Bartosz Hanc, Paweł Surdyka

Raport z zadania 6.

Rozkład QR (ang. *QR decomposition* lub *QR factorization*) jest jedną z podstawowych technik numerycznych stosowanych w algebrze liniowej. Polega on na przedstawieniu macierzy prostokątnej $A \in \mathbb{R}^{m\times n}$, gdzie $m \geq n$ jako iloczynu dwóch macierzy

$$A = QR$$

gdzie $Q \in \mathbb{R}^{m\times m}$ jest macierzą ortogonalną, czyli spełniającą warunek $Q^{-1} = Q^{\intercal}$, natomiast $R \in \mathbb{R}^{m\times n}$ jest macierzą trójkątną górną, czyli taką, w której wszystkie elementy poniżej głównej przekątnej są równe zeru. Rozkład QR znajduje szerokie zastosowanie w numerycznym rozwiązywaniu układów równań liniowych, szczególnie wtedy, gdy macierz A nie jest kwadratowa lub gdy układ jest nadokreślony (czyli ma więcej równań niż niewiadomych). Jest także powszechnie stosowany w algorytmach znajdowania wartości własnych, w metodzie najmniejszych kwadratów (ang. *least squares*), a także w analizie numerycznej i uczeniu maszynowym. Istnieje kilka metod obliczania rozkładu QR, w tym: klasyczna i zmodyfikowana metoda Grama-Schmidta, transformacje Householdera, czy obroty Givensa. W niniejszym projekcie skupiamy się na ostatnim podejściu, czyli rozkładzie QR przy użyciu obrotów Givensa. Rotacje te pozwalają w sposób numerycznie stabilny eliminować pojedyncze elementy macierzy za pomocą prostych obrotów w przestrzeni dwuwymiarowej. Pojedyncza rotacja Givensa przyjmuje postać macierzy $G(i, k, \theta)$ działającej na dwa wiersze i i k, w celu wyzerowania wybranego elementu:

$$G(i,k,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

gdzie $c = \cos\theta$, $s = \sin\theta$ i pozostałe elementy tworzą macierz jednostkową. Dzięki lokalności działania (tylko na dwóch wierszach) obroty Givensa są szczególnie efektywne dla rzadkich macierzy oraz dobrze nadają się do równoległej implementacji. W dalszej części raportu przedstawiono implementację tej metody w języku Python z użyciem biblioteki NumPy oraz porównano ją z wbudowaną funkcją `np.linalg.qr`.

Poniższy algorytm przedstawia sposób wyznaczania rozkładu QR macierzy A przy użyciu rotacji Givensa. W każdej iteracji rotacja Givensa eliminuje jeden element poniżej głównej przekątnej w macierzy R, przekształcając ją do postaci górnotrójkątnej. Macierz Q jest tworzona jako iloczyn kolejnych obrotów odwrotnych (transponowanych macierzy Givensa). Dzięki lokalnemu charakterowi działania rotacji Givensa (na dwóch wierszach) metoda ta jest numerycznie stabilna i nadaje się do zastosowań w macierzach rzadkich oraz systemach równoległych.

Algorithm 1: QR Decomposition via Givens Rotations

Input: $A \in \mathbb{R}^{m \times n}$
Output: $Q \in \mathbb{R}^{m \times m}$, $R \in \mathbb{R}^{m \times n}$
$Q \leftarrow I_m$;
$R \leftarrow A$;
for $j \leftarrow 0$ **to** $n-1$ **do**
  for $i \leftarrow m-1$ **downto** $j+1$ **do**
    $k \leftarrow i-1$;
    $(a, b) \leftarrow (R[k,j], R[i,j])$
    ▷ Compute cosine and sine of Givens rotation
    $(c, s) \leftarrow \left(\frac{a}{\sqrt{a^2+b^2}}, \frac{b}{\sqrt{a^2+b^2}} \right)$
    ▷ Apply Givens rotation to rows k and i of R
    $(R[k,:], R[i,:]) \leftarrow (cR[k,:] + sR[i,:], -sR[k,:] + cR[i,:])$;
    ▷ Apply inverse Givens rotation to columns k and i of Q
    $(Q[:,k], Q[:,i]) \leftarrow (cQ[:,k] + sQ[:,i], -sQ[:,k] + cQ[:,i])$;
  end
end
return Q, R;

Poniżej znajduje się implementacja powyższego algorytmu w języku Python z wykorzystaniem biblioteki NumPy. W finalnej wersji implementacji algorytmu QR z obrotami Givensa zastosowano podejście uproszczone, w którym nie konstruuje się jawnie pełnej macierzy obrotu Givensa. Zamiast tego, przekształcenia są wykonywane bezpośrednio na odpowiednich wierszach macierzy i kolumnach macierzy R i Q. W każdej iteracji algorytmu obliczane są współczynniki c i s. Zamiast stosować wprost mnożenie przez macierz G, algorytm aktualizuje odpowiednie wiersze i kolumny w macierz Q i R za pomocą liniowej kombinacji tych wierszy, odpowiadającej rotacji w przestrzeni dwuwymiarowej. Zaletą takiego podejścia jest duża oszczędność pamięci (brak potrzeby tworzenia dużych macierzy pośrednich) oraz uproszczenie kodu. Taka forma implementacji zachowuje dokładność numeryczną i pozwala na efektywne wykonywanie obliczeń dla dużych macierzy.

Aby zweryfikować poprawność implementacji rozkładu QR metodą obrotów Givensa, przeprowadzono testy porównujące wynikowe macierze Q i R względem oryginalnej macierzy wejściowej A. Dla kilku losowych macierzy A o różnych wymiarach sprawdzono trzy warunki: (1) czy iloczyn QR jest równy A z zadaną dokładnością, (2) czy macierz Q jest ortogonalna tzn. $Q^\intercal Q \approx QQ^\intercal \approx I$ oraz (3) czy macierz R jest macierzą trójkątną górną. We wszystkich testach błąd był

```python
def qrgivens(A: NDArray) -> tuple[NDArray, NDArray]:
    m, n = A.shape
    Q = np.eye(m)
    R = A.copy().astype(np.float64)

    for j in range(n):
        for i in range(m - 1, j, -1):
            k = i - 1
            a, b = R[k, j], R[i, j]

            r = np.sqrt(a**2 + b**2)
            c, s = a / r, b / r

            R[k, :], R[i, :] = c*R[k, :] + s*R[i, :], -s*R[k, :] + c*R[i, :]
            Q[:, k], Q[:, i] = c*Q[:, k] + s*Q[:, i], -s*Q[:, k] + c*Q[:, i]

    return Q, R
```

rzędu 10^{-14}, co potwierdza poprawność algorytmu w zakresie precyzji numerycznej. Przykładowo dla macierzy

$$A = \begin{bmatrix} 12 & -51 & 4 & 1 \\ 6 & 167 & -68 & 2 \\ -4 & 24 & -41 & 3 \\ -1 & 1 & 0 & 5 \end{bmatrix}$$

otrzymujemy następujące rozkłady QR:

- wynik implementacji własnej

$$Q = \begin{bmatrix} 0.855 & -0.3934 & -0.3314 & -0.0667 \\ 0.4275 & 0.9032 & 0.0343 & -0.0177 \\ -0.285 & 0.1711 & -0.9429 & 0.0228 \\ -0.0712 & 0.0142 & 0. & -0.9974 \end{bmatrix}$$

$$R = \begin{bmatrix} 14.0357 & 20.8754 & -13.9644 & 0.4987 \\ 0. & 175.0178 & -70.0071 & 1.9974 \\ 0. & 0. & 35. & -3.0914 \\ 0. & 0. & 0. & -5.0204 \end{bmatrix}$$

- wynik funkcji `np.linalg.qr`

$$Q = \begin{bmatrix} -0.855 & 0.3934 & -0.3314 & 0.0667 \\ -0.4275 & -0.9032 & 0.0343 & 0.0177 \\ 0.285 & -0.1711 & -0.9429 & -0.0228 \\ 0.0712 & -0.0142 & 0. & 0.9974 \end{bmatrix}$$

$$R = \begin{bmatrix} -14.0357 & -20.8754 & 13.9644 & -0.4987 \\ 0. & -175.0178 & 70.0071 & -1.9974 \\ 0. & 0. & 35. & -3.0914 \\ 0. & 0. & 0. & 5.0204 \end{bmatrix}$$

Jak widzimy wartości są równe co do znaku. Przy porównywaniu wyników rozkładu QR zaimplementowanego algorytmu z funkcją biblioteczną mogą pojawić się różnice w znakach kolumn macierzy Q oraz wierszy macierzy R. Wynika to z faktu, że rozkład QR jest niejednoznaczny względem mnożenia przez macierz diagonalną o elementach ± 1.

W celu porównania wydajności zaimplementowanego algorytmu QR z metodą obrotów Givensa oraz bibliotecznej funkcji `np.linalg.qr` wykonano serię pomiarów czasu działania na macierzach kwadratowych o różnych rozmiarach. Na zamieszczonym poniżej wykresie przedstawiono zależność czasu wykonania od rozmiaru macierzy. Widać wyraźnie, że dla małych wymiarów obie implementacje radzą sobie podobnie, jednak z rosnącym rozmiarem macierzy przewaga funkcji bibliotecznej staje się znacząca.