f-26-jupyter-praktisk-qr

May 6, 2021

[1]:
```python
import numpy as np
```

[2]:
```python
def house_eps(x):
    norm_x = np.linalg.norm(x)
    if norm_x == 0:
        v = np.zeros_like(x)
        v[0] = 1
        s = 0
        eps = 1
    else:
        u = x / np.linalg.norm(x)
        eps = -1 if u[0] >= 0 else +1
        s = 1 + np.abs(u[0])
        v = - eps * u
        v[0] += 1
        v /= s
    return v, s, eps

def householder_qr_data(a):
    data = np.copy(a)
    k = a.shape[1]
    s = np.empty(k)
    for j in range(k):
        v, s[j], eps = house_eps(data[j:, [j]])
        data[j:, j:] -= s[j] * v @ (v.T @ data[j:, j:])
        data[j+1:, [j]] = v[1:]
    return data, s

def householder_qr(a):
    data, s = householder_qr_data(a)
    n, k = a.shape
    r = np.triu(data[:k, :k])
    q = np.eye(n, k)
    for j in reversed(range(k)):
        x = data[j+1:, [j]]
        v = np.vstack([[1], x])
        q[j:, j:] -= s[j] * v @ (v.T @ q[j:, j:])
    return q, r
```

[3]:
```
def tridiagonal_data(a):
    data = np.copy(a)
    if not np.allclose(a, a.T):
        raise np.linalg.LinAlgError(
            'In tridiagonal_data() input must be a symmetric matrix')
    n = a.shape[0]
    s = np.empty(n - 2)
    for j in range(n - 2):
        v, s[j], eps = house_eps(data[j+1:, [j]])
        u = s[j] * (data[j+1:, j+1:] @ v)
        w = u - (((s[j]/2) * u.T) @ v) * v
        v_wT = v @ w.T
        data[j+1, j] = eps * np.linalg.norm(data[j+1:, [j]])
        data[j, j+1] = data[j+1, j]
        data[j+1:, j+1:] -= v_wT + v_wT.T
        data[j+2:, [j]] = v[1:]
    return data, s
```

[4]:
```
def tridiagonal(a):
    data, s = tridiagonal_data(a)
    return np.tril(np.triu(data, -1), 1)
```

[5]:
```
def tridiagonal_qt(a):
    data, s = tridiagonal_data(a)
    n = a.shape[0]
    t = np.tril(np.triu(data, -1), 1)
    q = np.eye(n)
    for j in reversed(range(n-2)):
        x = data[j+2:, [j]]
        v = np.vstack([[1], x])
        q[j+1:, j+1:] -= s[j] * v @ (v.T @ q[j+1:, j+1:])
    return q, t
```

[6]:
```
a = np.array([[1., 2., 3.],
              [2., -1., 2.],
              [3.,2.,0]])
print(a)
```

```
[[ 1.  2.  3.]
 [ 2. -1.  2.]
 [ 3.  2.  0.]]
```

[7]:
```
q, t = tridiagonal_qt(a)
t
```

[7]:
```
array([[ 1.        , -3.60555128,  0.        ],
       [-3.60555128,  1.53846154,  0.30769231],
       [ 0.        ,  0.30769231, -2.53846154]])
```

[8]:
```
q @ t @ q.T
```

[8]:
```
array([[ 1.00000000e+00,  2.00000000e+00,  3.00000000e+00],
       [ 2.00000000e+00, -1.00000000e+00,  2.00000000e+00],
       [ 3.00000000e+00,  2.00000000e+00,  7.77156117e-16]])
```

[9]:
```
rng = np.random.default_rng()
```

[10]:
```
n = 30
a = rng.normal(0.0, 5.0, (n, n))
a = (a + a.T) / 2
q, t = tridiagonal_qt(a)
print(np.allclose(q @ q.T, np.eye(n), atol=2*np.finfo(float).eps))
print(np.allclose(q @ t @ q.T, a, atol=np.finfo(float).eps))
print(np.abs((q @ q.T - np.eye(n))).max())
print(np.abs((q @ t @ q.T - a)).max())
```

```
True
True
6.661338147750939e-16
9.325873406851315e-15
```

[11]:
```
def semi_praktisk_qr_metode(a):
    n = a.shape[0]
    q, t = tridiagonal_qt(a)
    for i in range(20):
        mu_eye = t[-1,-1] * np.eye(n)
        q, r = householder_qr(t - mu_eye)
        t = r @ q + mu_eye
        if np.allclose(np.diag(t, -1), np.zeros(n-1),
                       atol = 10 * np.finfo(float).eps):
            break
    return t, i
```

[12]:
```
t, i = semi_praktisk_qr_metode(a)
```

[13]:
```
i
```

[13]:
```
19
```

[14]:
```
np.diag(t)
```

[14]:
```
array([ 36.31148313,  33.39317929, -28.37592104,  22.51685067,
       -28.40477405,  25.56944076, -19.84088295,  14.01177402,
        -8.24670037,   7.18368108,  -0.4300536 ,  -5.70790802,
        12.81810956, -15.34470471,  13.22300418, -14.34756173,
         9.68044414,   7.21312496, -11.50723249,   4.39546461,
       -11.80133313,  -3.22773161,  -2.81433904,   3.22104928,
```

```
        -8.13578479,  -1.22701714,  -5.30277257,  -5.94000722,
        -2.88172435,  -3.31782051])
```

[15]:
```
np.diag(t, -1)
```

[15]:
```
array([ 2.35248320e-01,  3.06973958e+00,  1.32140168e+01,  4.79996514e+00,
        4.25677497e+00, -7.52025293e+00,  1.49636680e+01,  2.59422461e+00,
       -2.24002521e+01, -2.58322023e+00,  1.92679150e+01,  1.32755205e+00,
       -9.77045836e+00, -2.93033852e+00,  4.80350011e+00,  4.42367871e+00,
        1.19576233e+00,  8.06164066e+00,  1.54068075e+00, -7.76610432e+00,
        2.16751162e-01,  8.18628421e+00,  7.80875262e-02, -1.59706401e-01,
       -1.01804804e-02,  3.00383351e+00, -7.44938900e-03, -2.50544570e-15,
        0.00000000e+00])
```

[16]:
```
np.linalg.eig(a)[0]
```

[16]:
```
array([ 36.33156217,  33.57958665, -32.16481406,  27.80615384,
        26.01019548,  23.87535367, -28.82806515, -26.58785543,
       -24.55552359, -22.14131193,  18.60600873,  16.90088201,
        15.45378432,  13.96704096, -18.68652573,  11.3595575 ,
         9.24473203,   7.48830035, -15.83987641, -15.32080253,
       -14.19412956, -11.20455618,   5.16911606,   3.22189502,
         0.36498729,  -8.13805788,  -6.89479266,  -5.93996378,
        -3.31782051,  -2.88172435])
```

[17]:
```
n = 10
a = rng.normal(0.0, 5.0, (n, n))
a = (a + a.T) / 2
```

[18]:
```
t, i = semi_praktisk_qr_metode(a)
print(np.diag(t), i)
```

```
[ 26.72516365 -22.07934905  16.00225936 -12.69784081   9.22403803
   6.58651917  -7.85478628  -0.79810945  -1.74232305  -1.74156886] 19
```

[19]:
```
np.linalg.eig(a)[0]
```

[19]:
```
array([ 26.72546356, -22.09063067,  16.01324117, -12.97900782,
         9.50560222,   6.59716388,  -7.86582827,  -4.89679081,
         2.35635831,  -1.74156886])
```

[20]:
```
np.diag(t, -1)
```

[20]:
```
array([-1.20935693e-01, -6.46789403e-01,  1.70121955e-03,  2.49854856e+00,
        3.42479744e-02, -3.99477203e-01,  1.12386920e-05,  3.59571387e+00,
        0.00000000e+00])
```

[21]:
```python
def wilkinson_qr_metode(a):
    n = a.shape[0]
    q, t = tridiagonal_qt(a)
    for i in range(20):
        delta = (t[-2,-2] - t[-1,-1]) / 2
        eps = 1 if delta >=0 else -1
        mu = t[-1,-1] - eps * t[-1, -2] / (np.abs(delta)
                                           + np.sqrt(delta**2 + t[-1,-2]**2))
        mu_eye = mu * np.eye(n)
        q, r = householder_qr(t - mu_eye)
        t = r @ q + mu_eye
        if np.allclose(np.diag(t, -1), np.zeros(n-1),
                       atol = np.finfo(float).eps):
            break
    return t, i
```

[22]:
```python
t, i = wilkinson_qr_metode(a)
print(np.diag(t), i)
```

```
[ 26.72527073 -22.07109498  15.99389826 -12.26293855   8.78904605
   6.5951311   -7.86330855   2.10401196  -4.64444446  -1.74156886] 19
```

[23]:
```python
def praktisk_qr_metode(a):
    n = a.shape[0]
    if n == 1:
        t = a
        i = 0
        return t, i
    q, t = tridiagonal_qt(a)
    for i in range(20):
        delta = (t[-2,-2] - t[-1,-1]) / 2
        eps = 1 if delta >=0 else -1
        mu = t[-1,-1] - eps * t[-1, -2] / (np.abs(delta)
                                           + np.sqrt(delta**2 + t[-1,-2]**2))
        mu_eye = mu * np.eye(n)
        q, r = householder_qr(t - mu_eye)
        t = r @ q + mu_eye
        underdiag_abs = np.abs(np.diag(t, -1))
        zz = np.argwhere(underdiag_abs < np.finfo(float).eps)
        if zz.shape[0] == underdiag_abs.shape[0]:
            break
        if zz.shape[0] != 0:
            zj = zz[0, 0] + 1
            t[:zj, :zj], j = praktisk_qr_metode(t[:zj, :zj])
            t[zj:, :zj], k = praktisk_qr_metode(t[zj:, zj:])
            i += j + k
            break
    return t, i
```

[24]:
```python
t, i = praktisk_qr_metode(a)
```

[25]:
```python
print(np.diag(t), i)
```

```
[ 26.72546356 -22.09063067  16.01324117 -12.97900782   9.50560222
   6.59716388  -7.86582827  -4.89679081   2.35635831  -1.74156886] 23
```

[26]:
```python
np.linalg.eig(a)[0]
```

[26]:
```
array([ 26.72546356, -22.09063067,  16.01324117, -12.97900782,
         9.50560222,   6.59716388,  -7.86582827,  -4.89679081,
         2.35635831,  -1.74156886])
```

[27]:
```python
np.sort(np.diag(t)) - np.sort(np.linalg.eig(a)[0])
```

[27]:
```
array([-8.52651283e-14, -4.61852778e-14,  5.32907052e-15, -8.88178420e-16,
       -1.11022302e-15,  8.88178420e-16, -1.33226763e-14, -1.59872116e-14,
        2.13162821e-14,  3.90798505e-14])
```

[ ]: