Solving a linear equation with QR decomposition

فاطمه صفری
ش.د ۹۷۲۸۰۶۳

لینک کد

در این کد با استفاده از پروسه‌ی هاوس هولدر ماتریس نمونه را به یک ماتریس مثلثی "R" و یک ماتریس متعامد "Q" تجزیه کرده و ازین تجزیه برای حل مسئله‌ی داده شده استفاده میکنیم.

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_solve(A, b):
    a. Q, R = qr_decomposition(A)
    b. n = R.shape[1]
    c. x = np.linalg.inv(R[:n, :]) @ Q.T[:n, :] @ b[:n]
     return x
```

حالا با این تابع میتوانیم با روش کمترین مربعات و استفاده از ماتریس های Q و R ، پاسخ معادلات را پیدا کنیم. این تابع دو ورودی دارد. ماتریس A و بردار b.

a. ابتدا تابع قبلی را که با آن ماتریس A را تجزیه به ماتریس های Q و R کردیم فراخوانی می کنیم.

b. متغیر n برابر با تعداد ستون های ماتریس R قرار میدهیم.

c. حالا بردار x را بدین صورت می سازیم که: ابتدا ماتریس R را با [:n ,:] از سطرها قطعه میکنیم که n سطر و ستون اولش را برداریم.
حالا با استفاده از تابع np.linalg.inv() از کتابخانه NumPy میتوانیم معکوس آن قطعه از ماتریس R را که مرحله قبل داشتیم، بدست آوریم.
حالا ترانهاده ماتریس Q را که با Q.T داریم، قطعه به n سطر و ستون اولش میکنیم.
برای بردار b نیز به همین ترتیب باید n سطر اولش را برداریم.
حالا همه‌ی این ها را با ضرب ماتریسی(@) در هم ضرب میکنیم و برابر بردار جواب یا همان x قرار میدهیم و در انتهای کد آن را چاپ میکنیم.

- خروجی این کد برای ماتریس مثال سوال:

$$\begin{pmatrix} 2 & 3 & 1 \\ 5 & 1 & 2 \\ 2 & 3 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 8 \\ 11 \end{pmatrix}$$

- `Solution: [1. 1. 1.]`