Macierzowy (niesymetryczny) problem własny - wyznaczanie modów własnych niejednorodnej struny w 1D

Tomasz Chwiej

20 marca 2018

1 Wprowadzenie

Na laboratorium zajmiemy się wyznaczaniem częstości drgań własnych struny, której wychylenie w czasie i przestrzeni opisuje funkcja $\psi = \psi(x,t)$. Dynamiką struny rządzi równanie falowe (N-naciąg struny, $\rho(x)$- liniowy rozkład gęstości):

$$\frac{N}{\rho(x)}\frac{\partial^2\psi}{\partial x^2} = \frac{\partial^2\psi}{\partial t^2} \tag{1}$$

Dokonujemy separacji zmiennych: i) najpierw podstawiając $\psi(x,t) = u(x)\theta(t)$, a następnie ii) dzieląc przez iloczyn $u\theta$

$$\frac{N}{\rho(x)}\frac{1}{u}\frac{\partial^2 u}{\partial x^2} = \frac{1}{\theta}\frac{\partial^2\theta}{\partial t^2} = const = -\lambda \quad (\lambda = \omega^2,\, \omega - \text{częstość własna drgan}) \tag{2}$$

dzięki czemu otrzymujemy równanie różniczkowe zależne tylko od zmiennej położeniowej

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

Struna przymocowana jest w punktach $\pm L/2$ (L-długość struny). Wprowadzamy siatkę równoodległych węzłów: $x = x_i$, $u(x) = u_i$, $\rho(x) = \rho_i$ Odległość pomiędzy węzłami wynosi

$$\Delta x = \frac{L}{n+1} \tag{4}$$

a położenie w przestrzeni wyznaczamy tak

$$x_i = -\frac{L}{2} + \Delta x \cdot (i+1), \quad i = 0,1,2,\ldots,n-1 \tag{5}$$

Teraz możemy dokonać dyskretyzacji równania (3) podstawiając trójpunktowy iloraz różnicowy centralny za drugą pochodną

$$-\frac{N}{\rho_i}\frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} = \lambda u_i \tag{6}$$

co można zapisać w postaci (A - macierz, $\boldsymbol{u}$ - wektor)

$$A\boldsymbol{u} = \lambda\boldsymbol{u} \tag{7}$$

czyli jako problem własny, w którym elementy macierzowe są zdefiniowane następująco

$$A_{i,j} = (-\delta_{i,j+1} + 2\delta_{i,j} - \delta_{i,j-1}) \cdot N/(\rho_i \Delta x^2) \tag{8}$$

gdzie

$$\delta_{i,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \tag{9}$$

jest deltą Kroneckera.

2 Zadania do wykonania

1. Używamy biblioteki GSL, więc konieczne jest dołączenie pliku nagłówkowego

   ```
   #include</usr/include/gsl/gsl_eigen.h>
   ```

2. Przyjmujemy następujące parametry: $L = 10$, $n = 200$, $\rho(x) = 1 + 4\,\alpha\,x^2$, $N = 1$.
3. Utworzyć macierz A i wypełnić ją zgodnie z wzorem (8).
4. Rozwiązać równanie (7) dla $\alpha \in [0, 100]$ z krokiem $\Delta\alpha = 2$. Dla każdej wartości parametru α do pliku zapisać wartości pierwiastków z 6 kolejnych najmniejszych wartości własnych i sporządzić odpowiedni wykres ($\omega = \sqrt{\lambda} = f(\alpha)$).
5. Dla $\alpha = 0$ oraz $\alpha = 100$ zapisać do pliku wektory własne odpowiadające 6 najniższym wartościom własnym i sporządzić ich wykresy.

3 Uwagi

1. Dla każdej wartości α macierz A zawsze wypełniamy zgodnie z wzorem (8) , łącznie z zerami (eliminujemy w ten sposób śmieci numeryczne z poporzednich przebiegów pętli).
2. Macierz A jest niesymetryczna, więc do rozwiązania problemu własnego używamy metody GSL-a

   ```
   int gsl_eigen_nonsymmv(gsl_matrix * A,
                          gsl_vector_complex * eval,
                          gsl_matrix_complex * evec,
                          gsl_eigen_nonsymmv_workspace *w);
   ```

   gdzie: *eval* to zespolony wektor wartości własnych (nieposortowany)

   ```
   gsl_vector_complex *eval=gsl_vector_complex_calloc(n);
   ```

   evec to zespolona macierz $n \times n$ w której kolumnach zapisane są prawostronne wektory własne

   ```
   gsl_matrix_complex *evec=gsl_matrix_complex_calloc(n,n);
   ```

   a wektor pomocniczy w definiujemy tak

   ```
   gsl_eigen_nonsymmv_workspace *w = gsl_eigen_nonsymmv_alloc(n);
   ```

3. Procedura zwraca wektory i wartości własne w postaci liczb zespolonych. Nas interesują ich części rzeczywiste (nasz problem jest rzeczywisty i rozwiązania też takie powinny być). W programie umieszczamy nagłówki

```
#include</usr/include/gsl/gsl_complex.h>
#include</usr/include/gsl/gsl_complex_math.h>
```

aby móc korzystać z operacji na liczbach zespolonych. Część rzeczywistą liczby zespolonej z pobieramy tak

```
double x = GSL_REAL(z);
```

a element macierzy

```
double x = gsl_matrix_complex_get(evec,i,j);
```

i wektora

```
double x = gsl_vector_complex_get(eval,i);
```

4. Wartości i wektory własne sortujemy (po rozwiązaniu problemu) stosując funkcję GSL-a

```
int gsl_eigen_nonsymmv_sort(gsl_vector * eval,
                            gsl_matrix_complex * evec,
                            gsl_eigen_sort_t sort_type);
```

gdzie podstawiamy

```
gsl_eigen_sort_t = GSL_EIGEN_SORT_ABS_ASC
```

(sortowanie od najmniejszej do największej wartości własnej)