Akademia Górniczo-Hutnicza w Krakowie

Wydział Fizyki i Informatyki Stosowanej

Metody Numeryczne

Laboratorium 04: Wektory i Wartości Własne. Metoda Potęgowa

Andrzej Świętek

19.03.2024

Contents

1. Wstęp teoretyczny

1.1. Iloczyn tensorowy

Iloczyn tensorowy jest operacją używaną w algebrze liniowej do tworzenia nowych przestrzeni wektorowych lub macierzy z istniejących przestrzeni wektorowych lub macierzy. Jest to operacja, która łączy dwie struktury tensorowe, tworząc nową strukturę.

Dla wektorów

Dla dwóch wektorów $\mathbf{u}$ i $\mathbf{v}$ w przestrzeniach wektorowych $\mathbb{R}^m$ i $\mathbb{R}^n$ odpowiednio, iloczyn tensorowy wektorów definiowany jest jako macierz:

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^T = \begin{bmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_n \\ u_2v_1 & u_2v_2 & \cdots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \cdots & u_mv_n \end{bmatrix}$$

Dla macierzy

Dla dwóch macierzy A i B o wymiarach $m \times n$ i $p \times q$ odpowiednio, iloczyn tensorowy macierzy definiowany jest jako macierz:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}$$

1.2. Metoda Potęgowa

Początek: Na początek potrzebujemy macierzy kwadratowej A oraz wektora początkowego $x^{(0)}$. Wektor ten może być losowy lub dobrany w inny sposób, ale powinien być niezerowy i niezależny od wektora własnego, którego szukamy.
Iteracje: Algorytm wykonuje następujące kroki w każdej iteracji:

1. Obliczanie następnego przybliżenia wektora własnego: $y^{(k)} = A \cdot x^{(k)}$
2. Normalizacja wektora: $x^{(k+1)} = \frac{y^{(k)}}{\|y^{(k)}\|}$
3. Obliczanie wartości własnej: $\lambda^{(k)} = \frac{(x^{(k+1)})^T \cdot A \cdot x^{(k+1)}}{(x^{(k+1)})^T \cdot x^{(k+1)}}$

Warunek stopu: Algorytm kończy się, gdy wystarczająco blisko zbiegnie do dominującego wektora własnego. Może to być ustalony limit liczby iteracji lub gdy różnica między kolejnymi wartościami własnymi jest wystarczająco mała.
Wynik: Po zakończeniu iteracji, $x^{(k)}$ jest przybliżeniem dominującego wektora własnego, a $\lambda^{(k)}$ jest odpowiadającą mu wartością własną.

Wartości własne wyznaczymy iteracyjnie, przy użyciu metody potęgowej, zgodnie z poniższym algorytmem:

Algorithm 1 Metoda potęgowa z iloczynem tensorowym

$W_0 = A$ ▷ Inicjalizacja macierzy iterującej
for $k = 0$ **to** K_{val} **do**
  $x_0^k = [1, 1, \ldots, 1]$ ▷ Inicjalizacja wektora startowego
  for $i = 1$ **to** IT_{MAX} **do**
    $x_{i+1}^k = W^k x_i^k$
    $\lambda_i^k = \frac{(x_{i+1}^k)^T x_i^k}{(x_i^k)^T x_i^k}$
    $x_i^k = \frac{x_{i+1}^k}{\|x_{i+1}^k\|_2}$
  end for
  $W_{k+1} = W_k - \lambda_k x_i^k (x_i^k)^T$ ▷ Iloczyn tensorowy
end for

2. Problem

Napisać program znajdujący wektory własne i wartości własne metodą potęgową dla macierzy A:

$$A_{i,j} = \frac{1}{\sqrt{2 + |i - j|}}$$

gdzie: $i, j = 0, 1, \cdots, n-1$. Macierz jest symetryczna więc ma wszystkie wartości własne rzeczywste, podobnie jak składowe wszystkich wektorów własnych

3. Implementacja

3.1. Inicjalizacja

Funckja initMatrix wypełnia macierz A i macierz iteracyjną W zgodnie poleceniem zadania.

```
constexpr int N = 7;
constexpr int Kval = N;
constexpr int IT_MAX = 12;

ofstream eigenvalues_file("eigenvalues.txt");
ofstream matrix_D_file("matrix_D.txt");

double A[N][N];
double W[N][N];

initMatrix(A,N);
initMatrix(W,N);
```

Listing 1: Inicjalizacja i deklaracje

3.2. Metoda Potęgowa

W głównej części programu znajduje się tylko wywołanie funckji odpowiedzialnej za wyliczenie wszystkich wartości włąsnych.

```
PowerMethodResult eigen_data =
                    powerMethod(W, Kval, N, IT_MAX, eigenvalues_file);
cout << "[ Eigen Values ]: \n";
for ( double val : eigen_data.eigen_values )
    cout << val << " ";
cout << "\n\n";
```

Listing 2: Wywołanie metody potęgowej

```
struct PowerMethodResult {
    vector<double> eigen_values;                    // wektor z wartosciami
    wlasnymi
    vector<vector<double>> eigen_vector_matrix;
                    // macierz wektorow wlasnych (transponowana)
};
```

Listing 3: Typ zwracany przez funckję powerMethod

Ważnym elementem mojej implementacji jest typ zwracany przez funckje *powerMethod*. Jest to struktura zawierajaca zarówno wartości własne jak i macierz wektorów własnych która jest transponowana. Ten szczegół będzie miał duże znaczenie przy wyznaczaniu macierzy diagonalnej.

```
PowerMethodResult powerMethod(
    double W[][7], int Kval,
    int N, int IT_MAX
) {
    PowerMethodResult data;

  for(int k =0; k < Kval; k++)  // Tyle razy ile wartosci wlasnych
  {
      vector<double> x_k(N, 1.0); // Inicjalizacja wektora startowego
      double lambda = 0;

      for (int i = 0; i < IT_MAX; i++) {
        // Obliczanie iloczynu macierz-wektor
        vector<double> result(N, 0.0);
        result = MatrixDotVector(W, x_k, N);    // x_k+1

        // Obliczenie lambda
        double new_product = VectorDotProduct(result, x_k, N);
        double old_product = VectorDotProduct(x_k, x_k, N);

            // nowy iloczyn skalany / starny
        lambda = new_product / old_product;

        old_product = new_product;            // zamiana mianownika

        // Normalizacja wyniku
        normalize(result);

        x_k = result;
      }
      data.eigen_values.push_back(lambda);
      data.eigen_vector_matrix.push_back(x_k);

      // Aktualizacja macierzy iteracyjnej
      for(int i =0; i < N; i++)
          for(int j =0; j < N; j++)
              W[i][j] = W[i][j] - lambda * x_k[i] * x_k[j];
    }

    return data;
}
```

Listing 4: Metoda potęgowa

Sama metoda potęgowa narzuca implementacje.

1. Ponieważ szukamy $Kval$ wartości własnych wykonujemy metodę tyleż samo razy
   (a) Tworzymy wektor $\vec{x_k}$ wypełniony wartościami "1". Będzie on służył jako poprzedzające rozwiązanie. i wykonujemy ponoższ epolecenia określoną ilość razy
      i. Wyznaczamy iloraz skalarny macierzy iteracyjnej z wektorem $\vec{x_k}$
      ii. Wyznaczamy nową wartość λ na którą składa się iloraz nowego iloczynu skalarnego wektora będącego wynikiem mnożenia macierzy i wektora z samym wektorem $\vec{x_k}$, z poprzedzjącą go wartością.
      iii. Otrzymany wynik aktualizujemy a wektor będący wynikiem mnożenie macierz-wketor normalizujemy
   (b) Zapamiętujemy obecną lambdę i wektor x_k

(c) Aktualizujemy macierz iteracyjną o "wpływ obecnie dominującej wartości własnej" by w następnej iteracji wyznaczyć następną

2. Zwracamy dane

3.3. Diagonalizacja

```
Matrix transposed_X = transpose( eigen_data.eigen_vector_matrix , N);

Matrix D = multiplyMatrices(
        multiplyMatrices( eigen_data.eigen_vector_matrix, arrayToVector(A, N)),
        transposed_X
);

cout << "[ Diagonal Matrix ]: \n";
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++)
        matrix_D_file << D[i][j] << " ";

    matrix_D_file << "\n";
}
```

Listing 5: Wykonanie diagonalizacji

Na podstawie otrzymancyh wartości własnych i macierzy wektorów własnych wyznaczamy macierz diagonalną z definicji:

$$D = X^T \cdot A \cdot X$$

gdzie X to macierz wektorów własnych w której kolumnach mamy zapisane wektory własne.
Ze względu używanie kontenera std::vector wraz z metodą $push\_back()$ w mmomencie tworzenia macierzy uzyskaliśmy macierz już transponowaną. Zatem zgodnie z wzorem $(X^T)^T = X$ wykonujemy mnożenie macierzy X już ztransponowanej z macierzą A by następnie wynik tego mnożenia przemnożć przez macierz X. Aby uzyskać dokładnie macierz X musimy ztransponować macierz otrzymaną z wyniku funckji.
Ostatnim ktorkiem jest już tylko zpaisanie macierzy diagonalnej D. Wartym uwagi jest fakt że macierz uzyskana w wyniku tych operacji nie zawsze musi być dokłanie niezerowa poza diagonalą - może powstać szum w wyniku operacji zmienno przecinkowych.

4. Wyniki

4.1. Wartości własne

$$\lambda_1 = 3.59586$$
$$\lambda_2 = 0.284988$$
$$\lambda_3 = 0.122785$$
$$\lambda_4 = 0.59039$$
$$\lambda_5 = 0.0865955$$
$$\lambda_6 = 0.170974$$
$$\lambda_7 = 0.0981544$$

4.2. Macierz diagonalna

Otrzymane wartości poza diagonalą były różne od 0 ale dopiero na 16 miejscu po przecinku w związku z tym można je uznać za szum a wyniki odpowiednio zaokrąglić.

$$D = \begin{bmatrix} 3.59586 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.284988 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.122786 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.59039 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.086 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.17 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0981 & 0 \end{bmatrix}$$

4.3. Wektory własne

λ_1: 3.595

$$\vec{x_1} = \begin{bmatrix} -0.35294071 \\ -0.37793502 \\ -0.39222145 \\ -0.39688868 \\ -0.39222145 \\ -0.37793502 \\ -0.35294071 \end{bmatrix}$$

λ_2: 0.590

$$\vec{x_2} = \begin{bmatrix} -0.479169277 \\ -0.446608023 \\ -0.266342032 \\ -1.68315413 \times 10^{-16} \\ 0.266342032 \\ 0.446608023 \\ 0.479169277 \end{bmatrix}$$

λ_3: 0.284

$$\vec{x_3} = \begin{bmatrix} -0.47723532 \\ -0.16477843 \\ 0.31485123 \\ 0.54030218 \\ 0.31485123 \\ -0.16477843 \\ -0.47723532 \end{bmatrix}$$

λ_4: 0.170

$$\vec{x_4} = \begin{bmatrix} 0.449003461 \\ -0.172674577 \\ -0.518246449 \\ 5.55917146 \times 10^{-16} \\ 0.518246449 \\ 0.172674577 \\ -0.449003461 \end{bmatrix}$$

λ_5: 0.122

$$\vec{x_5} = \begin{bmatrix} 0.36070546 \\ -0.46268262 \\ -0.14222333 \\ 0.52074734 \\ -0.14222333 \\ -0.46268262 \\ 0.36070546 \end{bmatrix}$$

λ_6: 0.098

$$\vec{x_5} = \begin{bmatrix} -0.262283617 \\ 0.520312179 \\ -0.400602721 \\ 1.29767955 \times 10^{-15} \\ 0.400602721 \\ -0.520312179 \\ 0.262283617 \end{bmatrix}$$

λ_7: 0.0865

$$\vec{x_7} = \begin{bmatrix} 0.13255517 \\ -0.34049668 \\ 0.47623898 \\ -0.5285595 \\ 0.47623898 \\ -0.34049668 \\ 0.13255517 \end{bmatrix}$$

5. Wizualizacja wyników

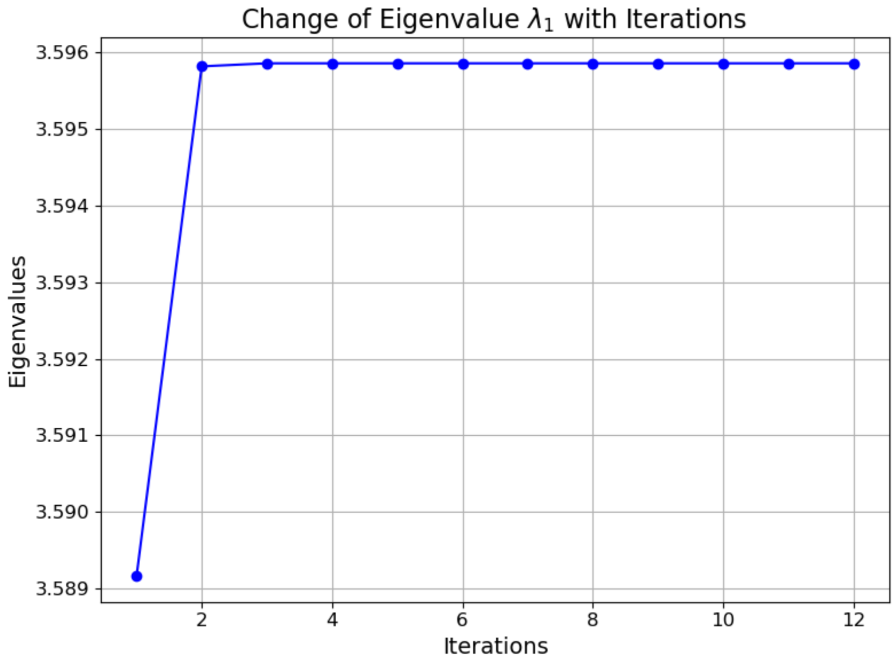

Figure 1: Wykres zmiany wartości własnej λ_1 wraz z iteracją metody potęgowej

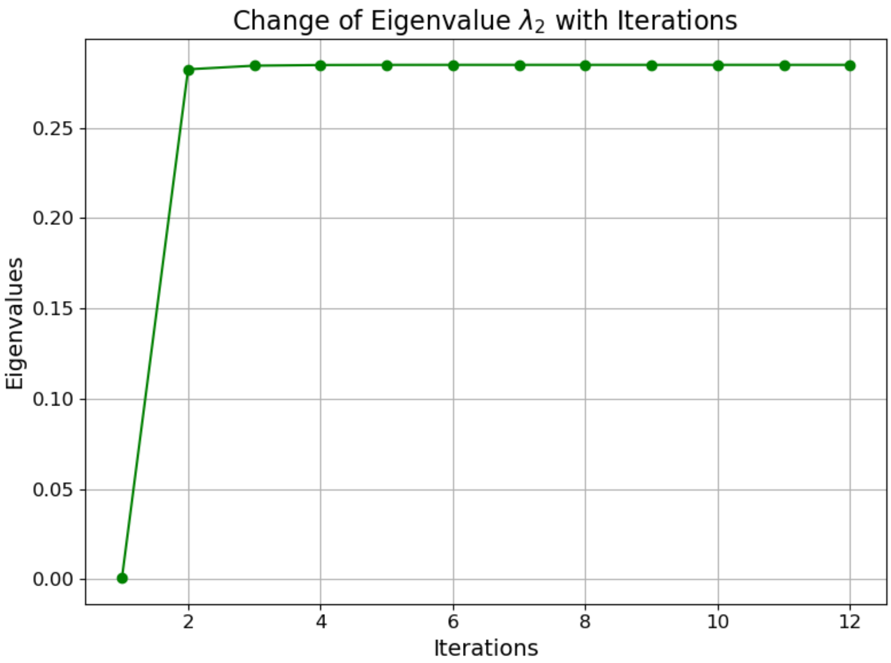

Figure 2: Wykres zmiany wartości własnej λ_2 wraz z iteracją metody potęgowej

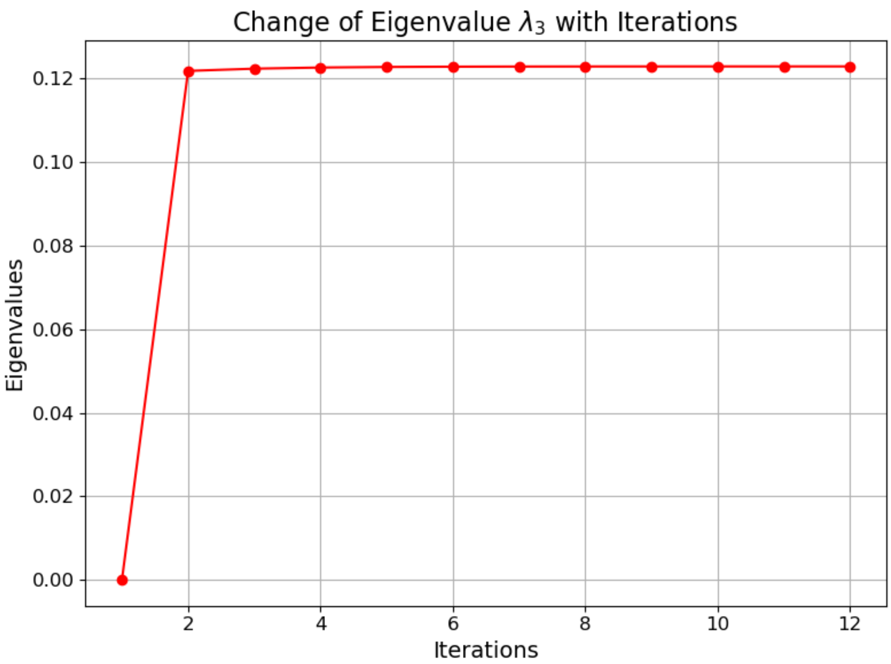

Figure 3: Wykres zmiany wartości własnej λ_3 wraz z iteracją metody potęgowej

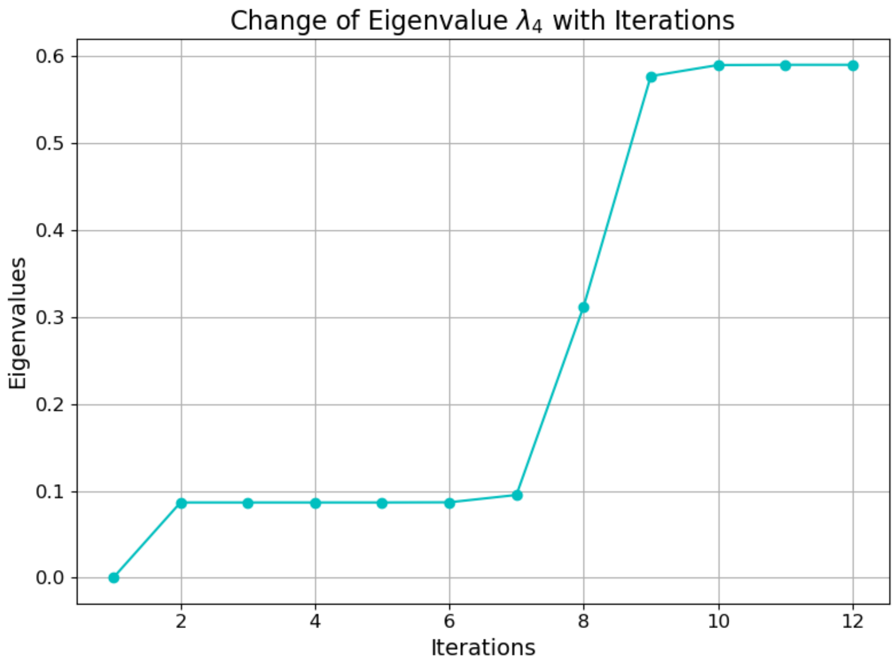

Figure 4: Wykres zmiany wartości własnej λ_4 wraz z iteracją metody potęgowej

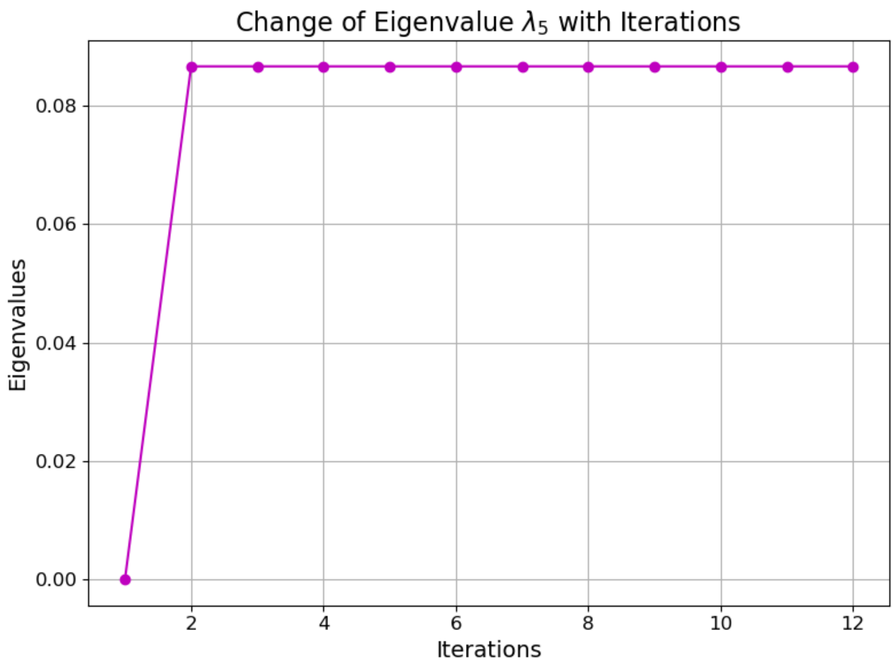

Figure 5: Wykres zmiany wartości własnej λ_5 wraz z iteracją metody potęgowej

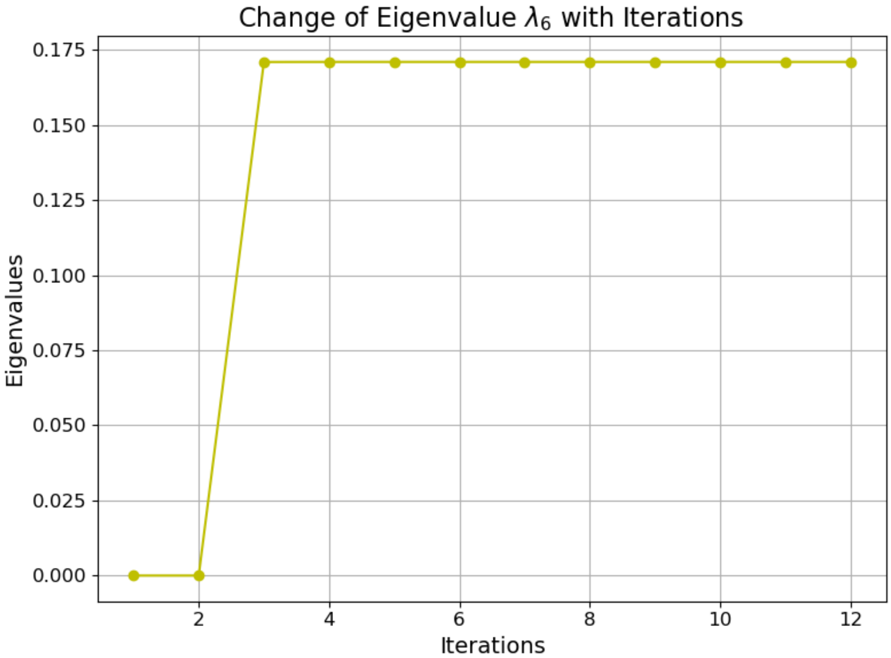

Figure 6: Wykres zmiany wartości własnej λ_6 wraz z iteracją metody potęgowej

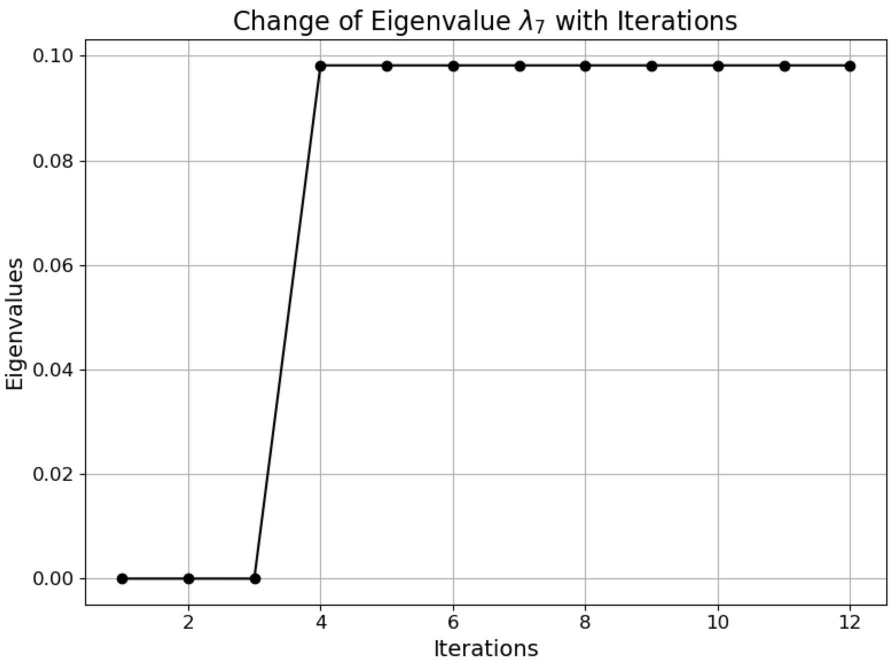

Figure 7: Wykres zmiany wartości własnej λ_7 wraz z iteracją metody potęgowej

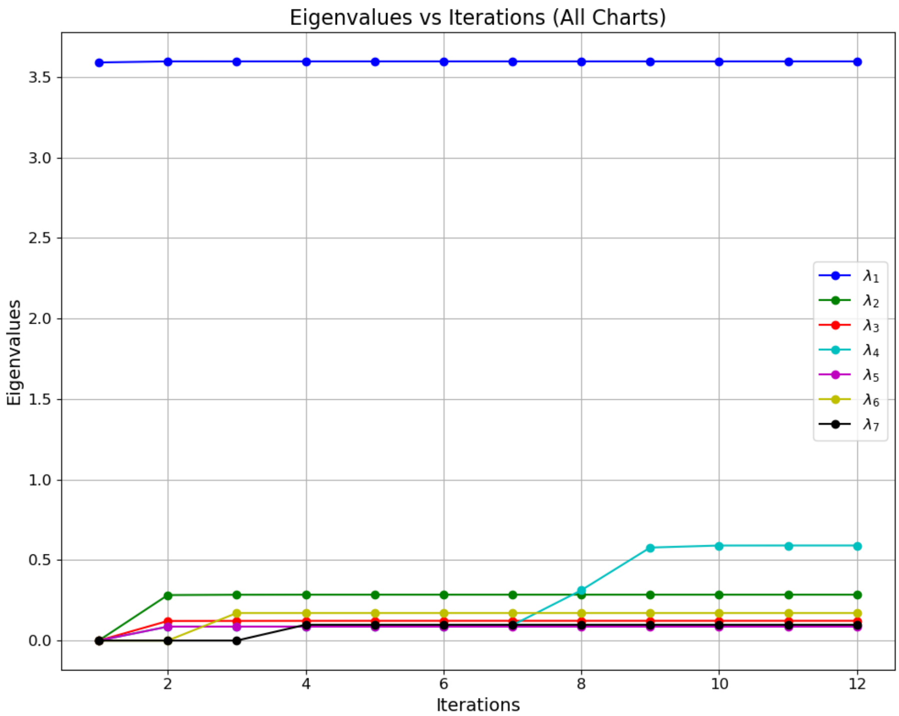

Figure 8: Wykres zmian wartości własnych λ_i wraz z iteracją metody potęgowej

6. Omówienie Kolejności wartości własnych

Wartości własne λ_i zostały znalezione przy użyciu metody potęgowej. Kolejność ich znalezienia może zależeć od różnych czynników, takich jak warunki początkowe, numeryczna dokładność obliczeń oraz własności macierzy A. W naszym przypadku, wartości własne zostały znalezione w następującej kolejności: $\lambda_1 = 3.59586$, $\lambda_2 = 0.59039$, $\lambda_3 = 0.284988$, $\lambda_4 = 0.170974$, $\lambda_5 = 0.122785$, $\lambda_6 = 0.0981544$ oraz $\lambda_7 = 0.0865$. Warto zauważyć, że wartości te są ułożone malejąco, co jest zgodne z oczekiwaniami dla dominujących wartości własnych macierzy A.

Liczba iteracji potrzebnych do znalezienia każdej wartości własnej może się różnić w zależności od szybkości zbieżności metody potęgowej dla danego wektora własnego. W naszym przypadku, ustaliliśmy maksymalną liczbę iteracji na 12 dla każdej wartości własnej. Okazało się, że większość wartości własnych została znaleziona w tej liczbie iteracji, co sugeruje dobrą szybkość zbieżności metody potęgowej dla analizowanej macierzy.

Macierz diagonalna D została wyznaczona na podstawie wartości własnych i macierzy wektorów własnych. Wartości te umieszczono na głównej przekątnej macierzy D, podczas gdy pozostałe elementy macierzy D są bliskie zeru, co jest zgodne z oczekiwaniami dla macierzy diagonalnej.

7. Wnioski

- Metoda potęgowa okazała się skutecznym narzędziem do znalezienia wartości własnych symetrycznej macierzy.
- Kolejność znalezionych wartości własnych odpowiadała oczekiwaniom, gdzie dominujące wartości własne były znacznie większe od pozostałych.
- Liczba iteracji potrzebnych do znalezienia wartości własnych była relatywnie niska, co świadczy o szybkości zbieżności metody potęgowej dla analizowanej macierzy.
- Macierz diagonalna uzyskana na podstawie wartości własnych i wektorów własnych potwierdza ich poprawność i zgodność z teorią.
- Wartości własne oraz odpowiadające im wektory własne mogą być wykorzystane do analizy dynamiki układów opisanych przez macierz A, co ma znaczenie w wielu dziedzinach nauki, w tym w fizyce, informatyce i inżynierii.