Metody numeryczne

Wojciech Chrobak

26 listopada 2017

Zadanie 3

Metoda potęgowa

Macierz $A^{N\times N}$ musi być symetryczna.
Aby znaleźć największą co do modułu wartość własną macierzy A należy przeprowadzić iteracje z założeniem że wektor startowy $||y_1|| = 1$:

$$Ay_k = z_k$$
$$y_{k+1} = \frac{z_k}{||z_k||}$$

Szukaną wartością własną jest $||z_k||$ a wektor y_{k+1} jest odpowiadającym jej unormowanym wektorem własnym.

Aby znaleźć drugą co do modułu największą wartość własną macierzy A należy przeprowadzić iteracje z założeniem, że wektor startowy $||y_1|| = 1$ i $e_1^T y_k = 0$:

$$Ay_k = z_k$$
$$z_k = z_k - e_1(e_1^T z_k)$$
$$y_{k+1} = \frac{z_k}{||z_k||}$$

Gdzie e_1 jest wektorem własnym (wyznaczony wyżej), który tworzy bazę ortonormalną w $\mathbb{R}^N$. Drugi krok jest reortogonalizacją wektora z_k.

Szukaną wartością własną jest $||z_k||$ a wektor y_{k+1} jest odpowiadającym jej unormowanym wektorem własnym.

Odwrotna metoda potęgowa

Macierz $A^{N\times N}$ musi być symetryczna.
Najmniejsza wartość własna jest największą wartością własną macierzy odwrotnej. Prowdzimy więc iterację z założeniem, że wektor startowy $||y_1|| = 1$:

$$A^{-1}y_k = z_k \Longrightarrow Az_k = y_k$$
$$y_{k+1} = \frac{z_k}{||z_k||}$$

Szukaną wartością własną jest $||z_k||^{-1}$ a wektor y_{k+1} jest odpowiadającym jej unormowanym wektorem własnym.

Podobnie szukamy drugiej, co do modułu najmniejszej wartości własnej. Założenia są takie same jak przy szukaniu największych wartości, czyli $||y_1|| = 1$ i $e_1^T y_k = 0$:

$$A^{-1}y_k = z_k \Longrightarrow Az_k = y_k$$

$$z_k = z_k - e_1(e_1^T z_k)$$
$$y_{k+1} = \frac{z_k}{||z_k||}$$

Szukaną wartością własną jest $||z_k||^{-1}$ a wektor y_{k+1} jest odpowiadającym jej unormowanym wektorem własnym.

Kolejne równania $Az_k = y_k$ rozwiązujemy korzystając z dokonanego tylko raz rozkładu LU (nie możemy użyć rozkładu Cholesky'ego ponieważ macierze te nie są dodatnio określone).

Kod

```
#include <iostream>
#include <vector>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_linalg.h>

using namespace std;

void powerMethodMin(gsl_matrix *M) {
    int size = M->size1;

    gsl_vector *y, *z, *temp;
    gsl_permutation *p;
    int s;

    y = gsl_vector_alloc(size);
    temp = gsl_vector_alloc(size);

    gsl_vector_set_zero(y);
    // aby || y || = 1
    gsl_vector_set(y, 0, 1);


    z = gsl_vector_alloc(size);
    p = gsl_permutation_alloc(size);

    // LU
    gsl_linalg_LU_decomp(M, p, &s);

    while (true) {
        // poprzednia norma aby porownac i zakonczyc w odpowiednim momencie
        double z_prev_norm = gsl_blas_dnrm2(z);

        // rozwiazywanie Az = y
        gsl_linalg_LU_solve(M, p, y, z);

        // wyliczanie kolejnego y
        double norm = gsl_blas_dnrm2(z);
        gsl_vector_memcpy(temp, z);
        gsl_vector_scale(temp, 1 / norm);
        gsl_vector_memcpy(y, temp);


        if (abs(norm - z_prev_norm) < 10e-16)
            break;
    }
    cout << 1 / gsl_blas_dnrm2(z) << " dla:" << endl;
    gsl_vector_fprintf(stdout, y, "%f");
    cout << endl;
```

```
50
51
52      // wyznaczanie wektora prostopadlego do wektora e1
53      gsl_vector *e1;
54      e1 = gsl_vector_alloc(size);
55      gsl_vector_memcpy(e1, y);
56      gsl_vector_set_all(y, 1);
57      double sum = gsl_blas_dasum(e1) - gsl_vector_get(e1, size - 1);
58      double buf = -(sum / gsl_vector_get(e1, size - 1));
59      gsl_vector_set(y, size - 1, buf);
60      double normY = gsl_blas_dnrm2(y);
61      gsl_blas_dscal(1 / normY, y);
62
63
64      while (true) {
65          // poprzednia norma aby porownac i zakonczyc w odpowiednim momencie
66          double z_prev_norm = gsl_blas_dnrm2(z);
67
68          // rozwiazywanie Az = y
69          gsl_linalg_LU_solve(M, p, y, z);
70
71
72          // reortogonalizacja
73          double data = 0;
74          double *e1z = &data;
75          gsl_blas_ddot(e1, z, e1z);
76          gsl_vector *e1_e1z;
77          e1_e1z = gsl_vector_alloc(size);
78          gsl_vector_memcpy(e1_e1z, e1);
79          gsl_blas_dscal(*e1z, e1_e1z);
80          gsl_vector_sub(z, e1_e1z);
81
82          // wyliczanie kolejnego y
83          double norm = gsl_blas_dnrm2(z);
84          gsl_vector_memcpy(temp, z);
85          gsl_vector_scale(temp, 1 / norm);
86          gsl_vector_memcpy(y, temp);
87
88
89          if (abs(norm - z_prev_norm) < 10e-16)
90              break;
91      }
92      cout << 1 / gsl_blas_dnrm2(z) << " dla:" << endl;
93      gsl_vector_fprintf(stdout, y, "%f");
94  }
95
96
97  void powerMethodMax(gsl_matrix *M) {
98      int size = M->size1;
99
100     gsl_vector *y, *z, *temp;
101
102     y = gsl_vector_alloc(size);
103     temp = gsl_vector_alloc(size);
104     gsl_vector_set_zero(y);
105     gsl_vector_set(y, 0, 1);
106
107
108     z = gsl_vector_alloc(size);
109
110
111     while (true) {
112         // poprzednia norma aby porownac i zakonczyc w odpowiednim momencie
113         double z_prev_norm = gsl_blas_dnrm2(z);
```

```
114
115         // wyliczanie Ay = z
116         gsl_blas_dgemv(CblasNoTrans, 1.0, M, y, 0.0, z);
117
118         // wyliczanie kolejnego y
119         double norm = gsl_blas_dnrm2(z);
120         gsl_vector_memcpy(temp, z);
121         gsl_vector_scale(temp, 1 / norm);
122         gsl_vector_memcpy(y, temp);
123
124
125         if (abs(norm - z_prev_norm) < 10e-16)
126             break;
127     }
128     cout << gsl_blas_dnrm2(z) << " dla:" << endl;
129     gsl_vector_fprintf(stdout, y, "%f");
130     cout << endl;
131
132     // wyznaczanie wektora prostopadlego do wektora e1
133     gsl_vector *e1;
134     e1 = gsl_vector_alloc(size);
135     gsl_vector_memcpy(e1, y);
136     gsl_vector_set_all(y, 1);
137     double sum = gsl_blas_dasum(e1) - gsl_vector_get(e1, size - 1);
138     double buf = -(sum / gsl_vector_get(e1, size - 1));
139     gsl_vector_set(y, size - 1, buf);
140     double normY = gsl_blas_dnrm2(y);
141     gsl_blas_dscal(1 / normY, y);
142
143
144     while (true) {
145         // poprzednia norma aby porownac i zakonczyc w odpowiednim momencie
146         double z_prev_norm = gsl_blas_dnrm2(z);
147
148         // wyliczanie Ay = z
149         gsl_blas_dgemv(CblasNoTrans, 1.0, M, y, 0.0, z);
150
151         // reortogonalizacja
152         double data = 0;
153         double *e1z = &data;
154         gsl_blas_ddot(e1, z, e1z);
155         gsl_vector *e1_e1z;
156         e1_e1z = gsl_vector_alloc(size);
157         gsl_vector_memcpy(e1_e1z, e1);
158         gsl_blas_dscal(*e1z, e1_e1z);
159         gsl_vector_sub(z, e1_e1z);
160
161         // wyliczanie kolejnego y
162         double norm = gsl_blas_dnrm2(z);
163         gsl_vector_memcpy(temp, z);
164         gsl_vector_scale(temp, 1 / norm);
165         gsl_vector_memcpy(y, temp);
166
167
168         if (abs(norm - z_prev_norm) < 10e-16)
169             break;
170     }
171     cout << gsl_blas_dnrm2(z) << " dla:" << endl;
172     gsl_vector_fprintf(stdout, y, "%f");
173 }
174
175
176 int main() {
177     vector<vector<double>> A_ = {{19.0 / 12.0,  13.0 / 12.0,  5.0 / 6.0,  5.0 / 6.0,
```

```
        13.0 / 12.0,  -17.0 / 12.0},
178                                 {13.0 / 12.0,  13.0 / 12.0,  5.0 / 6.0,  5.0 / 6.0,
        -11.0 / 12.0, 13.0 / 12.0},
179                                 {5.0 / 6.0,    5.0 / 6.0,    5.0 / 6.0,  -1.0 / 6.0,
        5.0 / 6.0,    5.0 / 6.0},
180                                 {5.0 / 6.0,    5.0 / 6.0,    -1.0 / 6.0, 5.0 / 6.0,
        5.0 / 6.0,    5.0 / 6.0},
181                                 {13.0 / 12.0,  -11.0 / 12.0, 5.0 / 6.0,  5.0 / 6.0,
        13.0 / 12.0,  13.0 / 12.0},
182                                 {-17.0 / 12.0, 13.0 / 12.0,  5.0 / 6.0,  5.0 / 6.0,
        13.0 / 12.0,  19.0 / 12.0}};
183     int A_size = A_.size();
184     vector<vector<double>> B_ = {{1, 2, 1, 1, 3, 4},
185                                  {2, 2, 3, 5, 5, 6},
186                                  {1, 3, 4, 4, 5, 5},
187                                  {1, 5, 4, 5, 4, 6},
188                                  {3, 5, 5, 4, 6, 2},
189                                  {4, 6, 5, 6, 2, 0}};
190
191     int B_size = B_.size();
192
193     gsl_matrix *A;
194     A = gsl_matrix_alloc(A_size, A_size);
195
196     for (int i = 0; i < A_size; ++i) {
197         for (int j = 0; j < A_size; ++j) {
198             gsl_matrix_set(A, i, j, A_[i][j]);
199         }
200     }
201
202     gsl_matrix *B;
203     B = gsl_matrix_alloc(B_size, B_size);
204
205     for (int i = 0; i < A_size; ++i) {
206         for (int j = 0; j < A_size; ++j) {
207             gsl_matrix_set(B, i, j, B_[i][j]);
208         }
209     }
210     cout << "Najwieksze wartosci wlasne A: " << endl;
211     powerMethodMax(A);
212     cout << "\nNajmniejsze wartosci wlasne A: " << endl;
213     powerMethodMin(A);
214     cout << "\n\nNajwieksze wartosci wlasne B: " << endl;
215     powerMethodMax(B);
216     cout << "\nNajmniejsze wartosci wlasne B: " << endl;
217     powerMethodMin(B);
218
219
220     return 0;
221 }
```

Wynik

$$|\lambda_1| > |\lambda_2| > ... > |\lambda_N| > 0$$

Dla macierzy A:

$|\lambda_1| = 4$ dla $\begin{bmatrix}0.408248 & 0.408248 & 0.408248 & 0.408248 & 0.408248 & 0.408248\end{bmatrix}$

$|\lambda_2| = 3$ dla $\begin{bmatrix}0.707107 & -0.000000 & -0.000000 & -0.000000 & -0.000000 & -0.707107\end{bmatrix}$

$|\lambda_N| = 1$ dla $\begin{bmatrix}-0.288675 & -0.288675 & 0.577350 & 0.577350 & -0.288675 & -0.288675\end{bmatrix}$

$|\lambda_{N-1}| = 1$ dla $\begin{bmatrix}0.000000 & -0.000000 & -0.707107 & 0.707107 & -0.000000 & 0.000000\end{bmatrix}$

Dla macierzy B:

$|\lambda_1| = 22.5925$ dla $\begin{bmatrix}0.219759 & 0.426288 & 0.414803 & 0.471336 & 0.456172 & 0.409478\end{bmatrix}$

$|\lambda_2| = 7.18717$ dla $\begin{bmatrix}0.319671 & 0.379840 & 0.256496 & 0.194316 & -0.257043 & -0.764141\end{bmatrix}$

$|\lambda_N| = 0.264982$ dla $\begin{bmatrix}0.340267 & -0.245391 & 0.676209 & -0.414990 & -0.355291 & 0.261333\end{bmatrix}$

$|\lambda_{N-1}| = 1.46728$ dla $\begin{bmatrix}-0.758773 & -0.212513 & 0.539939 & 0.195371 & 0.062692 & -0.213229\end{bmatrix}$