Eigenvalues using QR decomposition (user input example)

فاطمه صفری
ش.د ۹۷۲۸۰۶۳

لینک کد

در این کد با استفاده از پروسه‌ی هاوس هولدر ماتریسی نمونه را از کاربر گرفته، به یک ماتریس مثلثی R و یک ماتریس متعامد Q تجزیه کرده و ازین تجزیه برای بدست آوردن مقادیر ویژه استفاده می کنیم.

توضیح کلی هر قطعه کد:

```
import numpy as np
def qr_decomposition(A, threshold=1e-10):
    m, n = A.shape
    Q = np.eye(m)
    R = A.astype(float).copy()

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

توضیح جزئی تابع و حلقه‌ی بالا در فایل تجزیه QR به طور کامل بیان کردیم. اما به عنوان یک خلاصه، تابع بالا ماتریس A را با سطر و ستون m و n به عنوان ورودی

میگیرد، ماتریس های R و Q را به ترتیب برابر با ماتریس A با درایه های float ،و ماتریس همانی تعریف میکند.
سپس با استفاده از یک حلقه، ابتدا بردار x و e را ساخته که به ترتیب اول بردار x یک آرایه را از ماتریس اولیه R که زیر و روی قطر اصلی ماتریس R در (j, j) است.انتخاب میکند و بردار e را نیز ابتدا با همان اندازه‌ی بردار x ولی ابتدا با درایه های صفر قرار میدهیم.
حال بردار u را با را با ضرب کردن نرم اقلیدسی x و e و جمع کردن آن با بردار x بدست میاوریم. و آن را نرمال میکنیم.

حال ماتریس های اولیه Q و R را با استفاده از همین حلقه و جلو رفتن به ترتیب در سطر و ستون های آنها و آپدیت کردن آنها بدست می آوریم. و نیز اعداد بسیار کوچک را در درایه های این دو ماتریس، در نهایت به صفر رند میکنیم.

```
def qr_iteration_eigenvalues(A, num_iterations=100):
    a. m, n = A.shape
    eigenvalues = []
    a. for _ in range(num_iterations):
        Q, R = qr_decomposition(A)
        A = R @ Q
    b. for i in range(m):
        eigenvalues.append(A[i][i])
    return eigenvalues
```

حالا با این تابع میتوانیم با روش تکرار QR و استفاده از ماتریس های Q و R که بدست آوردیم، مقادیر ویژه را بدست آوریم.
این تابع دو ورودی دارد. ماتریس A و تعداد تکرار (که برابر ۱۰۰ قرار دادیم).
ابتدا همانند بالا m و n را برابر سطر و ستون ماتریس ورودی A میگذاریم.
یک لیست خالی برای مقادیر ویژه تعریف میکنیم.

a. حلقه‌ی اول:
این حلقه به تعداد شماره‌ی تکرار یعنی ۱۰۰ بار، مراحل زیر را انجام میدهد:
با استفاده از فراخوانی تابع بالا، Q و R ای را که با روش هوس هولدر بدست آوردیم، در هم ضرب ماتریسی میکند، تا ماتریس A را آپدیت کند. این مرحله به خاطر این است که با این کار میتوانیم ماتریس A را به ماتریسی با همان مقادیر ویژه همگرا کنیم.

b. حلقه‌ی دوم:
بعد از تمام شدن تعداد تکرار، وارد این حلقه میشویم که به تعداد سطر های ماتریس A تکرار میشود و با هر تکرار، عنصر قطر اصلی را به لیست مقادیر ویژه اضافه میکند.

- ```
  # 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]

eigenvalues = qr_iteration_eigenvalues(A)
print("Eigenvalues:")
print(eigenvalues)
```

در این قسمت نیز ماتریس دلخواه کاربر را دریافت میکنیم بدین ترتیب که ابتدا تعداد سطر و ستون را دریافت کرده، به همان ابعاد ماتریس A را با درایه های صفر تشکیل می دهیم. سپس در مرحله بعد سطر هایی که کاربر وارد کرده است را با تایپ float به ماتریس اضافه میکنیم و A را تشکیل می دهیم.

در نهایت مقادیر ویژه را خروجی میدهیم.

- خروجی این کد برای یک ماتریس نمونه:

[2 3]
[4 10]
```

```
Enter the number of rows: 2
Enter the number of columns: 2
Enter space-separated values for row 1: 2 3
Enter space-separated values for row 2: 4 10
Eigenvalues: [11.291502622165304, 0.7084973778685524]
Q: [[-0.4472136 0.89442719] [-0.89442719 -0.4472136 ]]
R: [[ -4.47213595 -10.2859127 ] [ 0. -1.78885438]]
Eigenvalues: [11.291502622165304, 0.7084973778685524]
```