QR decomposition using HouseHolder process (input example)

فاطمه صفری
ش.د ۹۷۲۸۰۶۳

لینک کد

در این کد با استفاده از پروسه‌ی هاوس هولدر میتوانیم یک ماتریس را از کاربر گرفته، آن را به یک ماتریس مثلثی R و یک ماتریس متعامد Q تبدیل و چاپ کنیم.
توضیح کلی هر قطعه کد:

```
import numpy as np
```

این کد از ماژول numpy برای محاسبات ماتریسی یعنی داده‌های عددی و تبدیل آنها به آرایه و... استفاده میکند.

```
def qr_decomposition(A, threshold=1e-10):
    m, n = A.shape

    Q = np.eye(m)

    R = A.astype(float).copy()
```

کد بالا تابعی برای ماتریس A و (آستانه‌ی اعداد کوچکتر از e^{-10} تعریف کرده که بعدا این اعداد را برابر صفر قرار میدهیم) به نام qr_decomposition که در آن ابتدا شکل ماتریس A را بدین صورت تعریف کردیم که m و n به ترتیب برابر تعداد سطر و ستون آن باشد. سپس ماتریس Q را تعریف کردیم به طوریکه برابر با ماتریس همانی مربعی با سطر و ستون m باشد. در قدم های بعدی این ماتریس همانی به روز میشود.
برای ماتریس R نیز ابتدا آن را برابر با یک کپی از ماتریسی که از کاربر دریافت کردیم گذاشتیم و داده های دریافت شده را تبدیل به تایپ float کردیم.

```
for j in range(n):
    a.      x = R[j:, j]
    b.      e = np.zeros_like(x)
    c.      e[0] = np.sign(x[0] or 1)
    d.      u = x + np.linalg.norm(x) * e
    e.      u /= np.linalg.norm(u)
    f.      R[j:, :] -= 2.0 * np.outer(u, u @ R[j:, :])
    g.      R[np.abs(R) < threshold] = 0.0
    h.      Q[:, j:] -= 2.0 * Q[:, j:] @ np.outer(u, u)
    i.      Q[np.abs(Q) < threshold] = 0.0
    j. return Q, R
```

در قطعه کد بالا حلقه ای تعریف کردیم که روی ستون های R جلو می رود.

a. ابتدا x یک آرایه‌ی ماتریس R را انتخاب میکند که از سطر و ستون j ام شروع میشود. این آرایه در واقع زیر و روی قطر اصلی ماتریس R در (j,j) است.

b. حال e را برابر با همان ابعاد x قرار داده و آرایه هایش را در مرحله اول صفر میگذاریم.

c. در مرحله بعد اولین عنصر x را برابر اولین عنصر e گذاشته و اگر این عدد برابر 0 بود آن را برابر 1 قرار میدهیم که برای بدست آوردن نرمال شده‌ی بردار هاوس هولدر به ارور تقسیم بر صفر بر نخوریم.

d. از این خط شروع به محاسبه بردار هاوس هولدر میکنیم. ابتدا بردار u را با ضرب کردن نرم اقلیدسی x و e و جمع کردن آن با بردار x بدست میاوریم.

e. با تقسیم بردار u بر نرم اقلیدسی آن، آن را نرمال می کنیم.

f. حالا زیر آرایه های R را آپدیت میکنیم: با شروع از سطر j ام و به ترتیب تا آخر، ضرب خارجی بردار u در خودش، ضرب در ۲ را از آن سطر کم

میکنیم. این کار درواقع دارد عناصر زیر قطر اصلی را صفر میکند تا ماتریس بالا مثلثی بسازد.

g. این خط عناصری از R را که قدر مطلق شان از عددی که اول تابع تعریف کردیم کمتر هستند را برابر با صفر قرار میدهد.

h. زیر ماتریس Q را نیز بدین ترتیب آپدیت میکنیم که: با شروع از ستون j ام و به ترتیب تا آخر، ضرب خارجی بردار u در خودش، ضرب در ۲ را از آن ستون کم میکنیم. با اینکار با استفاده از بردار u ماتریس Q را متعامد میکنیم.

i. مثل عناصر ماتریس R ، عناصر خیلی کوچک ماتریس Q را نیز برابر صفر قرار میدهیم.

```
# Example usage with user input
m = int(input( Enter the number of rows:  ))
n = int(input( Enter the number of columns:  ))
A = np.zeros((m, n))
for i in range(m):
      row =  input(f Enter space-separated values for row {i+1}:  ).split()
      A[i, :] = [float(val) for val in row]
Q, R = qr_decomposition(A)
print( Q: )
print(Q)
print( R: )
print(R)
```

در این قسمت نیز ماتریس دلخواه کاربر را دریافت کرده، سپس تابع بالا را برای ماتریس وارد شده فراخوانی می کنیم و در نهایت به عنوان خروجی ماتریس های Q و R را چاپ میکنیم.

یک نمونه از خروجی این کد برای ماتریس مثال زیر:

$$\begin{bmatrix} 2 & -2 & 18 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \end{bmatrix}$$

```
Enter the number of rows: 3
Enter the number of columns: 3
Enter space-separated values for row 1: 2 -2 18
Enter space-separated values for row 2: 2 1 0
Enter space-separated values for row 3: 1 2 0
```

```
Q:
[[-0.66666667 0.66666667 -0.33333333]
 [-0.66666667 -0.33333333 0.66666667]
 [-0.33333333 -0.66666667 -0.66666667]]
 R:
[[ -3. 0. -12.]
 [ 0. -3. 12.]
 [ 0. 0. -6.]]
```