Marcin Rociek
Informatyka, II rok

Metody Obliczeniowe w Nauce i Technice

laboratorium

zestaw 8: wartości własne i wektory własne macierzy

Wyznaczyć wartości własne i wektory własne dla macierzy

$$\begin{bmatrix} 2 & 2 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 0 & 1 & 3 & 2 \\ 0 & 0 & 2 & 4 \end{bmatrix}$$

a) metodą bisekcji
b) metodą QR

Porównać złożoność obu metod.

---

a) Metoda bisekcji.

Do rozwiązania zadania napisałem własną funkcję szukającą wartości własnych metodą bisekcji. Ponieważ zadana macierz jest macierzą trójdiagonalną symetryczną, wykorzystałem do tego celu algorytm szukający wartości własnych dla takiej macierzy podany w książce Z. Fortuny: *Metody numeryczne*.

Treść programu:

```
#include <stdio.h>

/*
 * Funkcja wyznaczajaca wielomian charakterystyczny
 */
void calc_omegas(double lambda, double *omega, double *a, double *b, int n)
{
  int i;
  omega[0] = 1.0;
  omega[1] = b[1]-lambda;
  for (i=2; i<=n; i++)
    omega[i] = ((b[i]-lambda)*omega[i-1])-(a[i]*a[i]*omega[i-2]);
}

/*
 * Funkcja wyznaczajaca ilosc zmian znaku wsrod kolejnych wspolczynnikow
 * wielomianu charakterystycznego
 */
int sign_changes(double *omega, int n)
{
  int s = 0, i;
  for (i=1; i<=n; i++)
   if ((omega[i]>=0.0&&omega[i-1]<0.0)||(omega[i]<0.0&&omega[i-1]>=0.0))
      s++;
  return s;
}

/*
 * Funkcja szukajaca k-tej wartosci wlasnej macierzy trojdiagonalnej symetrycznej
 * metoda bisekcji
 */
double eigenval(double *a, double *b, double *omega, int n, int k)
{
  double alfa, beta;
  double eps = 0.00001;
  int i, m;

  beta = b[1]*b[1];
  for (i=2; i<=n; i++)
  {
    beta += b[i]*b[i] + a[i]*a[i];
  }
  beta = sqrt(beta);
  alfa = -beta;
```

```
  for (; ; )
  {
    double t = 0.5*(alfa+beta);
    if ((beta-alfa)<(2.0*eps))
      return t;
    calc_omegas(t, omega, a, b, n);
    if (omega[n]==0.0)
    {
      m = sign_changes(omega, n-1);
      if (m==(n-k))
        return t;
    }
    else
      m = sign_changes(omega, n);
    if (m<=(n-k))
      alfa = t;
    else
      beta = t;
  }
}

/*
 * Funkcja wyznaczajaca wektor wlasny na podstawie wartosci wlasnej
 */
void eigenvec(double *x, double val, double *a, double *b, int n)
{
  int i;
  x[1] = 1.0;
  x[2] = (val-b[1])/a[2];
  for (i=2; i<n; i++)
    x[i+1] = ((val-b[i])*x[i]-a[i]*x[i-1])/a[i+1];
}

/*
 * Czesc glowna programu
 */
#define NP 4
void main(void)
{
  double a[5] = {0,0,   2, 1, 2};
  double b[5] = {0,     2, 1, 3, 4};
  double omega[5], x[5];
  int i, j;

  for (i=1; i<=NP; i++)
  {
    double val = eigenval(a, b, omega, NP, i);
    eigenvec(x, val, a, b, NP);
    printf("wartosc wlasna: %f\n", val);
    printf("wektor wlasny:\n");
    for (j=1; j<=NP; j++)
      printf("%f ", x[j]);
    printf("\n\n");
  }
}
```

Wynik działania programu:

```
wartosc wlasna: 5.671792
wektor wlasny:
1.000000 1.835896 6.576925 7.868141

wartosc wlasna: 3.597972
wektor wlasny:
1.000000 0.798986 0.075743 -0.376847

wartosc wlasna: 1.509405
wektor wlasny:
1.000000 -0.245297 -2.124956 1.706373

wartosc wlasna: -0.779166
wektor wlasny:
1.000000 -1.389583 0.472298 -0.197655
```

W metodzie tej ilość iteracji potrzebnej do wyznaczenia *k*-tej wartości własnej l_k wynosi

$$IT = E\left(\log_2 \frac{b_0 - a_0}{r} \right),$$

gdzie $[a_0,b_0]$ – przedział, w którym znajdują się wszystkie wartości własne
r - dopuszczalny błąd, z jakim obliczamy wartość własną
W każdej iteracji wykonywane jest $(n+1)$ mnożeń i dzieleń.

---

b) Metoda QR.

W tym zadaniu wykorzystałem funkcję **hqr** z biblioteki *Numerical Recipes*, która oblicza wartości własne macierzy metodą QR.

Treść programu:

```
#include <stdio.h>
#include "nrutil.h"
#include "matrix.h"

extern void hqr(double **a, int n, double wr[], double wi[]);

/*
 * Funkcja wyznaczajaca wektor wlasny na podstawie wartosci wlasnej
 */
void eigenvec(double *x, double val, double *a, double *b, int n)
{
  int i;
  x[1] = 1.0;
  x[2] = (val-b[1])/a[2];
  for (i=2; i<n; i++)
    x[i+1] = ((val-b[i])*x[i]-a[i]*x[i-1])/a[i+1];
}

/*
 * Czesc glowna programu
 */
#define NP 4
void main(void)
{
  int i,j;
  static double c[NP][NP]= { { 2.0, 2.0, 0.0, 0.0 },
                             { 2.0, 1.0, 1.0, 0.0 },
                             { 0.0, 1.0, 3.0, 2.0 },
                             { 0.0, 0.0, 2.0, 4.0 } };
  double _a[5] = {0,0,   2, 1, 2};
  double _b[5] = {0,     2, 1, 3, 4};
  double x[5];
  double *wr, *wi, **a;

  wr = dvector(1, NP);
  wi = dvector(1, NP);
  a = dmatrix(1, NP, 1, NP);
  for (j=0; j<NP; j++)
    for (i=0; i<NP; i++)
      a[j+1][i+1] = c[j][i];

  hqr(a, NP, wr, wi);

  for (i=1; i<=NP; i++)
  {
    printf("wartosc wlasna: %f + i%.1f\n", wr[i], wi[i]);
    printf("wektor wlasny:\n");
    eigenvec(x, wr[i], _a, _b, NP);
    vec_show(x, NP);
    printf("\n");
  }

  free_dmatrix(a, 1, NP, 1, NP);
  free_dvector(wi, 1, NP);
  free_dvector(wr, 1, NP);
}
```

Wynik działania programu:

```
wartosc wlasna: -0.779164 + i0.0
wektor wlasny:
1.000000 -1.389582 0.472294 -0.197647

wartosc wlasna: 3.597975 + i0.0
wektor wlasny:
1.000000 0.798988 0.075751 -0.376845

wartosc wlasna: 5.671789 + i0.0
wektor wlasny:
1.000000 1.835894 6.576911 7.868112

wartosc wlasna: 1.509399 + i0.0
wektor wlasny:
1.000000 -0.245300 -2.124956 1.706380
```

Złożoność metody:
Aby wyznaczyć macierze pomocnicze należy wykonać *9(n-1)* mnożeń, *3(n-1)* dodawań i *(n-1)* pierwiastkowań, natomiast aby wymnożyć macierze wykonuje się *(4n-1)* mnożeń i *n* dodawań.