Metody numeryczne

Wojciech Chrobak

15 listopada 2017

Zadanie 1

$$\begin{bmatrix} d_1 & e_1 & 0 & \cdots & 0 \\ e_1 & d_2 & e_2 & \ddots & \vdots \\ 0 & e_2 & d_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & e_{n-1} \\ 0 & \cdots & 0 & e_{n-1} & d_n \end{bmatrix}$$

Struktura macierzy ma następującą postać:

- elementy na diagonali to wektor d o długości równej N (rozmiar macierzy)
- elementy pod i nad diagonalą to wektor e (macierz symetryczna) o długości N-1

Dla tego typu macierzy najlepszą metodą rozwiązywania będzie zastosowanie obrotów Givensa aby otrzymać faktoryzacje QR. Robimy to w czasie liniowym O(N). Aby nie przechowywać w pamięci dużej ilości 0, macierz A możemy przedstawić jako 3 wektory. D na diagonali i E pod/nad diagonalą. W każdym kolejnym kroku pętli obliczamy wartości macierzy G i działamy nią na macierz A a także na wektor wyrazów wolnych. W efekcie dostajemy macierz R i zmodyfikowany wektor wyrazów wolnych. Stosujemy metodę back substitution i otrzymujemy wyniki.

Kod

```
#include <iostream>
#include <cmath>
#include <vector>
#include <iomanip>

using namespace std;

void printG(vector<vector<double>> G) {
  cout << endl;
  for (int i = 0; i < G.size(); i++) {
    for (int j = 0; j < G.size(); j++) {
      cout << "[ " << fixed << setprecision(6) << showpos << G[i][j] << " ]   ";
    }
    cout << endl;
  }
  cout << endl;
}

void printA(vector<double> A) {
  for (int i = 0; i < A.size(); i++) {
    cout << "[ " << showpoint << A[i] << " ]   ";
```

```
  }
  cout << endl;
}

int main() {
  const int sizeMatrix = 7;

  vector<double> d_vector;
  d_vector.assign(sizeMatrix, 4);

  // wektor pod diagonala
  vector<double> e1_vector;
  e1_vector.assign(sizeMatrix - 1, 1);

  // wektor nad diagonala
  vector<double> e2_vector;
  e2_vector.assign(sizeMatrix - 1, 1);

  // wyrazy wolne
  vector<double> right = {1, 2, 3, 4, 5, 6, 7};

  // wynik
  vector<double> x;
  x.assign(sizeMatrix, 0);

  // wektor drugi nad diagonala
  vector<double> f_vector;
  f_vector.assign(sizeMatrix - 2, 0);


  cout << "Ogolna postac macierzy: " << endl;
  for (int i = 0; i < sizeMatrix; i++) {
    for (int j = 0; j < sizeMatrix; j++) {
      if (i == j)
        cout << " d  ";
      else if (j == i + 1)
        cout << "e2  ";
      else if (j == i - 1)
        cout << "e1  ";
      else
        cout << " 0  ";
    }
    cout << endl;
  }
  cout << endl;
  cout << "wektor D  = ";
  printA(d_vector);
  cout << "wektor E1 = ";
  printA(e1_vector);
  cout << "wektor E2 = ";
  printA(e2_vector);
  cout << endl;
  vector<vector<double>> G = {{1, 1},
                              {1, 1}};

  for (int i = 0; i < d_vector.size() - 1; i++) {
    // b element zerowany, a element nad nim
    double a = d_vector[i];
    double b = e1_vector[i];

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    double cosX = a / sqrt(a * a + b * b);
    double sinX = -b / sqrt(a * a + b * b);
```

```
      cout << "cosX = " << cosX << endl;
      cout << "sinX = " << sinX << endl;
      G[0][0] = cosX;
      G[0][1] = -sinX;
      G[1][0] = sinX;
      G[1][1] = cosX;
      printG(G);

      vector<double> temp_a_diag = d_vector;
      vector<double> temp_a_pod = e1_vector;
      vector<double> temp_a_nad = e2_vector;
      vector<double> temp_a_nad2 = f_vector;

      // stosujemy obroty Givensa
      d_vector[i] = G[0][0] * temp_a_diag[i] + G[0][1] * temp_a_pod[i];
      d_vector[i + 1] = G[1][0] * temp_a_nad[i] + G[1][1] * temp_a_diag[i + 1];

      e2_vector[i] = G[0][0] * temp_a_nad[i] + G[0][1] * temp_a_diag[i + 1];
      e2_vector[i + 1] = G[1][0] * temp_a_nad2[i] + G[1][1] * temp_a_nad[i + 1];

      f_vector[i] = G[0][0] * temp_a_nad2[i] + G[0][1] * temp_a_nad[i+1];

      e1_vector[i] = 0;

      cout << "wektor D  = ";
      printA(d_vector);
      cout << "wektor E1 = ";
      printA(e1_vector);
      cout << "wektor E2 = ";
      printA(e2_vector);
      cout << "wektor F  = ";
      printA(f_vector);

      // dzialamy macierza g na wektor wyrazow wolnych
      vector<double> temp_right = right;

      right[i] = G[0][0] * temp_right[i] + G[0][1] * temp_right[i + 1];
      right[i + 1] = G[1][0] * temp_right[i] + G[1][1] * temp_right[i + 1];


      cout << endl << endl << "--------------------------------------------------------" <<
     endl;
    }

    // rozwiazujemy Ax=b metoda backSubs
    cout << "wektor B  = ";
    printA(right);
    for (int r = d_vector.size() - 1; r >= 0; r--) {
      double val = 0;
      val = e2_vector[r] * x[r+1] + f_vector[r] * x[r+2];

      val = right[r] - val;
      x[r] = val / d_vector[r];
    }

    cout << "x  = ";
    printA(x);
    return 0;
  }
```

Działanie programu

$$A_0 = \begin{bmatrix} 4 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ & & & 1 & 4 & 1 & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 4 \end{bmatrix}, b_0 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 4.12311 & 1.94029 & 0.242536 & & & & \\ & 3.63803 & 0.970143 & & & & \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ & & & 1 & 4 & 1 & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 4 \end{bmatrix}, b_1 = \begin{bmatrix} 1.45521 \\ 1.69775 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 4.12311 & 1.94029 & 0.242536 & & & & \\ & 3.77297 & 1.99562 & 0.265043 & & & \\ & & 3.59982 & 0.964237 & & & \\ & & 1 & 4 & 1 & & \\ & & & 1 & 4 & 1 & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 4 \end{bmatrix}, b_2 = \begin{bmatrix} 1.45521 \\ 2.43216 \\ 2.44273 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 4.12311 & 1.94029 & 0.242536 & & & & \\ & 3.77297 & 1.99562 & 0.265043 & & & \\ & & 3.73613 & 1.99968 & 0.267657 & & \\ & & & 3.59597 & 0.963514 & & \\ & & & 1 & 4 & 1 & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 4 \end{bmatrix}, b_3 = \begin{bmatrix} 1.45521 \\ 2.43216 \\ 3.42423 \\ 3.20024 \\ 5 \\ 6 \\ 7 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 4.12311 & 1.94029 & 0.242536 & & & & \\ & 3.77297 & 1.99562 & 0.265043 & & & \\ & & 3.73613 & 1.99968 & 0.267657 & & \\ & & & 3.73243 & 1.99998 & 0.267922 & \\ & & & & 3.59562 & 0.963441 & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 4 \end{bmatrix}, b_4 = \begin{bmatrix} 1.45521 \\ 2.43216 \\ 3.42423 \\ 4.42286 \\ 3.95979 \\ 6 \\ 7 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} 4.12311 & 1.94029 & 0.242536 & & & & \\ & 3.77297 & 1.99562 & 0.265043 & & & \\ & & 3.73613 & 1.99968 & 0.267657 & & \\ & & & 3.73243 & 1.99998 & 0.267922 & \\ & & & & 3.73208 & 2.00000 & 0.267947 \\ & & & & & 3.59558 & 0.963434 \\ & & & & & 1 & 4 \end{bmatrix}, b_5 = \begin{bmatrix} 1.45521 \\ 2.43216 \\ 3.42423 \\ 4.42286 \\ 5.42267 \\ 4.71959 \\ 7 \end{bmatrix}$$

$$A_6 = \begin{bmatrix} 4.12311 & 1.94029 & 0.242536 & & & & \\ & 3.77297 & 1.99562 & 0.265043 & & & \\ & & 3.73613 & 1.99968 & 0.267657 & & \\ & & & 3.73243 & 1.99998 & 0.267922 & \\ & & & & 3.73208 & 2.00000 & 0.267947 \\ & & & & & 3.73205 & 2.00000 \\ & & & & & & 3.59558 \end{bmatrix}, b_6 = \begin{bmatrix} 1.45521 \\ 2.43216 \\ 3.42423 \\ 4.42286 \\ 5.42267 \\ 6.42265 \\ 5.47942 \end{bmatrix}$$

Wynik końcowy

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 0.166789 \\ 0.332842 \\ 0.501841 \\ 0.659794 \\ 0.858984 \\ 0.904271 \\ 1.52393 \end{bmatrix}$$