Connected to pytorch (Python 3.9.18)

In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
```

1.1 Power method

In [ ]:
```
def power_method(A, v, max_iter=500, tol=1e-8):

    for _ in range(max_iter):
        v_next = A @ v
        entry_pos = np.argmax(np.abs(v_next))
        entry_max = v_next[entry_pos,0]
        v_next = v_next/entry_max
        diff_norm_v = np.linalg.norm(v_next-v)

        v = v_next.copy()

        if diff_norm_v <= tol:
            break

    return entry_max, v/np.linalg.norm(v)

# verification
n = 4
A = np.random.random([n,n])+ np.eye(n)
x = np.random.random([n,1]) + 1.j*np.random.random([n,1])
a, v = power_method(A, x)

eigvals, eigvecs = np.linalg.eig(A)

print(f'found a = {np.round(a,3)}')
print('\nfound v =  \n', np.round(v.ravel(),3))
print('\nAv = \n', np.round((A@v).ravel(),3))
print('\nAv / v = (all = a) \n', np.round((A@v / v).ravel(),3))
```

```
print('\ncompare to numpy v ratio (unqiue value):\n', np.round(v.ravel()/eigvecs[:,0].ravel(),3))
```

```
found a = (3.003-0j)

found v =
 [0.322-0.j 0.596-0.j 0.674-0.j 0.296-0.j]

Av =
 [0.968-0.j 1.789-0.j 2.023-0.j 0.889-0.j]

Av / v = (all = a)
 [3.003-0.j 3.003-0.j 3.003-0.j 3.003+0.j]

compare to numpy v ratio (unqiue value):
 [1.-0.j 1.-0.j 1.-0.j 1.-0.j]
```

1.2 QR decomposition

In [ ]:

```
def myQR(A):
    n = len(A)
    Q = A.copy()
    R = np.zeros_like(A)

    #i=0
    q = A[:,0]
    b = np.linalg.norm(q)

    R[0,0] = b
    Q[:,0] = q/b

    for i in range(1,n):

        q = Q[:,i:i+1]
        a = np.sum(q*Q[:,:i].conj(),0)
        q = q.ravel() - np.sum(Q[:,:i]*a,1)
        q = q/np.linalg.norm(q)

        R[:i,i] = a
        R[i,i] = q.conj() @ A[:,i]
```

```
        Q[:,i] = q

    return Q, R


n = 4
A = np.random.random([n,n]) + 1.j *np.random.random([n,n]) + np.diag(np.arange(n))

Q, R = myQR(A)
print(f'residual of A-QR = {np.sum(np.abs(A-Q@R)):.02e}')
print(f'residual of QQ*-I = {np.sum(np.abs(Q@Q.conj().T-np.eye(n))):.02e}')
print(f'R is upper triangular = {np.all(np.triu(R)==R)}')
```

```
residual of A-QR = 1.76e-15
residual of QQ*-I = 1.32e-15
R is upper triangular = True
```

1.3 Permutation function

In [ ]:

```
def permutation(v):
    nn = len(v.ravel())
    idxOrder = np.argsort(np.abs(v))
    M = np.zeros((nn, nn))
    for i in range(nn):
        M[i, idxOrder[nn-1-i]] = 1
    return M

v = np.arange(10)
np.random.shuffle(v)
P = permutation(v)
print('random v :       ', v)
print('Pv =     :       ',P@v)
```

```
random v :        [9 3 2 0 4 5 7 6 1 8]
Pv =     :        [9. 8. 7. 6. 5. 4. 3. 2. 1. 0.]
```

1.4 QR algorithm

In [ ]:

```
def QR_algo(A, tol=1e-4):
    diff = 1
    Ak = A.copy()
    nn = A.shape[0]
    V = np.eye(nn)
    i = 0

    while diff > tol:
        i+=1
        shift = complex(np.random.rand(1),np.random.rand(1))/(n+1) * np.eye(nn)
        A_tmp = Ak.copy()
        # Q,R = np.linalg.qr(Ak - shift)
        Q,R = myQR(Ak - shift)
        Ak = R@Q + shift
        Perm = permutation(np.diag(Ak))
        Ak = Perm.T@Ak@Perm
        V = V@Q@Perm
        diff = np.sum(np.abs(np.sort(np.diag(Ak))-np.sort(np.diag(A_tmp))))
    return Ak, V

n = 4
A = np.random.random([n,n])+1.j*np.random.random([n,n]) + np.diag(np.arange(n))
A = A+A.T.conj()

Ak,V = QR_algo(A)
print('Ak = \n', np.round(Ak,2))
print(f'\nresidual of VV*-I = {np.sum(np.abs(V@V.conj().T-np.eye(n))):.02e}')
print(f'\nresidual of V*DV-A = {np.sum(np.abs(V@Ak@V.conj().T - A)):.02e}' )
print(f'\nv is first col of V, Av / v =  {np.round(A@V[:,0] / V[:,0],2)}')
```

```
Ak =
 [[ 8.25-0.j  0.  +0.j  0.  -0.j  0.  -0.j]
 [ 0.  -0.j  4.54+0.j -0.  -0.j  0.  -0.j]
 [ 0.  +0.j -0.  +0.j  2.44-0.j -0.  +0.j]
 [ 0.  +0.j  0.  +0.j -0.  -0.j  0.74-0.j]]

residual of VV*-I = 6.56e-15

residual of V*DV-A = 2.89e-14

v is first col of V, Av / v =  [8.25+0.j 8.25+0.j 8.24+0.j 8.25-0.j]
```

2. Page Rank

In [ ]:
```
A = np.array([[0,  0,    1,  0,    0],
              [0,  0,    0,  1/2,  1/2],
              [1  ,0,    0,  0,    0],
              [0,  1/2,  0,  0  ,  1/2],
              [0,  1/2,  0,  1/2,  0]], dtype='float')
```

2.1 10 trials with different initial guesses

`Observation` : Power Method outputs different outputs each time (reached max_iter). We did NOT get eigen vectors.

In [ ]:
```
for i in range(10):
    a, v = power_method(A, np.random.random([5,1]) + 1.j*np.random.random([5,1]), max_iter=int(1e5))
    print(f'\n{i+1}th trial:\na = {np.round(a,2)}, v = {np.round(v.ravel(),2)}')
    print(f'(A * v) / v   = {np.round((A@v).ravel()/v.ravel(),2)}')
```

```
1th trial:
a = (1+0j), v = [0.67+0.j   0.3 +0.14j 0.47+0.02j 0.3 +0.14j 0.3 +0.14j]
(A * v) / v   = [0.7 +0.03j 1.  +0.j   1.42-0.05j 1.  +0.j   1.  +0.j  ]

2th trial:
a = (1+0j), v = [0.49+0.j   0.39+0.18j 0.23+0.4j  0.39+0.18j 0.39+0.18j]
(A * v) / v   = [0.47+0.81j 1.  +0.j   0.53-0.92j 1.  +0.j   1.  +0.j  ]

3th trial:
a = (1+0j), v = [0.17-0.08j 0.35-0.09j 0.76+0.j   0.35-0.09j 0.35-0.09j]
(A * v) / v   = [3.64+1.81j 1.  +0.j   0.22-0.11j 1.  -0.j   1.  +0.j  ]

4th trial:
a = (1+0j), v = [0.28+0.17j 0.48+0.j   0.39-0.2j  0.48+0.j   0.48+0.j  ]
(A * v) / v   = [0.68-1.13j 1.  +0.j   0.39+0.65j 1.  +0.j   1.  +0.j  ]

5th trial:
a = (1+0j), v = [0.3 -0.32j 0.46+0.j   0.39+0.15j 0.46+0.j   0.46+0.j  ]
(A * v) / v   = [0.37+0.9j  1.  +0.j   0.39-0.95j 1.  +0.j   1.  +0.j  ]

6th trial:
a = (1+0j), v = [0.42+0.14j 0.35-0.23j 0.52+0.j   0.35-0.23j 0.35-0.23j]
(A * v) / v   = [1.1 -0.37j 1.  +0.j   0.82+0.28j 1.  +0.j   1.  -0.j  ]

7th trial:
a = (1+0j), v = [0.42+0.19j 0.29-0.21j 0.65+0.j   0.29-0.21j 0.29-0.21j]
(A * v) / v   = [1.29-0.59j 1.  +0.j   0.64+0.29j 1.  +0.j   1.  +0.j  ]

8th trial:
a = (1+0j), v = [0.15-0.42j 0.32-0.23j 0.58+0.j   0.32-0.23j 0.32-0.23j]
(A * v) / v   = [0.43+1.23j 1.  +0.j   0.25-0.73j 1.  +0.j   1.  +0.j  ]

9th trial:
a = (1+0j), v = [0.12-0.19j 0.44-0.19j 0.51+0.j   0.44-0.19j 0.44-0.19j]
(A * v) / v   = [1.16+1.94j 1.  +0.j   0.23-0.38j 1.  +0.j   1.  +0.j  ]

10th trial:
a = (1+0j), v = [0.54+0.j   0.45-0.11j 0.26+0.08j 0.45-0.11j 0.45-0.11j]
(A * v) / v   = [0.49+0.15j 1.  +0.j   1.85-0.57j 1.  +0.j   1.  +0.j  ]
```

2.2 Given initial guess

`Observation` : Both time we converged to some vector which is close to the initial guess and the outputs are indeed eigenvectors associated to eigenvalue 1.

In [ ]:
```
x1 = np.array([0. , 0.57, 0. , 0.57, 0.57])[:,None]
x2 = np.array([0.7, 0., 0.7, 0., 0.])[:,None]

for  i in [1,2]:
    print(f'\ninitial guess = x{i} : \n')
    exec(f"a, v = power_method(A, x{i}, max_iter=int(1e10))")
    print(f'a = {np.round(a,3)}, v = {np.round(v.ravel(),3)}')
    print(f'A * v = {np.round((A@v).ravel(),3)}')
    # print(f'(A * v) / v = {np.round((A@v).ravel()/v.ravel(),3)}')
```

```
initial guess = x1 :

a = 1.0, v = [0.    0.577 0.    0.577 0.577]
A * v = [0.    0.577 0.    0.577 0.577]

initial guess = x2 :

a = 1.0, v = [0.707 0.    0.707 0.    0.   ]
A * v = [0.707 0.    0.707 0.    0.   ]
```

2.3 Why?

`Discussion` : The Power Method is unstable with this problem because A's principal eigenspace is of dimension 2, which violates the hypothesis which guarantees the convergence of Power Method.

2.4 Perturbed PageRank

`Answer` : With a small lambda, we found a vector which is in the 1-eigenspace of A.

In [ ]:
```
alpha = 1/20
Aprime = (1-alpha)*A + alpha * np.ones([5,5])/5
```

```
a, v = power_method(Aprime, np.random.random([5,1])+1.j*np.random.random([5,1]), max_iter=int(1e15))
print(f'perturbed A:\na = {np.round(a,3)}, v = {np.round(v.ravel(),3)}')
print(f'(A * v) / v   = {np.round((A@v).ravel()/v.ravel(),3)}')
```

```
perturbed A:
a = (1-0j), v = [0.447+0.j 0.447+0.j 0.447-0.j 0.447+0.j 0.447+0.j]
(A * v) / v   = [1.-0.j 1.+0.j 1.+0.j 1.+0.j 1.+0.j]
```

We also note that with this perturbation, the dimension of the eigenspace of A' associated to 1 is reduced to 1. Due to this, Power Method can converge.

In [ ]:
```
a, v = np.linalg.eig(Aprime)
a
```

Out[ ]:
```
array([-0.95 ,  0.95 ,  1.   , -0.475, -0.475])
```

2.5 Eigenvalues of A

In [ ]:
```
Ak,V = QR_algo(A, tol=1e-8)
print(np.round(np.diag(Ak),2))
```

```
[ 1. +0.j -1. -0.j  1. -0.j -0.5-0.j -0.5+0.j]
```

In [ ]:
```
eigval, _ = np.linalg.eig(A)
print(eigval)
```

```
[ 1.  -1.  -0.5  1.  -0.5]
```