Metody numeryczne

Nikodem Kocjan

19.03.2024

Diagonalizacja macierzy metodą potęgową

Spis treści

1 Cel ćwiczenia

Celem ćwiczenia było przeprowadzenie diagonalizacji macierzy symetrycznej metodą potęgową.

2 Wstęp teorytyczny

2.1 Diagonalizacja macierzy

Diagonalizacja macierzy polega na znalezieniu takiej macierzy przekształcenia P oraz macierzy diagonalnej D, że $A = PDP^{-1}$, gdzie A jest macierzą pierwotną. Macierz D zawiera wartości własne macierzy A na swojej głównej przekątnej, a kolumny macierzy P są wektorami własnymi macierzy A. Proces diagonalizacji jest możliwy tylko dla macierzy, które posiadają pełny zestaw liniowo niezależnych wektorów własnych. Diagonalizacja ułatwia wiele obliczeń, w tym potęgowanie macierzy, ponieważ potęgowanie macierzy diagonalnej sprowadza się do potęgowania jej elementów na przekątnej.

2.2 Wartości i wektory własne macierzy

Wartość własna macierzy A to taki skalar λ, że dla pewnego niezerowego wektora v zachodzi równanie $Av = \lambda v$. Wektor v nazywany jest wektorem własnym macierzy A odpowiadającym wartości własnej λ. Wartości i wektory własne odgrywają kluczową rolę w wielu dziedzinach matematyki i inżynierii, ponieważ pozwalają na analizę właściwości przekształceń liniowych reprezentowanych przez macierze. W kontekście metody potęgowej, poszukiwanie dominującej wartości własnej oraz odpowiadającego jej wektora własnego pozwala na zrozumienie, jak kierunek o największej 'siły' (w sensie wartości własnej) wpływa na przekształcenie opisane przez macierz A.

2.3 Metoda potęgowa

Metoda potęgowa jest iteracyjną techniką numeryczną stosowaną do znajdowania wartości własnej o największej wartości bezwzględnej (dominującej) oraz odpowiadającego jej wektora własnego dla danej macierzy A. Macierz symetryczna, jaką mamy do czynienia w tym ćwiczeniu, posiada wszystkie wartości własne rzeczywiste oraz ortogonalne wektory własne, co sprzyja stosowaniu metody potęgowej.

2.4 Metoda potęgowa - algorytm

Załóżmy, że A jest macierzą kwadratową rzędu n, a $\lambda_1, \lambda_2, ..., \lambda_n$ są jej wartościami własnymi takimi, że $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n|$. Metoda potęgowa pozwala znaleźć wartość λ_1 oraz odpowiadający jej wektor własny v_1. Iteracyjny proces rozpoczyna się od przyjęcia wektora startowego x_0, który nie jest ortogonalny do v_1. Następnie, poprzez wielokrotne mnożenie x_0 przez A i normalizację wynikowego wektora, proces zbiega do wektora własnego v_1 odpowiadającego dominującej wartości własnej λ_1. Algorytm metody potęgowej można opisać w następujących krokach:

1. Wybierz początkowy wektor x_0 i ustaw $k = 0$.
2. Oblicz $y_{k+1} = Ax_k$.
3. Oblicz $x_{k+1} = \frac{y_{k+1}}{|y_{k+1}|}$, gdzie $|\cdot|$ oznacza normę wektora.
4. Jeżeli proces nie zbiegł, zwiększ k o 1 i wróć do kroku 2.

2.5 Algorytm działania z zajęć

Do wyznaczenia wartości własnych, oraz macierzy wektorów własnych, wykorzystaliśmy algorytm:

$$
\begin{array}{l}
W_0 = A \qquad \text{(inicjalizacja macierzy iterującej)} \\
for(k = 0;\ k < K_{val};\ k++)\{ \\
\quad \boldsymbol{x}_k^0 = [1, 1, \ldots, 1] \qquad \text{(inicjalizacja wektora startowego)} \\
\quad for(i = 1; i <= IT\_MAX; i++)\{ \\
\qquad \boldsymbol{x}_k^{i+1} = W_k \boldsymbol{x}_k^i \\
\qquad \lambda_k^i = \dfrac{\left(\boldsymbol{x}_k^{i+1}\right)^T \boldsymbol{x}_k^i}{\left(\boldsymbol{x}_k^i\right)^T \boldsymbol{x}_k^i} \\
\qquad \boldsymbol{x}_k^i = \dfrac{\boldsymbol{x}_k^{i+1}}{\|\boldsymbol{x}_k^{i+1}\|_2} \\
\quad \} \\
\quad W_{k+1} = W_k - \lambda_k \, \boldsymbol{x}_k^i (\boldsymbol{x}_k^i)^T \qquad \text{(iloczyn tensorowy)} \\
\}
\end{array}
$$

Rysunek 1: Algorytm wyznaczania wartości i wektorów własnych

1. Inicjalizacja macierzy iteracyjnej $W_0 = A$.
2. Ustawienie wektora startowego x^0 złożonego z samych jedynek.
3. Dla $k = 0$ aż do $k < K_{\max}$:
   - Dla $i = 1$ aż do osiągnięcia IT_{MAX}:
     (a) Obliczenie $x^{i+1} = W_k x^i$.
     (b) Wyliczenie przybliżonej wartości własnej λ_k^i jako

$$\lambda_k^i = \frac{(x^{i+1})^\mathsf{T} x^i}{(x^i)^\mathsf{T} x^i}.$$

(c) Normalizacja wektora x^{i+1}:

$$\hat{x}^i = \frac{x^{i+1}}{\|x^{i+1}\|_2}.$$

(d) Zwiększenie i o 1.

- Po zakończeniu wewnętrznej pętli iteracyjnej, aktualizacja macierzy W_{k+1} poprzez odjęcie od W_k iloczynu tensorowego:

$$W_{k+1} = W_k - \lambda_k^i \hat{x}^i (\hat{x}^i)^\mathsf{T}.$$

- Zwiększenie k o 1.

3 Deklaracja zmiennych oraz implementacja bibliotek

Opisane zagadnienie rozwiązywałem za pomocą programu Visual Studio Code w języku c++
Wykorzystane zostały biblioteki
<iostream> - standardowa biblioteka c++
<cmath> - implementuje funkcję sinus potrzebną do rozwiązywania równań
<fstream> - operacje na plikach <algorithm> - metoda copy do ulatwienia pracy Zadeklarowane stałe, oraz tablice użyte w programie to:

```
    const int N = 7;         //rozmiar macierzy A
    const int IT_MAX = 12;   //maksymalna liczba iteracji K
    double A[N][N];        //macierz A
    double war_wl[N];      //wartosci wlasne macierzy
    double wek_wl[N][N];   //wektory wlasnie macierzy
```

Fragment kodu 1: Deklaracja stalych

4 Opis działania

4.1 Uzupełnienie macierzy A

Macierz A uzupełniliśmy za pomocą wzoru:

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

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        A[i][j] = 1.0 / sqrt(2.0 + fabs(i - j));
    }
}
```

Fragment kodu 2: Uzupełnienie macierzy A

4.2 Krótki opis programu

Program, zgodnie z oczekiwaniami rozbiłem na kilka funkcji, aby nie pisać wszystkiego w funkcji main. Z punktu widzenia poprawności programowania jest to optymalne rozwiązanie. 1. Mnożenie macierzy A przez wektor wektor. Rezultat zapisywany jest w tablicy wynik:

```
void macierz_x_wektor(double A[N][N], double wektor[N], double
    wynik[N])
```

Fragment kodu 3: macierz * wektor

2. Funkcja dokonuje skalarnego iloczynu dwóch wektorów wektorA oraz wektorB. Rezultat zwracany jest jako liczba typu double.

```
double iloczyn_wektorow(double wektor_A[N], double wektor_B[N])
```

Fragment kodu 4: iloczyn wektorow

3. Normalizacja wektora wektor. Rezultat zapisywany jest jako zmiana wektora wejściowego.

```
void znormalizuj_wektor(double wektor[N])
```

Fragment kodu 5: Normalizacja wektora

4. Dzielenie wektora przez podaną liczbę (obliczoną za pomocą normalizacji)

```
void podziel_wektor_przez_liczbe(double wektor_A[N], double liczba,
     double res[N]){
```

Fragment kodu 6: Normalizacja wektora

5. Funkcja dokonuje iloczynu tensorowego zgodnie z schematem opisanym w wstępnie teorytycznym. Rezultatem jest nadpisanie macierzy A.

```
void iloczyn_tensorowy(double A[N][N], double x[N], double lambda)
```

Fragment kodu 7: iloczyn tensorowy

6. Funkcja uzupełniająca kolumnę macierzy wektorów własnych

```
void uzupelnij_kolumne_macierzy(double wek_wl[N][N],double x[N],int
     k){
```

Fragment kodu 8: iloczyn tensorowy

6. Funkcja wywołujem metodę potęgową wyznaczania wartości własnych i wektorów własnych dla macierzy A.

```
void metoda_potegowa(double A[N][N], double war_wl[N], double
    wek_wl[N][N])
```

Fragment kodu 9: metoda potegowa

4.3 void macierz_x_wektor

```
void macierz_x_wektor(double A[N][N], double wektor[N], double
    wynik[N]) {
    for (int i = 0; i < N; i++) {
        wynik[i] = 0.0;
        for (int j = 0; j < N; j++) {
            wynik[i] += A[i][j] * wektor[j];
        }
    }
}
```

Fragment kodu 10: Funkcja macierz * wektor

4.4 iloczyn_wektorow

```
double iloczyn_wektorow(double wektor_A[N], double wektor_B[N]) {
    double product = 0.0;
    for (int i = 0; i < N; i++) {
        product += wektor_A[i] * wektor_B[i];
    }
    return product;
}
```

Fragment kodu 11: Iloczyn skalarny wektorow

4.5 znormalizuj_wektor

```
double znormalizuj_wektor(double wektor[N]) {
    double norma = 0.0;
    norma = std::sqrt(iloczyn_wektorow(wektor, wektor));
    return norma;
}
```

Fragment kodu 12: Normalizacja wektora

4.6 Podziel wektor przez liczbe

```
void podziel_wektor_przez_liczbe(double wektor_A[N], double liczba,
     double res[N]){
    for(int i = 0; i < N; i++){
        res[i] = wektor_A[i] / liczba;
    }
}
```

Fragment kodu 13: Iloczyn tensorowy

4.7 iloczyn_tensorowy

```
void iloczyn_tensorowy(double A[N][N], double x[N], double lambda)
    {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            A[i][j] -= lambda * x[i] * x[j];
        }
    }
}
```

Fragment kodu 14: Iloczyn tensorowy

4.8 Uzupełnij kolumnę macierzy

```
void uzupelnij_kolumne_macierzy(double wek_wl[N][N],double x[N],int
     k){
    for (int i = 0; i < 7; i++) {
        wek_wl[i][k] = x[i];
    }
}
```

Fragment kodu 15: Iloczyn tensorowy

4.9 metoda_potegowa

```
void metoda_potegowa(double A[N][N], double war_wl[N], double
    wek_wl[N][N]) {
    double W[N][N];

    for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
          W[i][j] = A[i][j];
      }
    }
    for (int k = 0; k < N; k++) {
        double norma = 0.0;
        double x[N];
        double lambda = 0.0;
        for (int i = 0; i < N; i++) {
            x[i] = 1.0; // Inicjalizacja wektora startowego
        }


        for (int it = 0; it < IT_MAX; it++) {
            double x_next[N];
            macierz_x_wektor(W, x, x_next);
            lambda = iloczyn_wektorow(x_next, x) / iloczyn_wektorow
    (x, x);
            std::cout << "Lambda: "  << lambda << endl;
            norma = znormalizuj_wektor(x_next);
            podziel_wektor_przez_liczbe(x_next, norma, x);
        }
```

```
        war_wl[k] = lambda;
        iloczyn_tensorowy(W, x, lambda);
        uzupelnij_kolumne_macierzy(wek_wl , x, k);
    }
}
```

Fragment kodu 16: metoda potegowa

5 Diagonalizacja macierzy

5.1 Krótki opis

Wykonując kroki opisane w poprzedniej sekcji zadania, otrzymujemy macierz wektorów własnych, oraz wektor wartości własnych macierzy A. Aby sprawdzić poprawność metody, możemy transponować macierz wektorów własnych, a następnie wykonać działanie:

$$D = X^T AX$$

W rezultacie macierz D powinna na swojej diagonalii zawierać wartości własne, takie same jak wektor wartości własnych obliczony za pomocą metody potęgowej.

5.2 Rozwiązanie

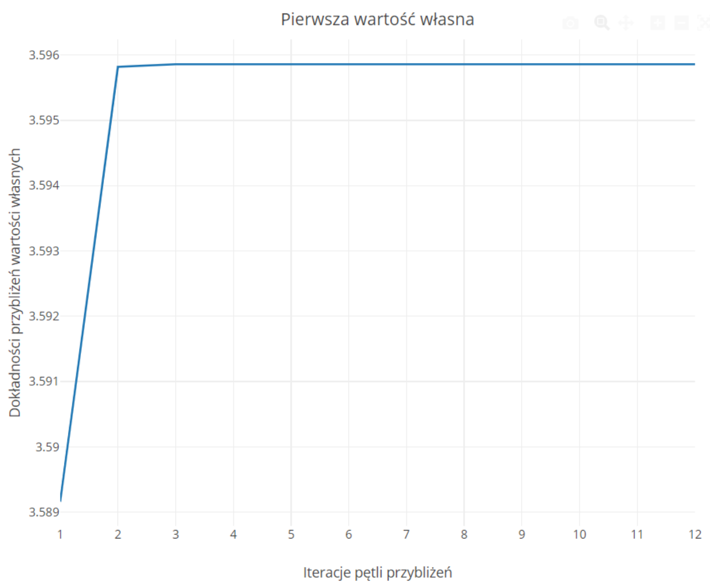
```
    // Transpozycjza macierzy wektorow wlasnych
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
          X_T[j][i] = wek_wl[i][j];
      }
    }

    //Iloczyn transponowanej macierzy wektorow
    //wlasnych z macierza A. Wynik zapisywany
    //jest do tymczasowej macierzy temp
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            for (int k = 0; k < N; k++) {
                temp[i][j] += X_T[i][k] * A[k][j];
            }
        }
    }
    //Iloczyn temp z macierza wektorow wlasnych
    //Rezultatem jest macierz diagonalna
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            D[i][j] = 0;
            for (int k = 0; k < N; k++) {
                D[i][j] += temp[i][k] * wek_wl[k][j];
            }
        }
    }
```

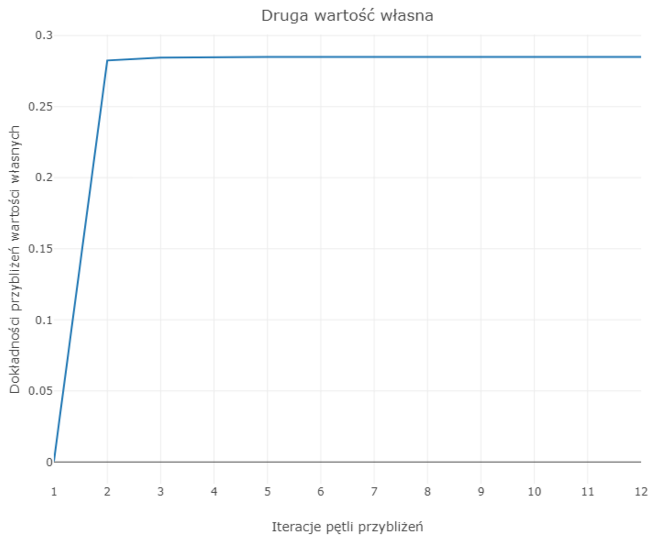
Fragment kodu 17: Diagonalizacja macierzy

5.3 Porównanie wyników

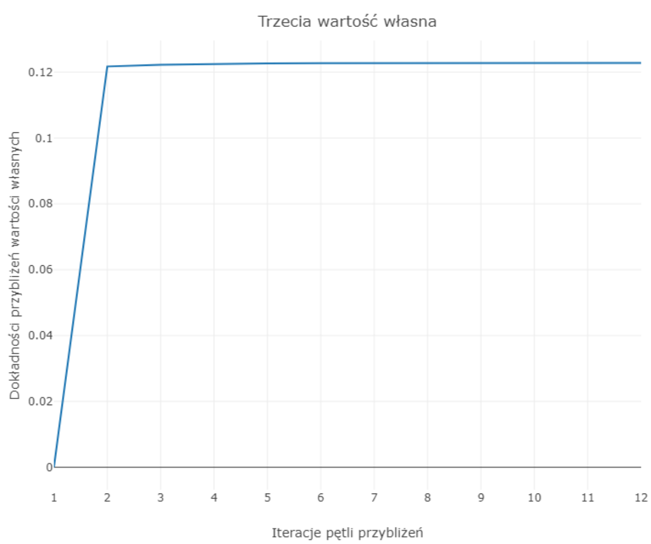
Poniższe wykresy prezentują wielkości wartości własnych zależne od danej iteracji. Na ich podstawie możemy wywnioskować po której iteracji dokładność wyniku staje się odpowiednia.



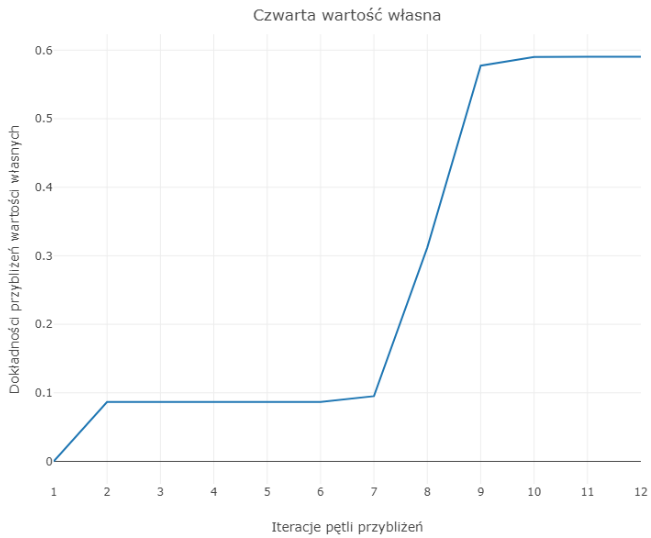
Rysunek 2: Pierwsza wartość własna



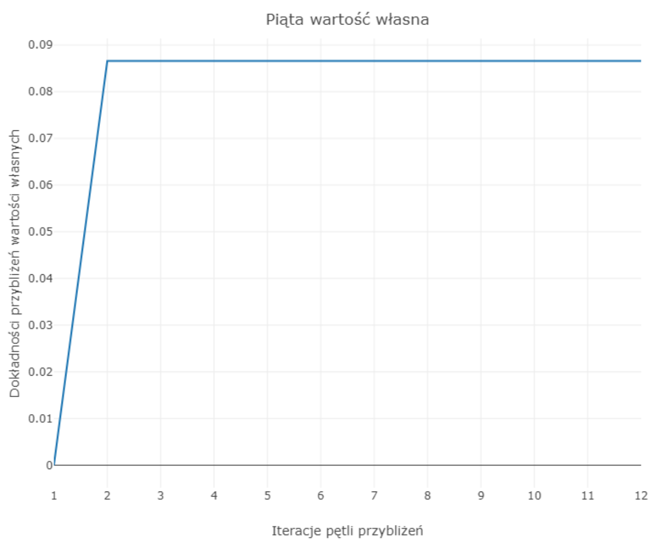
Rysunek 3: Druga wartość własna



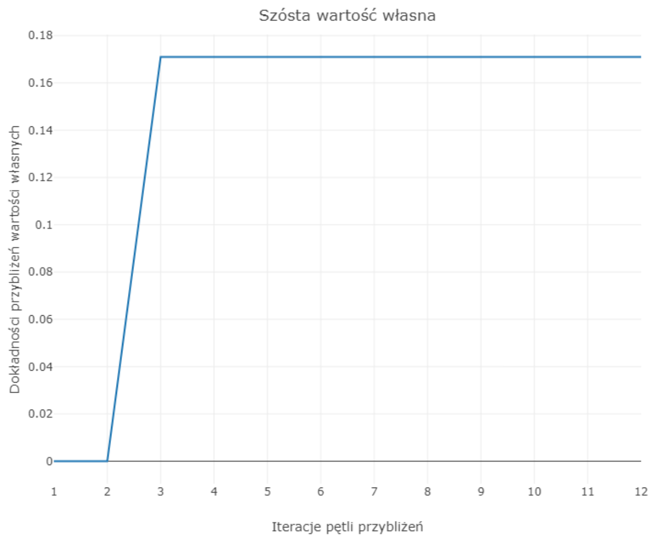
Rysunek 4: Trzecia wartość własna



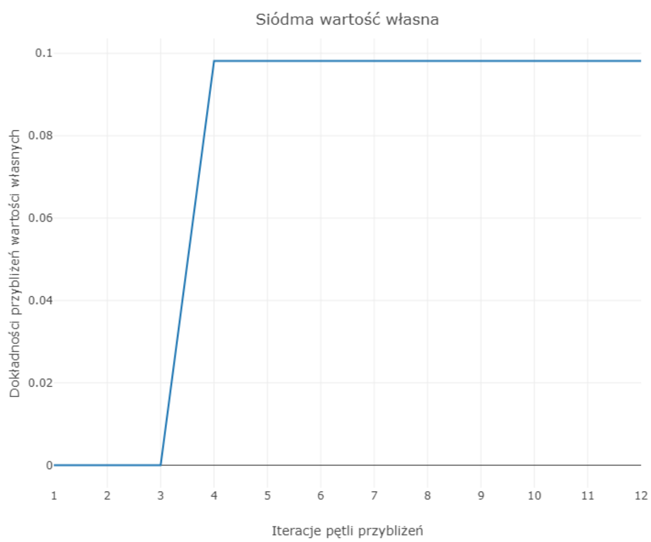
Rysunek 5: Czwarta wartość własna



Rysunek 6: Piąta wartość własna



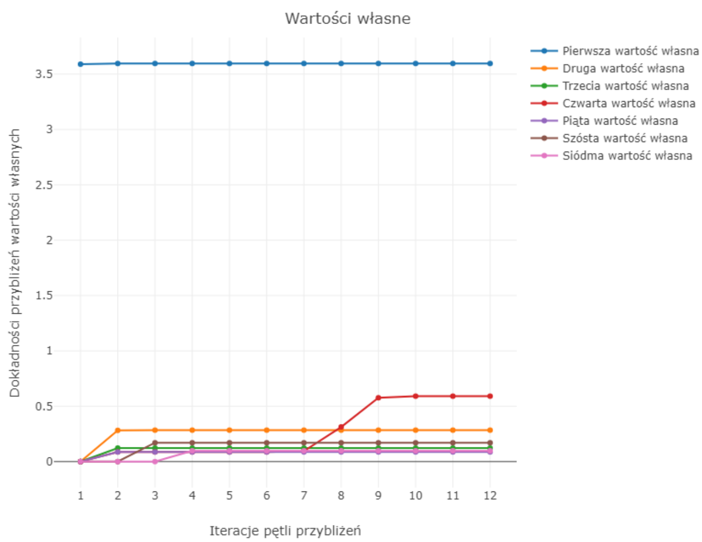
Rysunek 7: Szósta wartość własna



Rysunek 8: Siódma wartość własna

Jak możemy zauważyć, niektóre z wartości własnych tj. pierwsza, druga, trzecia, piąta wartość własna wyklarowała się już w drugiej iteracji. Najdłużej trwało obliczenie czwartej wartości własnej, bo aż dziewięć iteracji pętli. Program jako pierwszą wartość własną obliczył tą, która ma największą wartość bezwzględną, co jest zgodne z konwencją metody potęgowej.

Poniżej przedstawiam wykres zawierający wartości własne obliczane w poszczególnych iteracjach nałożone na jeden wykres:



Rysunek 9: Wartości własne w kolejnych iteracjach

Poniżej przedstawiam również macierz diagonalną D, obliczoną za pomocą macierzy wektorów własnych. Wyniki zostały zaokrąglone do 6 miejsc po przecinku, dla ułatwienia odczytu.

$$D = \begin{pmatrix} 3.595860 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.284988 & 0.000006 & 0 & 0 & 0 & 0 \\ 0 & 0.000006 & 0.122786 & 0.000001 & 0.000329 & 0 & 0 \\ 0 & 0 & 0.000001 & 0.590390 & 0.000297 & 0 & 0 \\ 0 & 0 & 0.000329 & 0.000297 & 0.086596 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.170974 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.098154 \end{pmatrix}$$

Jak możemy zauważyć, pomimo kilku niedokładności, macierz pozostała symetryczna.

6 Wnioski

Metoda potęgowa okazała się skutecznym narzędziem w znajdowaniu wartości własnych oraz odpowiadających im wektorom własnym dla macierzy symetrycznej. Demonstruje to zbieżność obliczeń do określonych wartości w limitowanym zestawie iteracji. Wyniki uzyskane podczas ćwiczenia wskazują na szybkie osiągnięcie poprawnej wartości dla większości wartości własnych. Szczególnie pierwsza, druga, trzecia i piąta wartość własna wykazały zbliżone rozwiązanie już w drugiej iteracji, co świadczy o wysokiej efektywności metody dla tych wartości. Ostateczna macierz D, zawierająca wartości własne na głównej przekątnej, została porównana z wartościami własnymi uzyskanymi metodą potęgową. Obserwowana korelacja pomiędzy tymi zestawami wyników potwierdza prawidłowość przeprowadzonych obliczeń. Przy dokładności numerycznej do 6 miejsc po przecinku, macierz diagonalna okazała się prawie idealna co do wartości oczekiwanych.

Potwierdziło się to, że metoda potęgowa to efektywna technika w znajdowaniu wartości własnych dla macierzy symetrycznych, co ma znaczenie w wielu zastosowaniach inżynierskich i matematycznych, gdzie takie obliczenia są niezbędne.