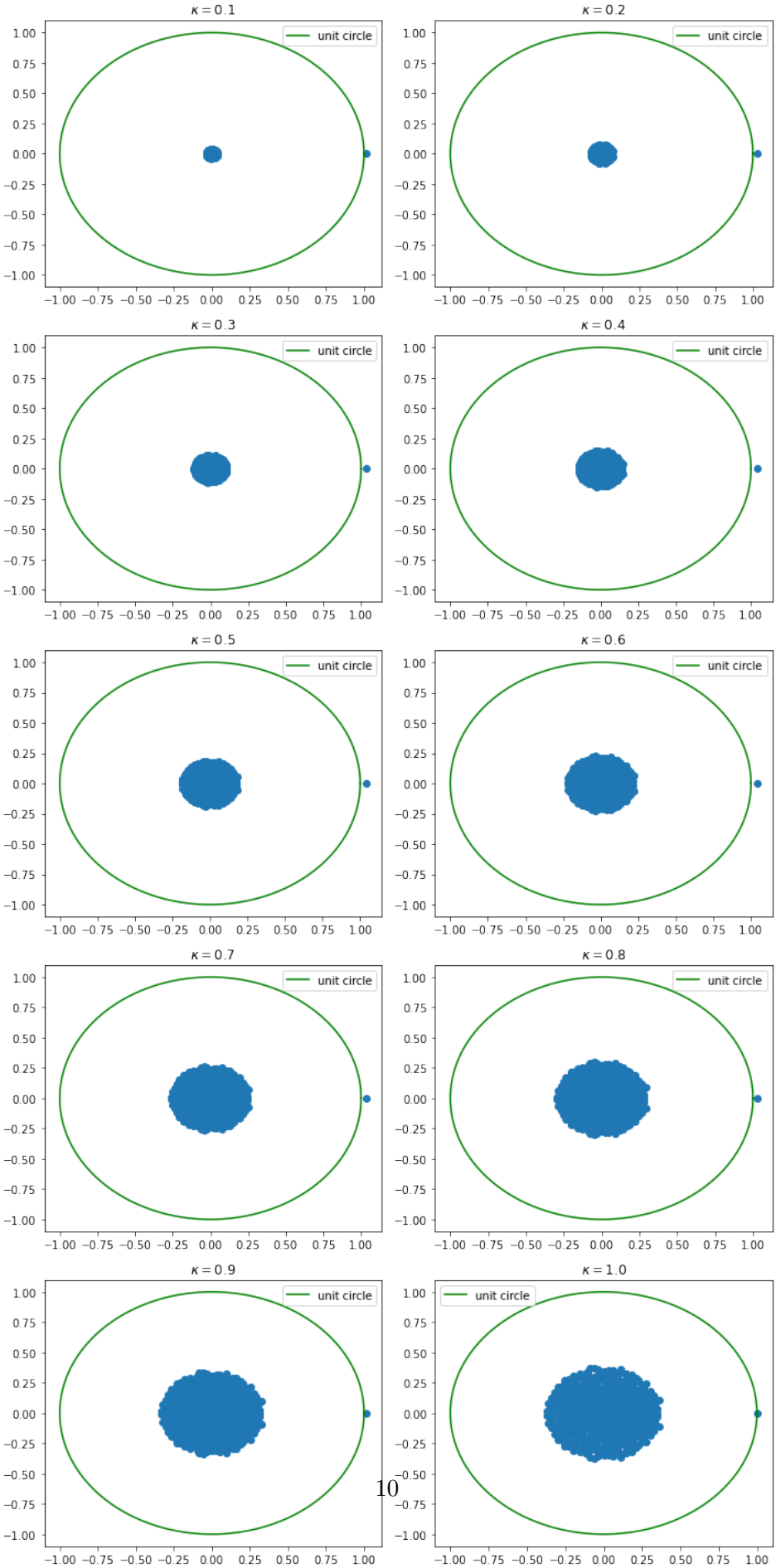
[17]:
```
plt.scatter(vals.real, vals.imag, label="Eigenvalues")

theta = np.linspace(0, 2*np.pi, 1000)

plt.plot(np.cos(theta), np.sin(theta), label="unit circle", c="g")
plt.title("Visualization of eigenvalues")
plt.legend()
```

[17]:
```
<matplotlib.legend.Legend at 0x7fa0c5f10ca0>
```



0.8 5(c)

[18]:
```
kappa_space = np.linspace(0.1, 1.0, 10)

E = np.array([1/n]* n**2).reshape(n, n)

fig, axs = plt.subplots(5, 2)


fig.set_size_inches(10, 20)


for i in range(10):
    kappa = kappa_space[i]
    S = kappa * L + (1 - kappa) * E
    S_vals = np.linalg.eigvals(S)
    axs[i//2][i%2].scatter(S_vals.real, S_vals.imag)
    axs[i//2][i%2].plot(np.cos(theta), np.sin(theta), label="unit circle",␣
 ↪c="g")
    axs[i//2][i%2].set_title(r"$\kappa = {:.1f}$".format(kappa_space[i]))
    axs[i//2][i%2].legend()


fig.tight_layout()
```


$\kappa = 0.1$
$\kappa = 0.2$
$\kappa = 0.3$
$\kappa = 0.4$
$\kappa = 0.5$
$\kappa = 0.6$
$\kappa = 0.7$
$\kappa = 0.8$
$\kappa = 0.9$
$\kappa = 1.0$
unit circle
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

- As we can see, by adding a $\kappa < 1$, the eigenvalues other than $\lambda = 1$(which we want) are more condensed to the origin. Therefore, when we are applying power method, it the ratio between the largest eigenvalue and second large eigenvalue is getting bigger, which will help the power method.