Metody wyznaczania wartości własnych

inż. Jakub Korczakowski
Wydział Elektryczny
Politechnika Warszawska

inż. Piotr Rosa
Wydział Elektryczny
Politechnika Warszawska

***Streszczenie*—W ramach prac nad projektem zaimplementowaliśmy metody służące do wyznaczania wartości własnych macierzy oraz opisaliśmy ich podstawy teoretyczne. Przeprowadziliśmy badania mające na celu porównanie zbieżności poszczególnych metod, a także zobrazowanie ich złożoności zarówno czasowych jak i pamięciowych. Zostały także opisane i zaprezentowane przykłady zastosowań tych metod w uczeniu maszynowym.**

1. Wstęp

Tematem projektu, który wybraliśmy do zrealizowania w ramach przedmiotu Algorytmy w Inżynierii danych jest Temat 2 - Metody wyznaczania wartości własnych. Celem tego projektu jest zaimplementowanie i przeprowadzenie analizy porównawczej trzech algorytmów wyznaczania wartości i wektorów własnych macierzy z pokazaniem przykładów ich zastosowania w analizie danych. Wybrane algorytmy to: metoda potęgowa, QR oraz metoda Jacobiego. Algorytmy zostały zaimplementowane w języku Julia. Wybór padł na te algorytmy, ze względu na to, że są one stosowane do trzech różnych rodzai macierzy. Metoda potęgowa może być stosowana dla macierzy o wartościach rzeczywistych, tylko do wyznaczania niezespolonych wartości własnych, metoda Jacobiego stosowana jest wyłącznie do macierzy symetrycznych, natomiast metoda QR do macierzy również niesymetrycznych, a także o zespolonych wartościach własnych.

Podczas pracy oprócz materiałów z wykładu używaliśmy dwóch źródeł: [1] oraz [2].

Pakietem użytym przy implementacji algorytmów był pakiet *LinearAlgebra 1.5.4*, natomiast do testowania wydajności algorytmów i wizualizacji wyników użyte zostały pakiety *BenchmarkTools v0.7.0* oraz *PyPlot v2.9.0*

2. Opis algorytmów

2.1. Metoda Potęgowa

Metodę potęgową wykorzystuje się do wyznaczenia promienia spektralnego macierzy, czyli największej co do modułu wartości własnej. Metoda ta polega na wielokrotnym wykonaniu szeregu mnożeń wybranego wektora startowego x przez macierz wejściową A. Wykonanie takich mnożeń jest równoznaczne z pomnożeniem wektora przez macierz A podniesioną do potęgi równej liczbie iteracji. W wyniku tych operacji, wektor x odpowiada wektorowi własnemu największej wartości własnej. Posiadając wektor własny oraz macierz wejściową A, można obliczyć wartość własną odpowiadającą wektorowi. W celu zachowania stabilności numerycznej algorytmu, po każdym wykonaniu mnożenia wektora przez macierz, wynik jest dodatkowo normalizowany.

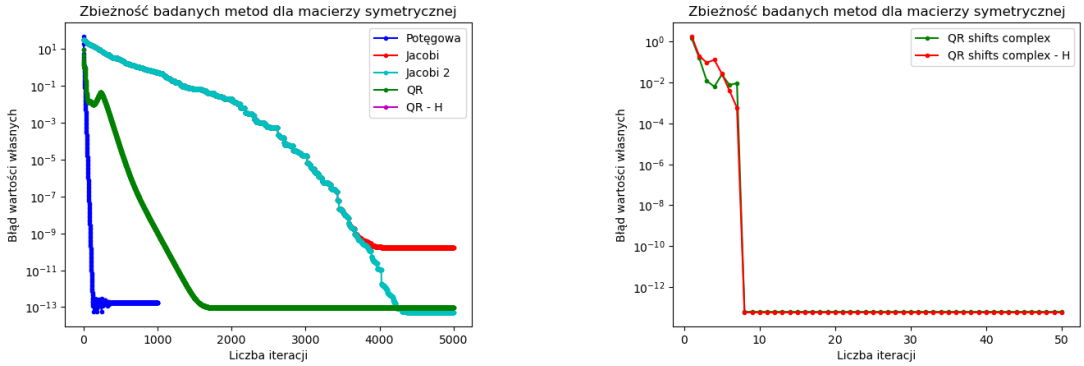
W ramach obliczania wartości własnych metodą potęgową, w każdej iteracji alokowany jest wektor własny. Z tego względu złożoność pamięciowa metody potęgowej wynosi $O(n)$. Złożoność obliczeniowa równa jest złożoności mnożenia wektora przez macierz, a więc jest to złożoność $O(n^2)$.

Zbieżność metody potęgowej jest liniowa. W zależności od tego, jak bardzo największa co do modułu wartość własna λ_1 różni się od kolejnej wartości własnej λ_2, algorytm wynik może zbiegać się szybciej lub wolniej. Zależność tę można opisać za pomocą wzoru $error = (\lambda_2/\lambda_1)^n$. Jeśli największe wartości własne są sprzężonymi liczbami zespolonymi, metoda jest rozbieżna i nie jest w stanie wyznaczyć tych wartości.

2.2. QR

Metoda QR [3] do wyznaczania wartości i wektorów własnych macierzy wykorzystuje rozkład QR. Rozkład ten polega na rozłożeniu oryginalnej macierzy na iloczyn macierzy Q i R, gdzie macierz Q jest macierzą ortogonalną, a macierz R górnotrójkątną. Ortogonalność macierzy Q zapewnia stabilność numeryczną metody. Następnie obliczany jest iloczyn macierzy R*Q. Powstała w wyniku tej operacji macierz jest podobna do macierzy A, a więc ma takie same wartości własne. Następnie macierz wynikowa rozkładana jest na macierze Q i R i cały proces jest powtarzany. Po wykonaniu dostatecznej ilości takich cykli, na diagonali macierzy wynikowej iloczynu R*Q znajdują się wartości własne.

W przypadku algorytmu QR do wyznaczania wartości własnych macierzy, w każdej iteracji tworzona jest pełna macierz o rozmiarach odpowiadających macierzy wejściowej. Złożoność pamięciowa tej metody to zatem $O(n^2)$. W każdej iteracji działania metoda wykonuje mnożenie macie-



Rysunek 1. Porównanie zbieżności metod dla macierzy symetrycznej z elementami rzeczywistymi. Wektor testowy inicjalizowany był jako wektor jedynek.

Tablica 1. PORÓWNANIE WYNIKÓW TESTÓW WYKONANYCH ZA POMOCĄ METODY @BENCHMARK.

Metoda	Mediana czasu działania	Estymowana ilość użytej pamięci	Liczba alokacji
Potęgowa	2.395 ms	969.84 KiB	2004
Jacobi	38.242 ms	57.54 MiB	6002
Jacobi 2	3.379 ms	3.80 MiB	8004
QR	568.027 ms	2.37 GiB	5006002
QR Hessenberg	597.097 ms	2.37 GiB	5006905
QR z przesunięciami	33.696 ms	106.25 MiB	246632
QR z przesunięciami, Hessenberg	34.785 ms	108.10 MiB	244370

rzy o rozmiarach proporcjonalnych do macierzy wejściowej, stąd złożoność obliczeniowa wynosi $O(n^3)$.

W wyniku metody otrzymywana jest macierz górnotrójkątna, gdzie na diagonali znajdują się wartości własne macierzy. Jeśli wartości te zostaną posortowane malejąco co do modułu, to elementy pod diagonalą dążą do zera zgodnie ze wzorem $|a_{ij}^{(k)}| = O(|\lambda_i/\lambda_j|^k)$, gdzie k jest numerem iteracji, λ oznacza wartość własną oraz $i > j$

2.3. Metoda Jacobiego

Metoda Jacobiego polega na wykonaniu ciągu transformacji ortogonalnych, na macierzy wejściowej A, w celu doprowadzenia jej do postaci diagonalnej. Przekształcenia ortogonalne macierzy nie zmieniają jej wartości własnych, a więc wynikowa macierz ma takie same wartości własne co macierz wejściowa. W związku z tym, że wynikowa macierz jest macierzą diagonalną, wartości znajdujące się na diagonali tej macierzy są jednocześnie poszukiwanymi wartościami własnymi macierzy A.

Metoda Jacobiego, podobnie jak metoda QR, w każdej iteracji tworzy macierz o takich samych wymiarach jak macierz wejściowa, a więc posiada złożoność pamięciową rzędu $O(n^2)$. Również jak w przypadku metody QR, w każdej iteracji wykonywane jest mnożenie macierzy, co oznacza, że metoda ta również ma złożoność obliczeniową równą $O(n^3)$.

Metoda Jacobiego jest zbieżna kwadratowo.

3. Implementacja i testy

Testy przeprowadzane były na maszynie o procesorze Intel(R) Xeon(R) CPU E31270 @ 3.40GHz, posiadającym 8 rdzeni oraz 16 wątków. Maszyna posiada 16G pamięci RAM. Dysk twardy używany podczas eksperymentów to WD7502AAEX-00Y9A0 750GB. Używany był system operacyjny Linux Ubuntu 20.04 focal. Algorytmy zostały przetestowane w programie Jupyter Notebook w przeglądarce Google Chrome. Notatnik zawierający kod oraz dodatkowe wykresy załączyliśmy wraz ze sprawozdaniem. Macierze testowe przygotowywane były jako macierze losowe. Błąd mierzony był jako wartość bezwzględna różnicy wektora wartości własnych oraz zakładanych wartości własnych (wartości własnej w przypadku metody potęgowej).

4. Wyniki badań

4.1. Zbieżność metod

W badaniu zbieżności metod zostały uwzględnione: metoda potęgowa, dwie implementacje metody Jacobiego oraz klasyczne implementacje metody QR bez oraz z redukcją macierzy wejściowej do postaci macierzy Hessenberga. Zbieżność badana była na macierzy symetrycznej o wymiarach 50 na 50 elementów.

Wyniki w postaci wykresu zostały przedstawione na obrazie 1a.

(a) Potęgowa (b) Jacobi (c) Jacobi 2 (d) QR (e) QR H (f) QR przes. (g) QR przes. H

Rysunek 2. Porównanie złożoności czasowej badanych algorytmów.

(a) Potęgowa (b) Jacobi (c) Jacobi 2 (d) QR (e) QR H (f) QR przes. (g) QR przes. H

Rysunek 3. Porównanie złożoności pamięciowej badanych algorytmów.

W przypadku metody potęgowej błąd dotyczy największej wartości własnej, natomiast w przypadku pozostałych metod jest to średnia błędu dla wszystkich wartości. Na podstawie wykresu można stwierdzić, że najszybciej zbieżna jest metoda potęgowa, natomiast najwolniej zbieżne są metody Jacobiego. Dodatkowo nie można zauważyć przebiegu dla metody QR - H. Wynika to z tego, że przebieg ten jest identyczny z tym dla metody QR, a więc zastosowanie redukcji macierzy do postaci Hessenberga nie poprawiło zbieżności.

Na wykresie 1b została natomiast przedstawiona zbieżność metody QR z przesunięciami oraz QR z przesunięciami oraz transformacją do postaci Hessenberga. Dodatkowo ta implementacja pozwala na obliczanie także zespolonych wartości własnych. Również w tym przypadku do pomiaru zbieżności użyta została symetryczna macierz 50 na 50 elementów.

Z wykresu można odczytać, że metoda ta zbiegła się do błędu rzędu poniżej 10^{-12} po zaledwie 8 iteracjach. Porównując to z wynikami na poprzednim wykresie 1a jest to zaskakująco niska liczba. Podobnie jak poprzednio nie widać jednak znaczących różnic pomiędzy wariantami z wykorzystaniem transformacji do postaci Hessenberga i bez niej.

4.2. Złożoność obliczeniowa i pamięciowa

Pierwsze badanie przeprowadziliśmy dla losowej symetrycznej macierzy o rozmiarze 50 na 50. Badaliśmy medianę czasu działania, estymowaną ilość zużytej pamięci oraz liczbę alokacji dla 1000 iteracji. Wyniki badania przedstawione zostały w tabeli 1. Metoda potęgowa potrzebowała zdecydowanie najmniej czasu i pamięci, co jest raczej oczywiste, ponieważ zwraca ona jedynie największą wartość własną. Kolejnymi badanymi metodami były dwie implementacje metody Jacobiego. Pierwsza z nich była naiwną implementacją, podczas gdy druga była zoptymalizowana, poprzez m. in. uniknięcie części mnożenia macierzy. Odzwierciedlone jest to w wynikach badań, zoptymalizowana metoda potrzebuje 10 razy mniej czasu oraz około 15 razy mniej pamięci. Następną grupą badanych metod są metody QR. Pierwsza metoda to naiwna implementacja metody QR, wraz z wariantem wykorzystującym redukcję Hessenberga. Wykorzystywała ona bardzo dużą ilość pamięci oraz wykonywała się najdłużej ze wszystkich metod. Znaczącą poprawę szybkości działania oraz zmniejszenie zajmowanej pamięci można zauważyć dla metod QR wykorzystujących przesunięcia.

5. Zastosowanie w uczeniu maszynowym

Jednym z zadań postawionych na początku projektu było wykorzystanie zaimplementowanych przez nas metod wyznaczania wartości i wektorów własnych w zadaniach uczenia maszynowego. Zdecydowaliśmy się zaimplementować dekompozycję SVD oraz analizę składowych głównych (PCA) w dwóch wariantach. Opisane w tej sekcji metody znajdują się w pliku `Zastosowania.ipynb`.

5.1. SVD

Dekompozycja SVD przedstawiona jest wzorem:

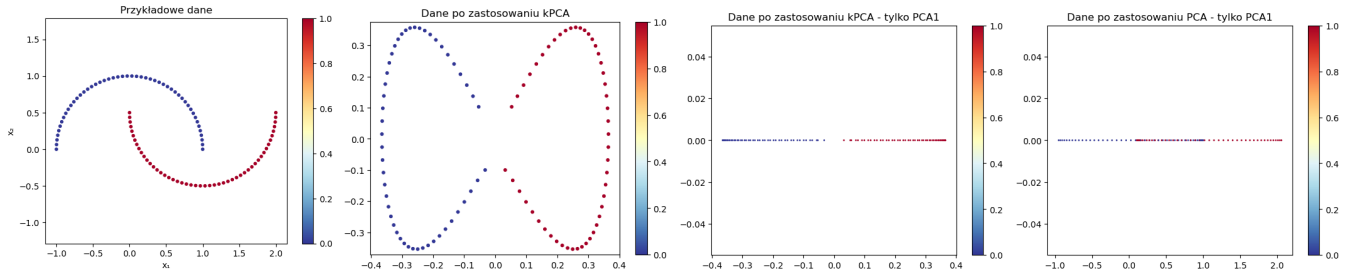
$$A = U\Sigma V^T = \sum_{j=1}^{r} \sigma_j u_j v_j^T \tag{1}$$

gdzie U i V to macierze ortogonalne, a Σ to macierz diagonalna zawierająca wartości osobliwe macierzy A.

W przypadku naszego projektu zrealizowaliśmy dekompozycję SVD używając zaimplementowanego przez nas algorytmu dekompozycji QR opartego na algorytmie Grama-Schmidta.

5.2. PCA

Kolejnym zaimplementowanym przez nas algorytmem jest metoda PCA [4]. Dla danego zbioru danych metoda



(a) Przykładowe dane (b) Dane po kPCA (c) Dane po kPCA - PCA 1 (d) Dane po PCA - PCA 1

Rysunek 4. Przedstawienie działania zaimplementowanej metody kPCA. Na obrazach 4c oraz 4d porównane są wyniki redukcji wymiaru do jednego dla danych przedstawionych na obrazie 4a.

PCA poszukuje osi, które maksymalizują wariancję analizowanych danych. Kolejne osie (składowe główne) objaśniają coraz mniej wariancji. Pozwala to na wykorzystanie metody PCA jako narzędzia do redukcji wymiarów danych, poprzez użycie danych opisywanych przez wybraną liczbę składowych głównych.

W przypadku naszego projektu metoda PCA została zaimplementowana za pomocą metody Jacobiego. Algorytm używany jest do wyznaczenia wektorów własnych macierzy kowariancji zbudowanej na podstawie danych wejściowych.

5.3. Kernel PCA

Ostatnim zaimplementowanym przez nas algorytmem była modyfikacja metody PCA [5] pozwalająca na wykorzystanie jej w problemach nieliniowych. Metoda ta może zostać użyta np. w zadaniu rozpoznawania twarzy [6].

Metoda Kernel PCA wykorzystuje projekcję danych na przestrzeń, w której te dane będą liniowo separowalne. Jądro (ang. kernel) oznaczać będzie funkcję, która pozwala dokonać projekcji danych x na przestrzeń ϕ.

$$\kappa(x_i, x_j) = \phi(x_i)\phi(x_j)^T \tag{2}$$

Taka projekcja danych pozwala na wyznaczenie wektorów własnych z macierzy kowariancji danych, które zostały zrzutowane na daną przestrzeń ϕ.

W przypadku naszego projektu zaimplementowaliśmy metodę Kernel PCA wykorzystujące jądro Gaussa, opisane jako:

$$\kappa(x_i, x_j) = \exp\left(-\gamma||x_i - x_j||_2^2\right) \tag{3}$$

Do wyznaczenia wektorów własnych wykorzystaliśmy metodę Jacobiego.

6. Wnioski

W tym raporcie opisaliśmy naszą pracę związaną z zadanym projektem. Zaimplementowaliśmy różne wersje trzech algorytmów do wyznaczania wartości własnych, odpowiednio: metody potęgowej, algorytmu Jacobiego oraz algorytmu QR. Przeprowadziliśmy testy zbieżności oraz złożoności obliczeniowej. Wyniki, które uzyskaliśmy potwierdziły założenia teoretyczne dotyczące tych algorytmów. Zbadaliśmy również wydajność za pomocą narzędzi dostępnych w języku Julia.

W dalszych badaniach można wziąć pod uwagę zrównoleglenie metod w celu poprawy szybkości ich działania. Zrównolegleniu mogą zostać poddane metoda potęgowa, która wykorzystuje mnożenie wektora i macierzy, oraz metoda Jacobiego, która w każdej iteracji obraca tylko określone kolumny i wiersze. Metoda QR nie może natomiast zostać poddana zrównolegleniu.

Literatura

[1] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997. 1

[2] S. Johnson, "18.335J Introduction to Numerical Methods," [online], Spring 2019, [dostęp: 2021-05-09]. 1

[3] J. G. F. Francis, "The QR Transformation A Unitary Analogue to the LR Transformation—Part 1," *The Computer Journal*, vol. 4, no. 3, pp. 265–271, 01 1961. [Online]. Available: https://doi.org/10.1093/comjnl/4.3.265 1

[4] K. P. F.R.S., "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. [Online]. Available: https://doi.org/10.1080/14786440109462720 3

[5] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998. 4

[6] Q. Wang, "Kernel principal component analysis and its applications in face recognition and active shape models," 2014. 4