Metody numeryczne

Nikodem Kocjan

19.03.2024

Uogólniony (symetryczny) problem własny - wyznaczanie modułów własnych struny w 1D

Spis treści

1 Cel ćwiczenia

Celem ćwiczenia było zbadanie drgań własnych struny dla róznych rozkładów masy.

2 Wstęp teorytyczny

2.1 Opis równania

Częstość drgań własnych struny opisuje funkcja

$$\psi = \psi(x,t) \tag{1}$$

Dynamiką struny rządzi równanie falowe:

$$\frac{N}{\rho(x)}\frac{\partial^2\psi}{\partial x^2} = \frac{\partial^2\psi}{\partial t^2}. \tag{2}$$

gdzie
N - naciąg struny
$\rho(x)$ - liniowy rozkład gęstości

2.2 Przekształcenie równania

Dokonujemy separacji zmiennych, najpierw podstawiając $\psi(x,t) = u(x)\theta(t)$, a następnie dzieląc obustronnie przez iloczyn $u\theta$, co prowadzi do równania różniczkowego zależnego tylko od zmiennej położeniowej:

$$-\frac{d^2u}{dx^2} = \frac{\rho(x)}{N}\lambda u, \tag{3}$$

gdzie $\lambda = \omega^2$ a ω oznacza częstość własną drgań struny.

Struna jest przymocowana w punktach $\pm L/2$, gdzie L jest długością struny. Przechodząc do dyskretyzacji równania, wprowadzamy siatkę równoodległych węzłów, co pozwala na zastosowanie metody różnic skończonych i sprowadzenie problemu do postaci uogólnionego problemu własnego:

$$\mathbf{Au} = \lambda\mathbf{Bu}, \tag{4}$$

gdzie $\mathbf{A}$ i $\mathbf{B}$ są odpowiednio macierzami opisującymi system oraz rozkład gęstości struny.

2.3 Definicja macierzy

Podczas rozwiązywania problemu własnego, macierze A oraz B są zdefiniowane następująco

$$A_{ij} = \left(-\delta_{i,j+1} + 2\delta_{ij} - \delta_{i,j-1}\right)/\Delta x^2 \tag{5}$$

$$B_{i,j} = \frac{\rho_i}{N}\delta_{i,j} \tag{6}$$

gdzie

$$\delta_{i,j} = \begin{cases} 1, & \text{jeśli } i = j \\ 0, & \text{jeśli } i \neq j \end{cases} \tag{7}$$

jest deltą Kroneckera.

2.4 Biblioteka GSL

Do rozwiązania zadania wykorzystaliśmy bibliotękę GSL dla C++. Dzięki temu mogliśmy w bardzo prosty sposób tworzyć wektory, macierze, wyznaczać ich wartości własne oraz wektory własne. Poniżej przedstawiam kilka podstawowych funkcji z biblioteki GSL użytecznych w naszym programie.

```
\\Tworzy macierz A o wymiarach n na m
gsl_matrix* A = gsl_matrix_alloc(n, m);

\\Tworzy wektor wec o rozmiarze n
gsl_vector* wec = gsl_vector_alloc(n);

\\Ustawia wszystkie komorki macierzy na 0
gsl_matrix_set_zero(A);

\\Zwraca wartosc z komorki i wektora wec
gsl_vector_get(wec, i);

\\Zwalnia pamiec macierzy A
gsl_matrix_free(A);
```

Fragment kodu 1: Opis podstawowych funkcji GSL

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++
<fstream> - operacje na plikach
<gsl/gsl_eigen.h> - wyznaczanie wektorow i wartosci wlasnych
<gsl/gsl_matrix.h> - macierze

Zadeklarowane stałe, macierze oraz wektory użyte w programie to:

```
const double L = 10.0; // Dlugosc struny
const int n = 200; // Liczba wezlow
const double N = 1.0; // Stala N
const double deltaX = L / (n + 1.0); // Odleglosc miedzy wezlami
gsl_matrix* A = gsl_matrix_alloc(n, n); //macierz A
gsl_matrix* B = gsl_matrix_alloc(n, n); //macierz B
gsl_vector* eval = gsl_vector_alloc(n); //wektor wartosci
gsl_matrix* evec = gsl_matrix_alloc(n, n);
//macierz wektorow wlasnych
gsl_eigen_gensymmv_workspace* w = gsl_eigen_gensymmv_alloc(n);
//workspace potrzebny do wyznaczenia wektorow oraz wartosci
    wlasnych
```

Fragment kodu 2: Deklaracja stalych

4 Opis działania

4.1 Krótki opis programu

Program posiada główną pętlę, która zgodnie z poleceniem przyjmuje wartości α od 0 do 100. W pętli oprócz podstawowego kodu używam funkcji fillMatrices odpowiedzialnej za uzupełnianie macierzy A oraz B. Oprócz wymienionych macierzy, przyjmuje ona również aktualną wartość α w danej iteracji pętli.

```
void fillMatrices(gsl_matrix* A, gsl_matrix* B, double alpha)
```

Fragment kodu 3: fillMatrices

Druga funkcja, to funkcja rho, która zwraca wartość ρ zależną od x oraz α.

```
double rho(double x, double alpha)
```

Fragment kodu 4: rho

4.2 Opis głównej pętli

```
      for (double alpha = 0; alpha <= 100; alpha += 2) {
        //wyzerowanie macierzy
        gsl_matrix_set_zero(A);
        gsl_matrix_set_zero(B);

        //uzupelnienie macierzy A i B
        fillMatrices(A, B, alpha);

        //wyznaczenie wektorow i wartosci wlasnych
        gsl_eigen_gensymmv(A, B, eval, evec, w);

        //posortowanie wartosci i wektorow wlasnych
        gsl_eigen_gensymmv_sort(eval, evec, GSL_EIGEN_SORT_VAL_ASC)
    ;
        //zapis danych do pliku
        lambda = gsl_vector_get(eval, 0);
        result_file0 << sqrt(lambda) << " ," << std::endl;
        lambda = gsl_vector_get(eval, 1);
        result_file1 << sqrt(lambda) << " ," << std::endl;
        lambda = gsl_vector_get(eval, 2);
        result_file2 << sqrt(lambda) << " ," << std::endl;
        lambda = gsl_vector_get(eval, 3);
        result_file3 << sqrt(lambda) << " ," << std::endl;
        lambda = gsl_vector_get(eval, 4);
        result_file4 << sqrt(lambda) << " ," << std::endl;
        lambda = gsl_vector_get(eval, 5);
        result_file5 << sqrt(lambda) << " ," << std::endl;
        if (alpha == 0 || alpha == 100) {
            for (int mode = 0; mode < 6; ++mode) {
                std::ofstream mode_file("mode_" + std::to_string(
    mode+1) + "_alpha_" + std::to_string(int(alpha)) + ".txt");
                for (int i = 0; i < n; ++i) {
                    mode_file << gsl_matrix_get(evec, i, mode) << "
    , " << std::endl;
                }
            }
        }
    }
```

Fragment kodu 5: Główna pętla

W głównej pętli zaczynamy od wyzerowania macierzy A, oraz B. Następnie wywołując wspomnianą wcześniej funkcję fillMatrices, uzupełniamy te macierze.

Kolejnym krokiem jest wyznaczenie wartości własnych, oraz macierzy wektorów własnych za pomocą funkcji gsl_eigen_gensymmv zawartej w bibliotece GSL. Po otrzymaniu rezultatów i zapisaniu ich do wektora eval, oraz macierzy evec, sortujemy je za pomocą gsl_eigen_gensymmv_sort, która również zawiera się w GSL.

Pozostała część pętli odpowiada za zapisanie danych do plików zgodnie z wymaganiami zadania.

4.3 Funkcja fillMatrices

```
void fillMatrices(gsl_matrix* A, gsl_matrix* B, double alpha) {
    for (int i = 0; i < n; ++i) {
        //wyznaczamy x dla kolejnej iteracji
        double x_i = -L / 2.0 + deltaX * (i + 1);
        //wyznaczamy rho za pomoca funkcji rho
        //zalezne od x oraz alpha
        double rho_i = rho(x_i, alpha);
        //uzupelniamy macierze A oraz B
        //zgodnie z wczesniej opisanymi wzorami
        gsl_matrix_set(A, i, i, 2.0 / (deltaX * deltaX));
        gsl_matrix_set(B, i, i, rho_i / N);
        if (i > 0) {
            gsl_matrix_set(A, i, i - 1, -1.0 / (deltaX * deltaX));
            gsl_matrix_set(A, i - 1, i, -1.0 / (deltaX * deltaX));
        }
    }
}
```

Fragment kodu 6: void fillMatrices

Funkcja nic nie zwraca, ponieważ nadpisuje ona zawartość macierzy A oraz B.

4.4 Funkcja rho

```
double rho(double x, double alpha) {
    //obliczenie rho dla danego alpha oraz x
    return 1.0 + 4.0 * alpha * x * x;
}
```

Fragment kodu 7: double rho

Funkcja zwraca wartość rho, co jest wykorzystywane w funkcji fillMatricies.

4.5 Zakończenie

Po wyjściu z pętli program czyści pamięć, która została wcześniej zaalokowana na wymagane dane

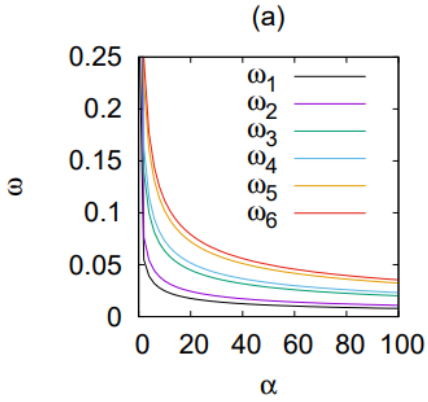
```
    gsl_matrix_free(A);
    gsl_matrix_free(B);
    gsl_vector_free(eval);
    gsl_matrix_free(evec);
    gsl_eigen_gensymmv_free(w);
```

Fragment kodu 8: Zwolnienie pamięci

5 Analiza wyników

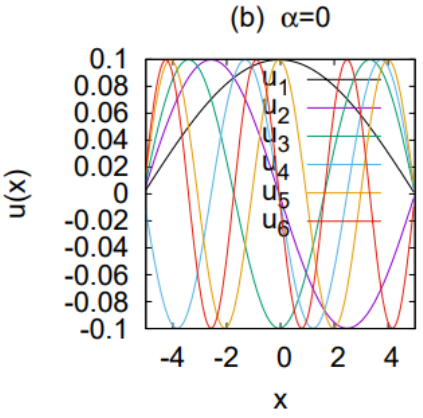
5.1 Krótki opis

Analizę dokonamy dla trzech różnych opcji. Jako pierwsze zbadamy wartości pierwiastków z 6 kolejnych najmniejszych wartości własnych zależnych od α. Poniżej przedstawiam oczekiwany rezultat.



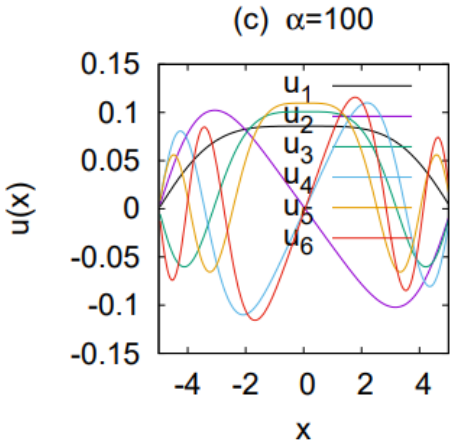
Rysunek 1: Wartości własne

Druga opcja to analiza wektorów własnych odpowiadających 6 najniższym wartością własnym. Badania dokonamy dla $\alpha = 0$. Poniżej znjaduje się oczekiwany rezultat.



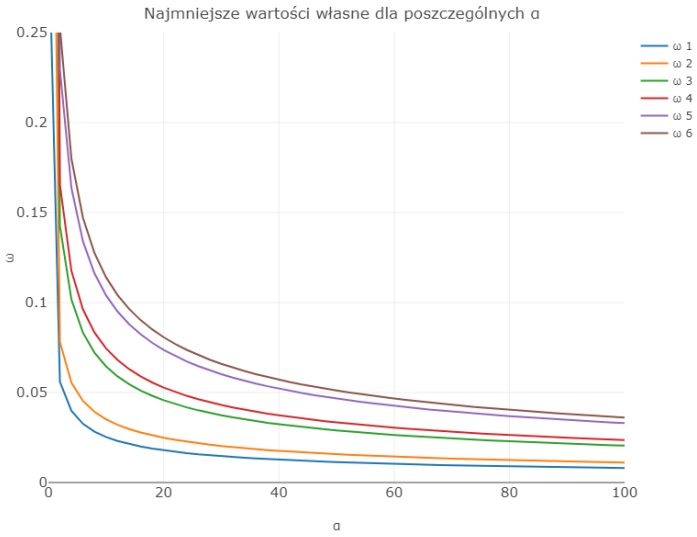
Rysunek 2: Wektory własne $\alpha = 0$

Jako ostatnie zrobimy analogiczne badanie jak w podpunkcie 2, lecz dla $\alpha = 100$. Poniżej oczekiwany wynik.



Rysunek 3: Wektory własne $\alpha = 100$

5.2 Wartości własne



Rysunek 4: Wartości własne

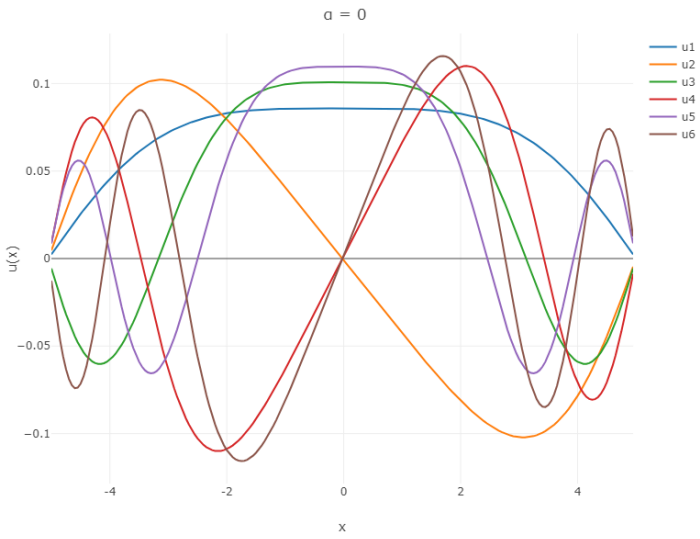
Powyższy wykres jest bardzo zbliżony do wykresu oczekiwanego. Dla lepszej czytelności zmniejszyłem zakres osi Y do 0,25. Warto zwrócić uwagę, że najmniejsze pierwiastki wartości własnych dla $\alpha = 0$ wynosiły od najmniejszej:
0.314156
0.628293
0.942391
1.25643
1.5704
1.88426

5.3 Wektory własne $\alpha = 0$

Rysunek 5: Wektory własne $\alpha = 0$

Wykres dokładnie pokrywa wartości oczekiwane.

5.4 Wektory własne $\alpha = 100$



Rysunek 6: Wektory własne $\alpha = 0$

Wykres dokładnie pokrywa wartości oczekiwane.

6 Wnioski

Przy użyciu biblioteki GSL, udało się rozwiązać uogólniony problem własny, co pozwoliło na szczegółową analizę wartości i wektorów własnych. Jest to z pewnością bardzo użyteczna biblioteka, ułatwiająca pracę nad badaniem różnych rozwiązań metodami numerycznymi.

Zadanie ukzało zależność częstości drgań struny (wartości własnych) oraz ich modułów (wektorów własnych) od wartości parametru α.

Zauważamy, że dla $\alpha = 0$ wektory własne są nieparzystymi wielokrotnościami połówek sinusa, natomiast dla $\alpha = 100$, wektory 1,3,5 mają płaską centralną część. Jest to spowodowane tym, że środek struny staje się masywny.

Powyższe wnioski potwierdzają, że parametr α ma kluczowe znaczenie w symulacji różnych warunków fizycznych, rozkład masy powoduje duże zmiany w drganiach własnych struny.