Wyznaczanie wartości własnych macierzy z wykorzystaniem oprogramowania Julia

Autorzy: Jarosław Królik (284363), Mateusz Derszniak (284293)

Streszczenie- Raport rozpoczyna się wprowadzeniem w temat wartości własnych macierzy. Zostaje przybliżona definicja wartości własnych macierzy oraz zostają zaprezentowane najczęściej spotykane przykłady ich zastosowań. Dalsza część raportu zawiera przegląd aktualnej literatury, pod kątem algorytmów wyliczania wartości własnych. Spośród znalezionych metod zostają wybrane 3 na których będą przeprowadzane badania. Metody te zostają opisane i wytłumaczona zostaje specyfika ich działania. W kolejnym rozdziale zostaje określony kierunek badań, który jest ukierunkowany na porównanie algorytmów pod względem szybkościowym i pod względem zajmowanej przestrzeni pamięciowej. Następnie zostaje przestawiony szczegółowy opis danych testowych, które będą użyte do przetestowania wybranych algorytmów. Środowiskiem jakie zostało wybrane do przeprowadzania testów jest Atom wraz z oprogramowaniem Julia. W tym języku na podstawie zgromadzonych danych zostają wykreślone charakterystyki czasowe i pamięciowe poszczególnych algorytmów w funkcji stopnia macierz. W ostatnim rozdziale została przeprowadzona analiza wszystkich uzyskanych danych, zostaje wybrany najbardziej optymalny algorytm spośród badanych oraz zostają wyciągnięte wnioski z przeprowadzonych badań.

Kluczowe słowa: wartość własna, wektor własny, macierz, Atom, Julia, prędkość działania, zajętość pamięciowa

1. Wstęp

Wyznaczanie wartości własnych macierzy to jeden z podstawowych i najbardziej czasochłonnych etapów projektowania nowoczesnych struktur informatycznych. Z problemem tym możemy się spotkać także realizując i wprowadzając w życie innowacyjne pomysły inżynierskie. Dobranie skutecznej metody obliczania wartości własnych macierzy to gwarancja pozytywnego wyniku i optymalnego czasu przetwarzania programu. Podstawowym zastosowaniem wartości własnych są przekształcenia liniowe, których to wektor własny ma charakterystyczne właściwości. Kolejnym zastosowaniem jest diagonalizacja macierzy [1], pozwala ona na łatwe potęgowanie i pierwiastkowanie macierzy, co otwiera nowe możliwości do rozwiązywania wielu problemów inżynierskich. Zastosowanie możemy znaleźć również w algorytmach PCA [2], stosowanych do kompresji sygnałów i danych. Algorytm ten jest stosowany między innymi w uczeniu maszynowym i sieciach neuronowych, jest to dziedzina nauki w pokłada się coraz większe nadzieje oraz stawia się im coraz bardziej wymagające zadania. Jest to spowodowane coraz większą chęcią tworzenia rzeczy inteligentnych, podejmujących samodzielnie poprawne decyzje. W tej właśnie dziedzinie możemy doszukać się kolejnego algorytmu wykorzystującego wartości własne. Jest to algorytm ICA [3], stosowany do rozkładu mieszaniny sygnałów na sygnały oryginalne. Algorytm ten możemy znaleźć w codziennym życiu, ponieważ podstawowym zastosowaniem tego algorytmu jest rozdzielanie sygnałów mowy w telefonii komórkowej. Wychodząc z dziedzin informatycznych a przechodząc do automatyki, odnajdziemy własności własne w wielu przykładach rozwiązywania równań różniczkowych, których wykorzystywanie jest używane w regulatorach PID [4], stanowiących integralną cześć sterowników PLC. Sterowniki te są wykorzystywane w szeroko pojętym przemyśle, dlatego też urządzenia te mają być niezawodne i szybkie. Głównym założeniem tych układów sterowania jest wydajność, dlatego implementacja w nich najlepszych algorytmów wyznaczania wartości własnych jest tak ważna.

2. Zagadnienia teoretyczne

Z definicji, jeżeli przekształcenie **A** przekształca prostą w siebie, to mówimy, że **v** jest wektorem własnym przekształcenia **A**. Oznacza to:

$$Av = \lambda v$$

dla pewnej liczby rzeczywistej **λ**, zwanej wartością własną związaną z wektorem własnym **v**. Innymi słowami wektory i własności własne są to wielkości opisujące endomorfizm (przekształcenie liniowe) danej przestrzeni liniowej. Wektor własny można rozumieć jako wektor, którego kierunek nie zmienia się po przekształceniu go endomorfizmem, zmianie ulega jedynie jego długość. Wartość własna może być rozumiana jako skala podobieństwa wektora przed przekształceniem do wektora będącego wynikiem endomorfizmu.

3. Przegląd literatury

Badania opublikowane nie później niż w roku 2016 sprowadzają problem wyznaczania wartości własnych do: **metod potęgowych** [5] które są wykorzystywane dla macierzy o wartościach rzeczywistych. Metoda ta, do wyprowadzenia wartości i wektora własnego macierzy, wykorzystuje szereg mnożeni na dowolnym wektorze z bazy w celu wyznaczenia głównej składowej szukanego wektora, **metod Lanczos** [6] które są zoptymalizowanym algorytmem bazującym na metodzie potęgowej. Działanie algorytmu polega na szukaniu nowego wektora bazowego w każdej iteracji, **metod QR** [7] która wywodzi się z metody LR, ale w celu zapewnienia stabilności numerycznej macierz L została zamieniona na macierz ortogonalną. Działanie metody polega na otrzymaniu ciągu podobnych do siebie macierzy, **metod Householder** [7] wykorzystującej do wyznaczenia wartości własnych transformacje geometryczne lustrzanego odbicia, **metod gradientu sprzężonego** [7] będącą kolejnym przykładem metody iteracyjnej**, metod Jacobiego** [8] bazującej na

przekształceniu przez podobieństwo oraz **metod Hilberta** [9] polegająca na iteracyjnym wykorzystaniu macierzy Hilberta do wyznaczanie wartości własnych macierz.

4. Wybór algorytmów do badań

Jako główne założenie raportu zostało postawione porównanie działania algorytmów wyznaczania wartości własnej. Porównywane algorytmy muszą różnić się od siebie specyfiką działania, dlatego zostały wybrane 3 algorytmy przedstawiające odmienne drogi do wyznaczenia wartości własnej macierzy. Tymi metodami są: metoda **potęgowa** opierająca się na prostych metodach iteracyjnych, **metoda Householdera**, której głównym celem jest redukcja macierzy do postaci trójdiagonalnej oraz **metoda QR** Grama -Schmidta**.**

4.1. Metoda potęgowa

Wykorzystuje własność:

$$|\lambda_1| \geq |\lambda_2| \geq \ldots |\lambda_n|$$

oraz własność, że każdy wektor może być wyrażony za pomocą kombinacji liniowej bazy zbudowanej z wektorów własnych:

$$X_2 = [A]X_1 = [A]\sum_{i=1}^{n} c_i v_i = \sum_{i=1}^{n} c_i [A] v_i = \sum_{i=1}^{n} c_i \lambda_i v_i$$

$$X_3 = [A]X_2 = [A]\sum_{i=1}^{n} c_i \lambda_i v_i = \sum_{i=1}^{n} c_i \lambda_i ([A] v_i) = \sum_{i=1}^{n} c_i \lambda_i^2 v_i$$

Co pozwala zapisać wyrażenie rekurencyjnie wektora własnego a następnie przekształcić do postaci:

$$X_{(r+1)} = \lambda_1^r \left[c_1 v_1 + c_2 \left(\frac{\lambda_2^r}{\lambda_1^r} \right) v_2 + \ldots + c_n \left(\frac{\lambda_n^r}{\lambda_1^r} \right) v_n \right]$$

Jest to wzór, z którego korzysta metoda potęgowa, wielokrotne iterowanie tego wzoru pozawala określić wartość i wektor własny macierzy.

4.2. Metoda QR

Zakłada zbudowanie bazy ortonormalnej z dowolnej podprzestrzeni liniowej za pomocą procedury ortogonizacji Grama-Schmidta, zgodnie ze schematem:

$$\begin{array}{lccccc} & \mathcal{B} = (v_1, v_2, \ldots, v_n) & \longrightarrow & \mathcal{P} = (y_1, y_2, \ldots, y_n) & \longrightarrow & \mathcal{Q} = (u_1, u_2, \ldots, u_n) \\ \text{Baza:} & \text{wyjściowa} & & \text{ortogonalna} & & \text{ortonormalna} \end{array}$$

Procedura ta pozwala zbudować ciąg ortonormalnych złożonych z kombinacji liniowych wektorów, który można wyrazić wzorem:

$$u_k = \left\| x_k - \sum_{i<k} \langle x_k, u_i \rangle u_i \right\|_2^{-1} \left[x_k - \sum_{i<k} \langle x_k, u_i \rangle u_i \right]$$

Dla liniowo niezależnych wektorów x_1, x_2, …, x_n

Następnym etapem prowadzącym do wyznaczenia wartości własnych macierzy jest faktoryzacja, czyli proces w kategorii obiektów wyposażonych w produkty. Może być rozumiany jako iloczyn, który dla danego obiektu matematycznego prowadzi do wskazania takich pod obiektów, których iloczyn jest równy wartości tego obiektu. W naszym przypadku faktoryzacja odbędzie się z użyciem macierzy wynikowej procedury Grama - Schmidta. Faktoryzacja polega na przedstawieniu danych w postaci:

$$\mathbf{A} = \mathbf{QR}$$

Dane w tej postaci nie generują problemów w dziedzinie matematycznej i w efekcie możliwe jest wyznaczenie wartości własne macierzy.

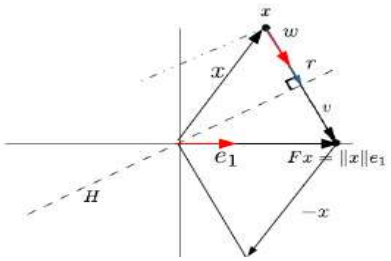
4.3. Metoda Houseldera

Metoda, aby przekształcić macierz do postaci trójdiagonalnej wykorzystuje mnożenie macierzy **A** przez macierze unitarne. W tej metodzie macierzami unitarnymi stasowane są specjalnie dobrane odbicia prowadzące do eliminacji wartości poniżej diagonali.

$$\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{Q_1} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{Q_2} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{Q_3} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{matrix} a \\ A \end{matrix} \qquad Q_1 A \qquad Q_2 Q_1 A \qquad Q_3 Q_2 Q_1 A$$

Kluczem tej metody jest dobór odpowiednich odbić takich, aby przekształcany wektor x był równoległy do pierwszego wersora e_1.



Wersor e_1 będzie specjalnie dobierany tak, że będzie miał tylko pierwszy element niezerowy. Aby tego dokonać wykorzystamy odbicie względem płaszczyzny zdefiniowanej przez wektor:

$$v = ||\mathrm{x}||\mathrm{e}_1 - \mathrm{x}.$$

Zakładając, że aby wykonać odbicie względem hiperpłaszczyzny ortogonalnej do wektora v, musimy wymnożyć go przez macierz unitarną:

$$F = I - 2\frac{vv^*}{v^*v}$$

Możemy zauważyć, że:

$$||x||e_1 = x - 2r = x - 2(w^T x)w = x - 2(ww^T)x = (I - 2ww^T)x = Px = Hx$$

I wyprowadzić transformację Householdera:

$$\mathbf{Q} = \mathbf{I} - 2ww^T \qquad w = \frac{v}{\sqrt{v^T v}}$$

Która pomoże nam ostatecznie skonstruować ciąg odbić :

$$\mathbf{X}_k = \underbrace{\mathbf{Q}_{k-1}\cdots\mathbf{Q}_1}_{\mathbf{Q}}\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & \dots & \dots & x_{1m} \\ & x_{22} & x_{23} & \dots & \dots & \dots & x_{2m} \\ & & x_{33} & \dots & \dots & \dots & x_{3m} \\ & & & \ddots & \dots & \dots & \vdots \\ & & & & x_{kk} & \dots & \vdots \\ & & & & x_{k+1,k} & \dots & x_{k+1,m} \\ & & & & \vdots & \vdots & \vdots \\ & & & & x_{n,k} & \dots & x_{n,m} \end{bmatrix} = \mathbf{R_k}$$

Podobnie jak w przypadku metody QR, ostatnim etapem przed wyznacznniem wartości wałsnych jest faktoryzacja, a więc przedstawienie danych w postaci:

$$\mathbf{A} = \mathbf{QR}$$

i obliczenie wartości i wektora własnego macierzy.

5. Kierunek badań

Po przedstawieniu badanych algorytmów należy zdefiniować zagadnienia jakie będą badane w raporcie. Aby otrzymać jasną i rzetelną odpowiedź który z badanych algorytmów jest najlepszy. Zdecydowano się wyznaczyć, jak zmieniają się czasy trwania oraz zajętość pamięciowa poszczególnych algorytmów w funkcji stopnia macierzy danych wejściowych.

6. Faza testów

Do przeprowadzenia testów zdecydowano się na wykorzystanie języka programowania Julia. Jest to stosunkowo nowy język programowania, który został stworzony głównie do rozwiązywania problemów naukowej natury. Język pozwala na szybkie tworzenie nowych bibliotek i posiada składnie przyjazną obliczeniom matematycznym. Ponadto, posiada porównywalne wyniki pod względem szybkości obliczeniowej do języka C. W naszym przypadku został on połączony ze środowiskiem Atom z zaadaptowanymi bibliotekami: LinearAlgebra, Plots oraz BenchmarkTools, które pozwoliło na uzyskanie w pełni funkcjonalnego stanowiska badawczego.

Dane techniczne stanowiska badawczego

Procesor:
Intel i7-6700HQ 2.60GHz
Pamięć RAM:
DDR4 2133MHz 16GB
Dysk twardy:
WD Black SN750 NVMe SSD

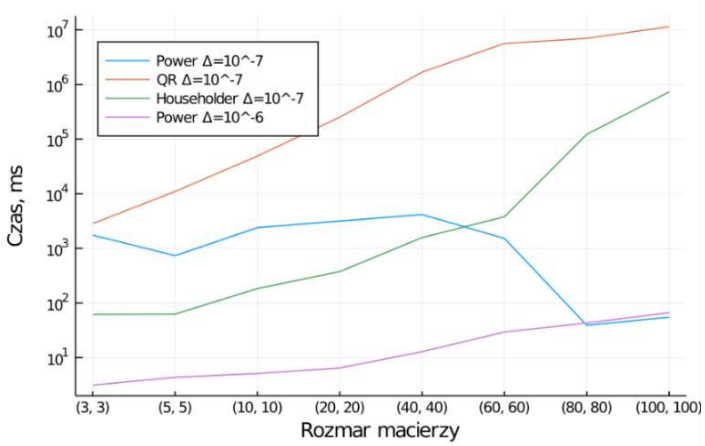
Pierwszym etapem przeprowadzonych badań było przygotowanie danych testujących. Dane te będą reprezentowane przez macierze kwadratowe o stopniach: 3, 5, 10, 20, 40, 60, 80, 100. Dane wejściowe, aby były czytelne i uporządkowane zostały umieszczone w specjalnie przygotowanej macierzy. Aby zwiększyć wiarygodność rezultatów dla każdego rozmiaru macierz, wartości zostały wygenerowane 21 razy. Do losowania liczb wykorzystano funkcję *rand*, która losuje wartości z przedziału [0,1). Po przygotowaniu danych testowy, została utworzona grupa BenchmarkGroup w której zawarte są testów wydajności dla różnych kombinacji algorytmów wyznaczania wartości własnych macierzy oraz rozmiaru macierzy. Podczas testów zwrócono uwagę na czas jaki potrzebował algorytm do obliczeń oraz na zajętość pamięciową. Dla danej kombinacji (metoda wyznaczania oraz jedna wylosowana macierz) funkcja wykonująca benchmark w zależności od rozmiaru macierzy wykonywała od kilkunastu do kilkuset wyznaczeni wartości własnej. Spośród wszystkich uzyskanych rezultatów dla jednej kombinacji brano pod uwagę wartość środkową (mediana). Nieznaczna część wyników (prawdopodobnie ze względu na czas dostępu do dysku), wymagała znacznie dłuższego czasu na wyznaczenie wartości własnych. Z tego powodu uwzględniano medianę zamiast wartości średnie. Wartości środkowe zawarto w Tabeli 1 przedstawiającej czas trwania algorytmów.

Tabela 1 Czas trwania algorytmów (ms) w zależności od rozmiaru macierzy oraz algorytmu wyznaczania wartości własnych

	Power $\Delta=10^{-7}$	QR $\Delta=10^{-7}$	Householder $\Delta=10^{-7}$	Power $\Delta=10^{-6}$
3x3	$1.74 \cdot 10^3$	$2.84 \cdot 10^3$	$6.20 \cdot 10^1$	3.14
5x5	$7{,}34 \cdot 10^2$	$1.10 \cdot 10^4$	$6.26 \cdot 10^1$	4.36
10x10	$2.40 \cdot 10^3$	$4.89 \cdot 10^4$	$1.83 \cdot 10^2$	5.11
20x20	$3.16 \cdot 10^3$	$2.52 \cdot 10^5$	$3.76 \cdot 10^2$	6.46
40x40	$4.15 \cdot 10^3$	$1.68 \cdot 10^6$	$1.57 \cdot 10^3$	$1.28 \cdot 10^1$
60x60	$1.52 \cdot 10^3$	$5.63 \cdot 10^6$	$3.79 \cdot 10^3$	$2.94 \cdot 10^1$
80x80	$3.90 \cdot 10^2$	$7.00 \cdot 10^6$	$1.22 \cdot 10^5$	$4.30 \cdot 10^1$
100x100	$5.50 \cdot 10^2$	$1.14 \cdot 10^7$	$7.32 \cdot 10^5$	$6.65 \cdot 10^1$

W kolumnach tabeli zostały przedstawione analizowane algorytmy wyznaczania wartości własnej macierzy, a w kolejnych wierszach można znaleźć coraz to większe stopnie macierzy wejściowych.

Język Julia został wyposażony w bibliotekę, której użycie pozwala na wizualizację uzyskanych danych. Dane z tabeli zostały zwizualizowane z użyciem biblioteki Plots języka programowania Julia.



Rysunek 1 Mediana czasu w zależności wyznaczania wektorów własnych od rozmiaru macierzy

Na wykresie możemy zobaczyć 4 łamane obrazujące medianę czasu przetwarzania w funkcji rozmiaru macierzy. Możemy zauważyć, że metoda potęgowa została zawarta na wykresie dwa razy. Jest to metoda iteracyjna w której czas trwania jest zależny od ilości iteracji. Ilość iteracji w naszym przypadku jest uzależniona od zmiany wartości własnych macierzy w kolejnych 2 iteracjach. Wykres zawiera 2 przypadki tej metody, dla których różnice w kształcie wykresu były największe.

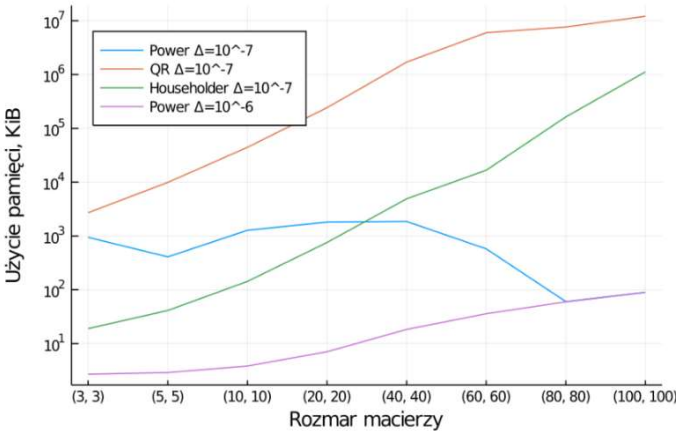
Powtarzając metodę wyznaczania danych została wygenerowana Tabela 2. zawierająca zajętość pamięciową badanych algorytmów. Dane potrzebne do stworzenia

tabeli odpowiadają przedstawionym powyżej czasom trwania algorytmów.

Tabela 2 Zajętość pamięciowa algorytmów w kiB w zależności od rozmiaru macierzy oraz algorytmu wyznaczania wartości własnych

	Power $\Delta=10^{-7}$	QR $\Delta=10^{-7}$	Householder $\Delta=10^{-7}$	Power $\Delta=10^{-6}$
3x3	$9.40 \cdot 10^2$	$2.70 \cdot 10^2$	$1.89 \cdot 10^1$	2.69
5x5	$4.07 \cdot 10^2$	$9.88 \cdot 10^3$	$4.12 \cdot 10^1$	2.88
10x10	$1.27 \cdot 10^3$	$4.43 \cdot 10^4$	$1.42 \cdot 10^2$	3.81
20x20	$1.80 \cdot 10^3$	$2.42 \cdot 10^5$	$7.55 \cdot 10^2$	7.04
40x40	$1.85 \cdot 10^3$	$1.70 \cdot 10^6$	$4.89 \cdot 10^3$	$1.83 \cdot 10^1$
60x60	$5.74 \cdot 10^2$	$5.98 \cdot 10^6$	$1.66 \cdot 10^4$	$3.58 \cdot 10^1$
80x80	$6.01 \cdot 10^1$	$7.64 \cdot 10^6$	$1.63 \cdot 10^5$	$5.96 \cdot 10^1$
100x100	$8.95 \cdot 10^1$	$1.21 \cdot 10^7$	$1.12 \cdot 10^6$	$8.96 \cdot 10^1$

Ponownie dane zostały zwizualizowane w języku programowania Julia i przedstawione na poniższym wykresie.



Rysunek 2 Użycie pamięci w zależności od rozmiaru macierzy

7. Wnioski

Analizując sporządzone wykresy możemy stwierdzić, że algorytmem, który najszybciej wyznaczył wartości własne we wszystkich rodzajach macierzy był algorytm potęgowy. Ta metoda wyznaczania, zaalokowała przy prowadzonych obliczeniach również najmniejszą ilość pamięci. Zwycięskiemu algorytmowi nie można jednoznacznie przypisać miana najlepszego algorytmu. Warto zauważyć, że metoda potęgowa potrafi wyznaczyć jedynie główną wartość własną macierzy, kiedy metody Househldera i QR wszystkie. Kolejnym spostrzeżeniem może być znaczne wydłużenie czasu pracy przy zmianie parametru delta odpowiedzialnego za dokładność uzyskiwanej wartości wektora własnego macierzy. Z przebiegów wynika, że dla większej dokładności (mniejszej wartości parametru delta) wynika, potrzeba znacznie dłuższego czasu wyznaczania. Spowodowane jest to tym, że algorytm dla niektórych macierzy nie jest w stanie uzyskać delty mniejszej niż progowa i związku z tym wyznaczanie wartości kończy się po określonej ilości iteracji (1000). Dla każdego rozmiaru macierzy testy wykonywane są dla 21 różnych macierzy, wskutek czego rezultaty są uśrednione.

Warto również wspomnieć ze obliczane czasy przetwarzania poszczególnych algorytmów są ściśle związane z parametrami użytkowymi stanowiska pomiarowego, w efekcie czego w zależności od mocy obliczeniowych czasy te mogą ulec zmianie. Nie wpłynie to jednak na pozycje najszybszego algorytmu.

Szukając zależności pomiędzy czasem i zajętością pamięciową możemy zauważyć związek, że czas przetwarzania algorytmu jest proporcjonalny do zajmowanej przez algorytm pamięci.

Podsumowując, metoda potęgowa pozwala na bardzo szybkie wyznaczenie tylko wybranych wartości własnych. Jeżeli zależy nam na wyznaczeniu wszystkich wartości własnych z dużą precyzją, należy skorzystać z metody Householdera, która okazała się szybszą metodą niż QR Grama-Schmidta.

8. Źródła i inspiracje

Programy sporządzone podczas testów zostały dołączone do raportu. Źródła zawierające sprawdzane algorytmy: Householder QR [10], QR algorithm for eigenvalues [11], Power Method with Inverse & Rayleigh [12]. Ponadto dodatkowe informacje o wartościach własnych macierzy były zaczerpnięte ze strony studia informatycznego [13] oraz materiałów „Algorytmy w inżynierii danych" [14]

Bibliografia

[1] A. Pieper, M. Kreutzer, A. Alvermann i M. Galgon, „High-performance implementation of Chebyshev filter diagonalization for interior eigenvalue computations," 2016.

[2] M. Zhai, „The PLC Signals' Noise Mitigating Algorithm with PCA," 2017.

[3] M. N. Patil, B. Iyer i R. Arya, „Performance Evaluation of PCA and ICA Algorithm for Facial Expression Recognition Application," 2016.

[4] K. Rajagopal i A. S. Guessas Laarem: Anitha Karthikeyan, „FPGA implementation of adaptive sliding mode control and genetically optimized PID control for fractional-order induction motor system with uncertain load," 2017.

[5] M. Tammen, I. Kodrasi i S. Doclo, „COMPLEXITY REDUCTION OF EIGENVALUE DECOMPOSITION-BASED DIFFUSE POWER SPECTRAL DENSITY ESTIMATORS USING THE POWER METHOD," 2018.

[6] A. R. d. Faria, „Adaptation of the Lanczos Algorithm for the Solution of Buckling Eigenvalue Problems," 2018.

[7] T. Lyche, „Numerical Linear Algebra and Matrix Factorizations," 2020.

[8] Z.-W. Sun, „Generalized inverse eigenvalue problems for augmented periodic Jacobi Matrices," 2019.

[9] R. R. Sharma i R. B. Pachori, „A New Method for Non-stationary Signal Analysis using Eigenvalue Decomposition of the Hankel Matrix and Hilbert Transform," 2017.

[10] [Online]. Available: https://www.youtube.com/watch?v=d-yPM-bxREs.

[11] https://www.youtube.com/watch?v=_neGVEBjLJA.

[12] https://www.youtube.com/watch?v=LHlg_lfihiA.

[13] http://wazniak.mimuw.edu.pl/index.php?title=MN13.

[14] https://isod.ee.pw.edu.pl/isod-tud/?wicket:bookmarkablePage=:isod.app.courseinfo.CourseInfoPage&idCourseDef=3846.

„Oświadczam, że niniejsza praca stanowiąca podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu 1DI2153:A - Algorytmy w inżynierii danych została wykonana przeze mnie samodzielnie. ”

15.06.2020
Królik Jarosław
Derszniak Mateusz