THÔNG TIN SINH VIÊN

Mã số sinh viên: 23127447
Họ và tên: Nguyễn Thanh Owen
Lớp: 23CLC06
Email: ntowen23@clc.fitus.edu.vn

Thuật toán phân rã QR bằng phương pháp Gram–Schmidt - Hàm QR_Gram_Schmidt(A)

Mục tiêu của bài toán

Cho ma trận $A \in \mathbb{R}^{m \times n}$, thuật toán Gram–Schmidt phân rã A thành tích của hai ma trận:

$$A = Q \cdot R$$

Trong đó:

- Input: Một ma trận thực A có kích thước $m \times n$.
- Output: Hai ma trận thực Q và R thỏa mãn $A = Q \cdot R$:
- $Q \in \mathbb{R}^{m \times n}$: các cột là vector **trực chuẩn** (vuông góc và đơn vị).
- $R \in \mathbb{R}^{n \times n}$: **ma trận tam giác trên**.

Các bước thực hiện:

Gọi $a_0, a_1, \ldots, a_{n-1}$ là các cột của ma trận A. Với mỗi a_j, ta tính:

$$f_0 = a_0$$
$$e_0 = \frac{f_0}{\|f_0\|}$$

$$f_j = a_j - \sum_{k=0}^{j-1} (a_j \cdot e_k) e_k \quad \text{(loại bỏ thành phần không trực giao)}$$
$$e_j = \frac{f_j}{\|f_j\|} \quad \text{(chuẩn hóa)}$$

$$R_{k,j} = a_j \cdot e_k \quad \text{với } k < j$$
$$R_{j,j} = \|f_j\|$$

Sau khi có tất cả các vector e_j, ta xây dựng:

- Ma trận Q: các cột là các vector e_j
- Ma trận R: các hệ số chiếu và chuẩn $R_{k,j}$

In [102…

```python
# Tich vo huong giua hai vector u va v
def dot(u, v):
    dot_Result = 0
    for i in range(len(u)):
        dot_Result += u[i]*v[i]
    return dot_Result

# Nhan vector u voi moi hang so c
def scalar_multi(c, u):
    z = []
    for i in range(len(u)):
        z.append(c*u[i])
    return z

# Cong hai vector u va v
def sum_vector(u, v):
    z = []
    for i in range(len(u)):
        z.append(u[i] + v[i])
    return z
```

```
# Ham thuc hien phan ra QR theo phuong phap Gram-Schmidt
def QR_Gram_Schmidt(A):
    numRow = len(A) # So hang cua ma tran A
    numCol = len(A[0]) # So cot cua ma tran A

    Q = [] # Ma tran truc chuan ket qua
    R = [[0 for _ in range(numCol)] for _ in range(numCol)] # Khoi tao ma tran tam giac tren R toan 0
    e = [[0 for _ in range(numRow)] for _ in range(numCol)] # Luu cac vector truc chuan e_j
    f = [[0 for _ in range(numRow)] for _ in range(numCol)] # Luu cac vector chuan hoa f_j

    for j in range(numCol):
        a = []
        # Lay cot thu j cua ma tran A
        for i in range(numRow):
            a.append(A[i][j])

        # Voi cot dau tien f[0] = a[0]
        if j == 0:
            f[0] = a[:]
            e[0] = scalar_multi(1/(dot(f[0], f[0])**0.5), f[0])

        else:
            f[j] = a[:]
            # Loai bo cac thanh phan khong truc giao
            for k in range(j):
                R[k][j] = dot(a, e[k]) # He so chieu cua vector a[j] len vector don vi e[k]
                factor = scalar_multi(dot(a, e[k]) * (-1), e[k])
                f[j] = sum_vector(f[j], factor)

            # Chuan hoa f[j] de thanh e[j]
            e[j] = scalar_multi(1/(dot(f[j], f[j])**0.5), f[j])

        # Gan gia tri duong cheo cua R (chuan cua f[j])
        R[j][j] = dot(f[j], f[j])**(1/2)

    # Tao ma tran Q voi cac cot la cac vector e[j] tim duoc
    for i in range(numRow):
        row = []
        for j in range(numCol):
            row.append(e[j][i])
        Q.append(row)
```

```
    return Q, R


# Goi ham phan ra QR voi ma tran
A = [
    [1, 1, 2],
    [2, -1, 1],
    [-2, 4, 1]
]

Q, R = QR_Gram_Schmidt(A)

print("Ma trận Q:")
for row in Q:
    print(["{:.4f}".format(x) for x in row])

print("\nMa trận R:")
for row in R:
    print(["{:.4f}".format(x) for x in row])
```

```
Ma trận Q:
['0.3333', '0.6667', '0.6667']
['0.6667', '0.3333', '-0.6667']
['-0.6667', '0.6667', '-0.3333']

Ma trận R:
['3.0000', '-3.0000', '0.6667']
['0.0000', '3.0000', '2.3333']
['0.0000', '0.0000', '0.3333']
```

Giải thích các hàm trong mã phân rã QR (Gram–Schmidt)

`dot(u, v)`

Tính **tích vô hướng** giữa hai vector u và v:

$$\text{dot}(u, v) = \sum u_i \cdot v_i$$

`scalar_multi(c, u)`

Nhân vector u với **hằng số vô hướng** c:

$$c \cdot u = [c \cdot u_1,\ c \cdot u_2,\ \ldots]$$

`sum_vector(u, v)`

Tính **tổng hai vector** cùng chiều dài:

$$u + v = [u_1 + v_1,\ u_2 + v_2,\ \ldots]$$

`QR_Gram_Schmidt(A)`

Phân rã ma trận A thành:

$$A = Q \cdot R$$

Trong đó:

- Q: ma trận trực chuẩn (các cột là vector đơn vị, vuông góc với nhau)
- R: ma trận tam giác trên chứa hệ số chiếu và độ dài

Các bước chính:

1. Với mỗi cột a_j của A:
2. Loại bỏ thành phần không trực giao:

$$f_j = a_j - \sum_{k=0}^{j-1} (a_j \cdot e_k) \cdot e_k$$

3. Chuẩn hóa thành:

$$e_j = \frac{f_j}{\|f_j\|}$$

4. Cập nhật ma trận R:

$$R_{k,j} = a_j \cdot e_k \quad (k < j), \quad R_{j,j} = \|f_j\|$$

Phân rã QR sử dụng thư viện `sympy`

Đoạn mã sử dụng thư viện `sympy` để phân rã ma trận A thành $Q \cdot R$ bằng phương pháp **Gram–Schmidt trực chuẩn hóa**.

Các bước thực hiện:

- **Bước 1:** Khởi tạo ma trận A.
- **Bước 2:** Lấy các vector cột từ A bằng `A.columnspace()`.
- **Bước 3:** Áp dụng `GramSchmidt(..., orthonormal=True)` để tạo các vector trực chuẩn $e_1, e_2, \ldots$.
- **Bước 4:** Dùng `Matrix.hstack(...)` để ghép các vector thành ma trận Q.
- **Bước 5:** Tính R bằng công thức $R = Q^T \cdot A$.

Kết quả:

- Q: ma trận trực chuẩn (vuông góc, đơn vị).
- R: ma trận tam giác trên.
- Đảm bảo: $A = Q \cdot R$.

In [103…

```
from sympy import Matrix, GramSchmidt, pprint, init_printing

init_printing()

# Khoi tao ma tran A
A = Matrix([
    [1, 1, 1],
    [2, -2, 2],
    [1, 1, -1]
])

def qr_gram_schmidt_sympy(A):
    # Lay danh sach cac vector cot cua A
    cols = A.columnspace()

    # Ap dung Gram-Schmidt de truc chuan hoa cac vector
```

```
    orthonormal_basis = GramSchmidt(cols, orthonormal=True)

    # Tao ma tran Q tu cac vector truc chuan
    Q = Matrix.hstack(*orthonormal_basis)

    # Tinh ma tran R
    R = Q.T * A
    return Q, R


Q, R = qr_gram_schmidt_sympy(A)
# In kết quả
print("Ma trận Q:")
pprint(Q)

print("\nMa trận R:")
pprint(R)
```

```
Ma trận Q:
⎡√6   √3    √2 ⎤
⎢──   ──    ── ⎥
⎢6    3     2  ⎥
⎢                ⎥
⎢√6  -√3        ⎥
⎢──  ────   0   ⎥
⎢3    3         ⎥
⎢                ⎥
⎢√6   √3   -√2  ⎥
⎢──   ──   ────⎥
⎣6    3     2  ⎦

Ma trận R:
⎡    -√6    2⋅√6 ⎤
⎢√6  ────   ──── ⎥
⎢     3      3   ⎥
⎢                ⎥
⎢    4⋅√3  -2⋅√3 ⎥
⎢0   ────  ─────⎥
⎢     3      3   ⎥
⎢                ⎥
⎣0    0      √2  ⎦
```

So sánh hai cách phân rã QR: Python thủ công và `sympy.GramSchmidt`

Tiêu chí	Tự cài đặt bằng Python (thuần túy)	Sử dụng `sympy.GramSchmidt`
Cách thực hiện	Từng bước viết tay: 1. Tích vô hướng 2. Chuẩn hóa vector 3. Trừ thành phần không trực giao 4. Tạo Q và R	Gọi hàm `GramSchmidt(cols, orthonormal=True)` để lấy trực chuẩn Tạo Q từ các vector, R = $Q^T A$
Thư viện sử dụng	Không cần thư viện ngoài	Cần cài `sympy`
Cách lưu trữ dữ liệu	Dùng `list` 2 chiều (float)	Dùng `sympy.Matrix` (phân số, căn số)
Hiển thị kết quả	Làm tròn đến 4 chữ số (ví dụ `0.7071`)	Hiển thị phân số hoặc căn bậc hai
Độ chính xác	Có thể sai số làm tròn	Chính xác tuyệt đối
Hiểu thuật toán	Tốt cho học lý thuyết, thấy rõ từng bước	Ẩn chi tiết thuật toán
Tốc độ chạy	Nhanh hơn với ma trận nhỏ	Chậm hơn do tính biểu thức
Kiểm tra $A = QR$	Có thể lệch nhỏ do float	Khớp tuyệt đối (do không làm tròn)

Ứng dụng của QR Decomposition

QR Decomposition là cách tách một ma trận thành 2 ma trận đặc biệt:

- **Q**: ma trận có các cột vuông góc và chuẩn hóa (vector đơn vị).
- **R**: ma trận tam giác trên.

Việc phân rã này giúp giải quyết nhiều bài toán trong khoa học dữ liệu, học máy, và xử lý tín hiệu.

1. Giải bài toán hồi quy tuyến tính (Linear Regression)

- Dùng QR để tìm đường thẳng dự đoán tốt nhất giữa dữ liệu đầu vào và đầu ra.
- So với cách truyền thống (giải phương trình), QR **ổn định và nhanh hơn**.

Ví dụ: Dự đoán điểm thi từ số giờ học.

2. Lọc ra các đặc trưng quan trọng (Feature Selection)

- Khi dữ liệu có nhiều cột và một số cột gần giống nhau (gọi là cộng tuyến), mô hình dễ bị sai.
- QR giúp phát hiện và loại bỏ các cột dư thừa.

Ví dụ: "Tuổi" và "năm sinh" mang thông tin trùng lặp.

3. Giảm chiều dữ liệu (PCA – Principal Component Analysis)

- Dữ liệu có quá nhiều cột -> khó xử lý.
- PCA giúp giảm số cột, nhưng vẫn giữ lại thông tin chính.
- QR được dùng trong tính toán nội bộ của PCA.

Ví dụ: Ảnh có 1000 pixel -> chọn ra 50 pixel quan trọng nhất.

4. Tránh quá khớp trong hồi quy (Regularization)

- QR decomposition hỗ trợ các kỹ thuật như **Ridge Regression** để tránh mô hình học quá sát dữ liệu huấn luyện (overfitting).

Ví dụ: Dự đoán kết quả học tập mà không bị ảnh hưởng bởi chi tiết nhỏ không quan trọng.

5. Tìm trị riêng (Eigenvalues)

- Trong các thuật toán như **phân cụm (clustering)** hay **giảm chiều (dimensionality reduction)**, cần tính trị riêng.
- QR decomposition có thể dùng để tìm trị riêng một cách hiệu quả.

Ví dụ: Nhóm người dùng có hành vi tương tự.

6. Lọc nhiễu trong xử lý tín hiệu

- Dùng QR để phân tách phần chính và phần nhiễu của tín hiệu (âm thanh, hình ảnh).
- Giúp làm rõ thông tin và giảm nhiễu.

Ví dụ: Làm rõ giọng nói trong môi trường ồn hoặc tăng chất lượng ảnh.