proj_2_EigenTest

May 14, 2024

[ ]:
```
from P2_module import *
```

[ ]:
```
def generate_matrix(n, type='symmetric'):
    if type not in ['symmetric', 'hermitian']:
        raise ValueError("Type must be 'symmetric' or 'hermitian'")

    A = np.random.rand(n, n)

    if type == 'symmetric':
        A = (A + A.T) / 2
    elif type == 'hermitian':
        B = np.random.rand(n, n)
        A = A + 1j * B
        A = (A + A.conj().T) / 2
        np.fill_diagonal(A, np.real(A.diagonal()))

    return A

def compare_complex_vectors(v1, v2, tolerance=1e-4):
    if np.allclose(v1, v2, atol = tolerance):
        return True
    for i in range(v1.size):
        if np.abs(v2[i]) > 1e-3:
            ratio = v1[i] / v2[i]
            v1 = v1 / ratio
            break
    return np.allclose(v1, v2, atol=tolerance, rtol=tolerance)

def compare_eigen_system(d_numpy, d_myfunc, U_numpy, U_myfunc, tol = 1e-4):
    flag = True
    d_numpy, U_numpy = order_eigensystems(d_numpy, U_numpy)
    n = d_numpy.size
    for i in range(n):
        if not np.allclose(d_numpy[i], d_myfunc[i], atol = tol, rtol = tol):
            flag = False
    for i in range(n):
        if not compare_complex_vectors(U_numpy[:,i], U_myfunc[:,i],␣
 ↪tolerance=tol):
```

```
            flag = False

    return flag

def comphrehensive_test(n = 30):
    flag = True
    count_symmetric = 0
    fail_symmetric = []
    for i in range(2, n+1):
        rho = generate_matrix(i, type='symmetric')
        d_numpy, R_numpy = np.linalg.eig(rho)
        d_myfunc, R_myfunc = hermitian_eigensystem(rho, 1e-15)
        if not compare_eigen_system(d_numpy, d_myfunc, R_numpy, R_myfunc):
            flag = False
            count_symmetric = count_symmetric+1
            fail_symmetric.append(d_myfunc.size)

    count_hermitian = 0
    failed = []
    for i in range(2, n+1):
        rho = generate_matrix(i, type='hermitian')
        d_numpy, R_numpy = np.linalg.eig(rho)
        d_myfunc, R_myfunc = hermitian_eigensystem(rho, 1e-15)
        if not compare_eigen_system(d_numpy, d_myfunc, R_numpy, R_myfunc):
            flag = False
            count_hermitian = count_hermitian + 1
            failed.append(d_myfunc.size)
    print('Symmetric failed: ', count_symmetric)
    print('Hermitian failed: ', count_hermitian)
    print('Symmemtric index: ', fail_symmetric)
    print('Hermitian index: ', failed)
    return flag
```

1 Simple checks

[ ]:

```
rho = generate_matrix(20, type='symmetric')
d_myfunc, R_myfunc = hermitian_eigensystem(rho, 1e-15)
```

$d[0] <= d[1] <= ... <= d[n-1]$ (where n is the dimension of H)

[ ]:

```
np.all(np.diff(d_myfunc) > 0)
```

[ ]:

```
True
```

np.transpose(U) * U = U * np.transpose(U) = np.eye(n)

[ ]:

```
np.allclose(R_myfunc.T @ R_myfunc, np.eye(d_myfunc.size), atol=1e-5, rtol=1e-5)
```

[ ]: True

H * U[:,k] = d[k] *   U[:,k] k=0,1,2,...,(n-1)

[ ]:
```python
flag = True
for i in range(d_myfunc.size):
    if not np.allclose(rho@R_myfunc[:,i], d_myfunc[i]*R_myfunc[:,i], atol = 1e-5, rtol = 1e-5):
        print(i)
        flag = False
print(flag)
```

```
True
```

2 Comprehensive test

[ ]:
```python
flag = True
for i in range(5):
    print('{}th comphrehensive_test'.format(i+1))
    if not comphrehensive_test():
        flag = False

assert flag == True
print("All test passed")
```

```
1th comphrehensive_test
Symmetric failed:  0
Hermitian failed:  0
Symmemtric index:  []
Hermitian index:  []
2th comphrehensive_test
Symmetric failed:  0
Hermitian failed:  0
Symmemtric index:  []
Hermitian index:  []
3th comphrehensive_test
Symmetric failed:  0
Hermitian failed:  0
Symmemtric index:  []
Hermitian index:  []
4th comphrehensive_test
Symmetric failed:  0
Hermitian failed:  0
Symmemtric index:  []
Hermitian index:  []
5th comphrehensive_test
Symmetric failed:  0
Hermitian failed:  0
```

```
Symmemtric index:  []
Hermitian index:  []
All test passed
```