Metody numeryczne

Wojciech Chrobak

15 listopada 2017

Zadanie 1 - obowiązkowe

Wzór Shermana-Morrisona

$$A_1^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}$$

Szukamy takich x, że:

$$x = A_1^{-1}b = (A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u})b$$

$$z = A^{-1}b$$

$$q = A^{-1}u$$

- Rozwiązujemy równanie:

$$Az = b$$

- Rozwiązujemy równanie:

$$Aq = u$$

- Obliczamy x:

$$x = z - \frac{v^Tz}{1+v^Tq}q$$

$$u = v = \begin{bmatrix} u_1 & 0 & \cdots & 0 & u_n \end{bmatrix}^T$$

$$uv^T = \begin{bmatrix} u_1v_1 & 0 & \cdots & 0 & u_1v_n \\ 0 & & & & 0 \\ \vdots & & & & \vdots \\ 0 & & & & 0 \\ u_nv_1 & 0 & \cdots & 0 & u_nv_n \end{bmatrix}$$

$$A_1 = A + uv^T$$

$$A_1 = \begin{bmatrix} d_1+u_1v_1 & e_1 & 0 & \cdots & u_1v_n \\ e_1 & d_2 & e_2 & \ddots & \vdots \\ 0 & e_2 & d_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & e_{n-1} \\ u_nv_1 & \cdots & 0 & e_{n-1} & d_n+u_nv_n \end{bmatrix}$$

$$A = \begin{bmatrix} d_1 - u_1 v_1 & e_1 & 0 & \cdots & 0 \\ e_1 & d_2 & e_2 & \ddots & \vdots \\ 0 & e_2 & d_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & e_{n-1} \\ 0 & \cdots & 0 & e_{n-1} & d_n - u_n v_n \end{bmatrix}$$

Powstała macierz A 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;
}

double multiVectorVector(vector<double> a, vector<double> b) {
  double sum = 0;
  for(int i = 0; i < a.size(); i++) {
    sum = sum + a[i] * b[i];
  }

  return sum;
}


int main() {
```

```
const int sizeMatrix = 7;
vector<double> u_vector = {1, 0, 0, 0, 0, 0, 1};
vector<double> v_vector = {1, 0, 0, 0, 0, 0, 1};

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

// A1 = A + uvT
d_vector[0] -= u_vector[0];
d_vector[sizeMatrix-1] -= v_vector[sizeMatrix-1];


// 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};

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

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

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];

       // dzialamy macierza g na wektor u
      vector<double> temp_u = u_vector;

      u_vector[i] = G[0][0] * temp_u[i] + G[0][1] * temp_u[i + 1];
      u_vector[i + 1] = G[1][0] * temp_u[i] + G[1][1] * temp_u[i + 1];


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


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

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

    cout << "z = ";
    printA(z);

    // rozwiazujemy Aq=u metoda backSubs
    cout << "u  = ";
    printA(u_vector);
    for (int r = d_vector.size() - 1; r >= 0; r--) {
      double val = 0;
```

```
      val = e2_vector[r] * q[r+1] + f_vector[r] * q[r+2];

      val = u_vector[r] - val;
      q[r] = val / d_vector[r];
    }

    cout << "q = ";
    printA(q);

    double l = multiVectorVector(u_vector,z);
    double m = 1 + multiVectorVector(u_vector,q);
    double temp = l/m;

    vector<double> temp2;
    temp2.assign(sizeMatrix, 0);

    for(int i = 0; i < sizeMatrix; i++) {
      temp2[i] = temp * q[i];
    }

    for(int i = 0; i < sizeMatrix; i++) {
      x[i] = z[i] - temp2[i];
    }

    cout << "x = ";
    printA(x);

    return 0;
  }
```

Działanie programu

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

$$u = v = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T$$

$$A_1 = \begin{bmatrix} 3.16228 & 2.21359 & 0.316228 & & & & \\ & 3.47851 & 0.948683 & & & & \\ & & 4 & 1 & & & \\ & & & 4 & 1 & & \\ & & & & 4 & 1 & \\ & & & & & 4 & 1 \\ & & & & & & 3 \end{bmatrix}, b_1 = \begin{bmatrix} 1.58114 \\ 1.58114 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 3.16228 & 2.21359 & 0.316228 & & & & \\ & 3.61939 & 2.01691 & 0.276289 & & & \\ & & 3.58219 & 0.961074 & & & \\ & & & 4 & 1 & & \\ & & & & 4 & 1 & \\ & & & & & 4 & 1 \\ & & & & & & 3 \end{bmatrix}, b_2 = \begin{bmatrix} 1.58114 \\ 2.34846 \\ 2.44637 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 3.16228 & 2.21359 & 0.316228 & & & & \\ & 3.61939 & 2.01691 & 0.276289 & & & \\ & & 3.71915 & 2.00120 & 0.268879 & & \\ & & & 3.59428 & 0.963174 & & \\ & & & & 4 & 1 & \\ & & & & & 4 & 1 \\ & & & & & & 3 \end{bmatrix}, b_3 = \begin{bmatrix} 1.58114 \\ 2.34846 \\ 3.43180 \\ 3.19492 \\ 5 \\ 6 \\ 7 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 3.16228 & 2.21359 & 0.316228 & & & & \\ & 3.61939 & 2.01691 & 0.276289 & & & \\ & & 3.71915 & 2.00120 & 0.268879 & & \\ & & & 3.73080 & 2.00009 & 0.268039 & \\ & & & & 3.59546 & 0.963408 & \\ & & & & & 4 & 1 \\ & & & & & & 3 \end{bmatrix}, b_4 = \begin{bmatrix} 1.58114 \\ 2.34846 \\ 3.43180 \\ 4.41821 \\ 3.96068 \\ 6 \\ 7 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} 3.16228 & 2.21359 & 0.316228 & & & & \\ & 3.61939 & 2.01691 & 0.276289 & & & \\ & & 3.71915 & 2.00120 & 0.268879 & & \\ & & & 3.73080 & 2.00009 & 0.268039 & \\ & & & & 3.73194 & 2.00001 & 0.267957 \\ & & & & & 3.59557 & 0.963431 \\ & & & & & & 3 \end{bmatrix}, b_5 = \begin{bmatrix} 1.58114 \\ 2.34846 \\ 3.43180 \\ 4.41821 \\ 5.42358 \\ 4.71929 \\ 7 \end{bmatrix}$$

$$A_6 = \begin{bmatrix} 3.16228 & 2.21359 & 0.316228 & & & & \\ & 3.61939 & 2.01691 & 0.276289 & & & \\ & & 3.71915 & 2.00120 & 0.268879 & & \\ & & & 3.73080 & 2.00009 & 0.268039 & \\ & & & & 3.73194 & 2.00001 & 0.267957 \\ & & & & & 3.73204 & 1.73205 \\ & & & & & & 2.63215 \end{bmatrix}, b_6 = \begin{bmatrix} 1.58114 \\ 2.34846 \\ 3.43180 \\ 4.41821 \\ 5.42358 \\ 6.42237 \\ 5.47950 \end{bmatrix}$$

Wynik końcowy

$$u = \begin{bmatrix} 0.948683 \\ -0.303918 \\ 0.084153 \\ -0.022632 \\ 0.006067 \\ 0.266324 \\ 0.963885 \end{bmatrix}, z = \begin{bmatrix} 0.228100 \\ 0.315699 \\ 0.509103 \\ 0.647887 \\ 0.899347 \\ 0.754723 \\ 2.081759 \end{bmatrix}, q = \begin{bmatrix} 0.366197 \\ -0.098592 \\ 0.028169 \\ -0.014085 \\ 0.028169 \\ -0.098592 \\ 0.366197 \end{bmatrix}, x = \begin{bmatrix} -0.260163 \\ 0.447154 \\ 0.471545 \\ 0.666667 \\ 0.861789 \\ 0.886179 \\ 1.593496 \end{bmatrix}$$