Marcin Rociek
Informatyka, II rok

Metody Obliczeniowe w Nauce i Technice

laboratorium

zestaw 6: Metody iteracyjne rozwiązywania układów równań liniowych

Dany jest układ równań liniowych $Ax=B$.
Macierz A o wymiarze $n \cdot n$ określona jest wzorem:

$$\begin{bmatrix} 1 & \frac{1}{2} & 0 & 0 & \cdots & 0 & 0 \\ \frac{1}{2} & 2 & \frac{1}{3} & 0 & & 0 & 0 \\ 0 & \frac{1}{3} & 2 & \frac{1}{4} & & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 2 & & 0 & 0 \\ \vdots & & & & \ddots & & \vdots \\ 0 & 0 & 0 & 0 & & 2 & \frac{1}{n} \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{n} & 1 \end{bmatrix}$$

Przyjmij wektor x jako n-elementową permutację ze zbioru $\{-1, 1\}$ i oblicz wektor b. Metodą Jacobiego oraz metodą Czebyszewa rozwiąż układ równań $Ax=b$ przyjmując jako niewiadomą wektor x. W obu przypadkach oszacuj liczbę iteracji przyjmując test stopu:

a) $\left\| x^{(i+1)} - x^{(i)} \right\| < r$, *b)* $\frac{1}{\|b\|} \left\| Ax^{(i+1)} - b \right\| < r$

1: Metoda Jacobiego

W metodzie Jacobiego kolejne przybliżenia rozwiązania otrzymujemy ze wzoru:

$$Dx^{(i+1)} = -(L+U)x^{(i)} + b,$$

gdzie D to macierz zawierająca elementy z przekątnej macierzy A, zaś macierz $L+U$ to macierz A pozbawiona elementów na przekątnej głównej.
Mnożąc lewostronnie przez D^{-1} otrzymujemy:

$$x^{(i+1)} = D^{-1}b - D^{-1}(L+U)x^{(i)}$$

Treść programu realizującego ten algorytm:

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

/*
 * funkcja rozwiazujaca uklad rownan liniowych metoda Jacobiego
 */
int jacobisolve(double **a, double *x, double *b, int n, int stop, double blad)
{
  int i, j, iters = 0, calc = 1;
  double *d1_b = dvector(1, n);
  double *xold = dvector(1, n);
  double **d1_lu = dmatrix(1, n, 1, n);
  double *d1_lu_xold = dvector(1, n);
  double *tmp = dvector(1, n);

  for(i=1; i<=n; i++)
    xold[i] = 0.0;

  /* stworzenie wektora D^(-1) * b */
  for(i=1; i<=n; i++)
    d1_b[i] = b[i]/a[i][i];

  /* stworzenie macierzy D^(-1) * (L+U) */
  for(j=1; j<=n; j++)
    for(i=1; i<=n; i++)
    {
      if(i==j)
        d1_lu[j][i] = 0.0;
      else
        d1_lu[j][i] = a[j][i]/a[j][j];
```

```
    }

  /* petla obliczeniowa */
  while(calc)
  {
    /* obliczenie wyrazenia x = D^(-1)*b - D^(-1)*(L+U)*xold */
    mat_vec_mul(d1_lu_xold, n, d1_lu, xold);
    vec_sub(x, n, d1_b, d1_lu_xold);

    /* sprawdzenie, czy zakonczyc obliczenia */
    if(stop==1)
    {
      vec_sub(tmp, n, x, xold);
      if(vec_norm(tmp, n)<blad)
        calc = 0;
    }
    else
    {
      mat_vec_mul(tmp, n, a, x);
      vec_sub(tmp, n, tmp, b);
      if((vec_norm(tmp, n)/vec_norm(b, n))<blad)
        calc = 0;
    }

    /* podstawienie xold = x */
    for(i=1; i<=n; i++)
      xold[i] = x[i];

    /* licznik iteracji */
    iters++;
  }

  free_dvector(d1_b, 1, n);
  free_dvector(xold, 1, n);
  free_dmatrix(d1_lu, 1, n, 1, n);
  free_dvector(d1_lu_xold, 1 , n);
  free_dvector(tmp, 1, n);

  return iters;
}

/*
 * czesc glowna programu
 */
void main(int argc, char **argv)
{
  int n = 5;
  double blad = 0.01;

  double **a, *x, *b;
  double stuff[] = { 1.0, -1.0 };
  int i, j;

  if(argc>1)
    n = atoi(argv[1]);
  if(argc>2)
    blad = atof(argv[2]);

  a = dmatrix(1, n, 1, n);
  x = dvector(1, n);
  b = dvector(1, n);

  /* stworzenie macierzy A */
  for(j=1; j<=n; j++)
    for(i=1; i<=n; i++)
    {
      if(j==i)
      {
        if(i==1 || i==n)
          a[j][i] = 1;
       else
         a[j][i] = 2;
      }
      else
      if (i-j==1)
        a[j][i] = 1.0/((float)i);
      else
      if (j-i==1)
        a[j][i] = 1.0/((float)j);
      else
        a[j][i] = 0.0;
    }
```

```
    /* stworzenie wektora x */
    for(i=1; i<=n; i++)
      x[i] = stuff[i&1];

    /* stworzenie wektora b (jako A*x) */
    mat_vec_mul(b, n, a, x);

    printf("a=\n");
    mat_show(a, n);

    printf("\nx=\n");
    vec_show(x, n);

    printf("\nb=\n");
    vec_show(b, n);

    printf("\nprzyjety blad: %f\n", blad);

    i = jacobisolve(a, x, b, n, 1, blad);
    printf("\nwynik (test stopu 1):\n");
    vec_show(x, n);
    printf("ilosc iteracji: %d\n", i);

    i = jacobisolve(a, x, b, n, 2, blad);
    printf("\nwynik (test stopu 2):\n");
    vec_show(x, n);
    printf("ilosc iteracji: %d\n", i);

}
```

Wynik działania programu dla macierzy 6x6 i dokładności 0.001:

```
a=
| 1.000000 0.500000 0.000000 0.000000 0.000000 0.000000 |
| 0.500000 2.000000 0.333333 0.000000 0.000000 0.000000 |
| 0.000000 0.333333 2.000000 0.250000 0.000000 0.000000 |
| 0.000000 0.000000 0.250000 2.000000 0.200000 0.000000 |
| 0.000000 0.000000 0.000000 0.200000 2.000000 0.166667 |
| 0.000000 0.000000 0.000000 0.000000 0.166667 1.000000 |

x=
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000

b=
-0.500000 1.166667 -1.416667 1.550000 -1.633333 0.833333

przyjety blad: 0.001000

wynik (test stopu 1):
-0.999661 0.999737 -0.999873 0.999958 -0.999988 0.999995
ilosc iteracji: 9

wynik (test stopu 2):
-0.997832 0.998315 -0.999187 0.999725 -0.999919 0.999964
ilosc iteracji: 7
```

Dla dokładności 0.00001:

```
a=
| 1.000000 0.500000 0.000000 0.000000 0.000000 0.000000 |
| 0.500000 2.000000 0.333333 0.000000 0.000000 0.000000 |
| 0.000000 0.333333 2.000000 0.250000 0.000000 0.000000 |
| 0.000000 0.000000 0.250000 2.000000 0.200000 0.000000 |
| 0.000000 0.000000 0.000000 0.200000 2.000000 0.166667 |
| 0.000000 0.000000 0.000000 0.000000 0.166667 1.000000 |

x=
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000

b=
-0.500000 1.166667 -1.416667 1.550000 -1.633333 0.833333

przyjety blad: 0.000010

wynik (test stopu 1):
-0.999997 0.999997 -0.999999 1.000000 -1.000000 1.000000
```

```
ilosc iteracji: 14

wynik (test stopu 2):
-0.999979 0.999983 -0.999992 0.999997 -0.999999 1.000000
ilosc iteracji: 12
```

Jak widać test stopu nr 2 powoduje wcześniejsze zakończenie obliczeń.

2: Metoda Czebyszewa

Wyznaczając rozwiązanie układu metodą Czebyszewa korzystamy z następującego algorytmu:

Krok 0:

$$i=0,\ w_0=\frac{b-a}{b+a},\ c=\frac{2}{b+a},\ L=2\frac{b+a}{b-a},\ x^{(-1)}=0$$

Krok 1:

$k=0,\ x^{(i)}=x_0$ – dany wektor, $w_{i-1}=0,\quad w_i=w_0$

Krok 2:

$$x^{(i+1)}=x^{(i)}+w_i w_{i-1}\left(x^{(i)}-x^{(i-1)}\right)-c\left(1+w_i w_{i-1}\right)-\left(Ax^{(i)}-b\right),\quad w_{i+1}=\frac{1}{L-w_i}$$

Krok 3:

$k=k+1,\ i=i+1.$
Jeśli $k<s.$, to wracamy do kroku 2, w przeciwnym wypadku podstawiamy $x_0=x^{(i)}$ i wracamy do kroku 1.

Dla uproszczenia założyłem, że $s=¥$, czyli nie dokonuje się odnowienie algorytmu co s iteracji.
Ponadto, ponieważ w danym momencie trzeba znać jedynie dwie poprzednie wartości w, oraz wektora x, wartości te są zapisywane w małej 4 elementowej tablicy (wektor x w 4 wierszowej macierzy).

Treść programu:

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

/*
 * funkcja rozwiazujaca uklad rownan liniowych metoda Czebyszewa
 */
int chebyshevsolve(double **a, double *x0, double *b, int n, int stop, double blad)
{
  int i, im1, ip1, k, j, iters = 0, calc = 1;
  double c, l, oo, c1oo;

  double **x = dmatrix(0, 4, 1, n);
  double *omega = dvector(0, 4);
  double *Ax = dvector(1, n);
  double *Ax_b = dvector(1, n);
  double *tmp = dvector(1, n);

  double alfa = 5.0, beta = 5000.0;

  c = 2.0/(beta+alfa);
  l = 2.0*(beta+alfa)/(beta-alfa);

  omega[(-1)&3] = 0.0;
  omega[0] = (beta-alfa)/(beta+alfa);

  for(k=1; k<=n; k++)
  {
    x[(-1)&3][k] = 0.0;
    x[0][k] = x0[k];
  }
```

```
  while(calc)
  {
    i = iters&3;
    ip1 = (iters+1)&3;
    im1 = (iters-1)&3;

    mat_vec_mul(Ax, n, a, x[i]);
    vec_sub(Ax_b, n, Ax, b);

    oo = omega[i]*omega[im1];
    c1oo = c*(1+oo);

    for(k=1; k<=n; k++)
      x[ip1][k] = x[i][k] +oo*(x[i][k]-x[im1][k]) -c1oo -Ax_b[k];

    omega[ip1] = 1.0/(l-omega[i]);

    /* sprawdzenie, czy zakonczyc obliczenia */
    if(stop==1)
    {
      vec_sub(tmp, n, x[i], x[im1]);
      if(vec_norm(tmp, n)<blad)
        calc = 0;
    }
    else
    {
      mat_vec_mul(tmp, n, a, x[i]);
      vec_sub(tmp, n, tmp, b);
      if((vec_norm(tmp, n)/vec_norm(b, n))<blad)
        calc = 0;
    }

    /* licznik iteracji */
    iters++;
  }

  i = iters&3;
  for(k=1; k<=n; k++)
    x0[k] = x[i][k];

  free_dmatrix(x, 0, 4, 1, n);
  free_dvector(omega, 0, 4);
  free_dvector(Ax, 1, n);
  free_dvector(Ax_b, 1, n);
  free_dvector(tmp, 1, n);

  return iters;
}

/*
 * czesc glowna programu
 */
void main(int argc, char **argv)
{
  int n = 5;
  double blad = 0.01;

  double **a, *x, *b;
  double stuff[] = { 1.0, -1.0 };
  int i, j;

  if(argc>1)
    n = atoi(argv[1]);
  if(argc>2)
    blad = atof(argv[2]);

  a = dmatrix(1, n, 1, n);
  x = dvector(1, n);
  b = dvector(1, n);

  /* stworzenie macierzy A */
  for(j=1; j<=n; j++)
    for(i=1; i<=n; i++)
    {
      if(j==i)
      {
        if(i==1 || i==n)
          a[j][i] = 1;
       else
         a[j][i] = 2;
      }
      else
```

```
      if (i-j==1)
        a[j][i] = 1.0/((float)i);
      else
      if (j-i==1)
        a[j][i] = 1.0/((float)j);
      else
        a[j][i] = 0.0;
    }

    /* stworzenie wektora x */
    for(i=1; i<=n; i++)
      x[i] = stuff[i&1];

    /* stworzenie wektora b (jako A*x) */
    mat_vec_mul(b, n, a, x);

    printf("a=\n");
    mat_show(a, n);

    printf("\nx=\n");
    vec_show(x, n);

    printf("\nb=\n");
    vec_show(b, n);

    printf("\nprzyjety blad: %f\n", blad);

    i = chebyshevsolve(a, x, b, n, 1, blad);
    printf("\nwynik (test stopu 1):\n");
    vec_show(x, n);
    printf("ilosc iteracji: %d\n", i);

    i = chebyshevsolve(a, x, b, n, 2, blad);
    printf("\nwynik (test stopu 2):\n");
    vec_show(x, n);
    printf("ilosc iteracji: %d\n", i);

}
```

Wynik działania programu dla macierzy 6x6, dla dokładności 0.001:

```
a=
| 1.000000 0.500000 0.000000 0.000000 0.000000 0.000000 |
| 0.500000 2.000000 0.333333 0.000000 0.000000 0.000000 |
| 0.000000 0.333333 2.000000 0.250000 0.000000 0.000000 |
| 0.000000 0.000000 0.250000 2.000000 0.200000 0.000000 |
| 0.000000 0.000000 0.000000 0.200000 2.000000 0.166667 |
| 0.000000 0.000000 0.000000 0.000000 0.166667 1.000000 |

x=
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000

b=
-0.500000 1.166667 -1.416667 1.550000 -1.633333 0.833333

przyjety blad: 0.001000

wynik (test stopu 1):
-1.000993 0.999540 -1.000560 0.999387 -1.000646 0.998874
ilosc iteracji: 2

wynik (test stopu 2):
-1.000351 0.999584 -1.000613 0.999580 -1.000207 0.999938
ilosc iteracji: 2
```

Dla dokładności 0.00001:

```
a=
| 1.000000 0.500000 0.000000 0.000000 0.000000 0.000000 |
| 0.500000 2.000000 0.333333 0.000000 0.000000 0.000000 |
| 0.000000 0.333333 2.000000 0.250000 0.000000 0.000000 |
| 0.000000 0.000000 0.250000 2.000000 0.200000 0.000000 |
| 0.000000 0.000000 0.000000 0.200000 2.000000 0.166667 |
| 0.000000 0.000000 0.000000 0.000000 0.166667 1.000000 |

x=
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000
```

```
b=
-0.500000 1.166667 -1.416667 1.550000 -1.633333 0.833333

przyjety blad: 0.000010

wynik (test stopu 1):
-1.000670 0.999838 -1.000313 0.999688 -1.000290 0.999292
ilosc iteracji: 81
```

Test stopu nr 2 nie dał w tym przypadku pozytywnych rezultatów. Po jednej z iteracji wektor *x* przestaje się już zmieniać, a nie spełnia się warunek stopu i procedura zapętla się.

Pomocnicze funkcje operujące na macierzach i wektorach wykorzystane w obu powyższych programach:

```
#include <math.h>
#include <stdio.h>

/*
 * funkcja mnozaca macierz przez wektor
 */
void mat_vec_mul(double *v, int n, double **a, double *x)
{
  int j, k;

  for(j=1; j<=n; j++)
  {
    double sum = 0.0;
    for(k=1; k<=n; k++)
      sum += a[j][k]*x[k];
    v[j] = sum;
  }
}

/*
 * funkcja odejmujaca wektory od siebie
 */
void vec_sub(double *c, int n, double *a, double *b)
{
  int i;

  for(i=1; i<=n; i++)
    c[i] = a[i]-b[i];
}

/*
 * funkcja liczaca norme wektora
 */
double vec_norm(double *a, int n)
{
  double d = 0.0;
  int i;

  for(i=1; i<=n; i++)
    d += a[i]*a[i];
  return sqrt(d);
}

/*
 * funkcja wypisujaca macierz (n x n)
 */
void mat_show(double **a, int n)
{
  int i, j;
  for(j=1; j<=n; j++)
  {
    printf("| ");
    for(i=1; i<=n; i++)
      printf("%f ", a[j][i]);
    printf("|\n");
  }
}

/*
 * funkcja wypisujaca wektor
 */
void vec_show(double *a, int n)
{
  int i;
```

```
  for(i=1; i<=n; i++)
    printf("%f ", a[i]);
  printf("\n");
}
```