Marcin Rociek
Informatyka, II rok

Metody Obliczeniowe w Nauce i Technice

laboratorium

zestaw 5: Metody bezpośrednie rozwiązywania układów równań liniowych

---

Zadanie 1: Dany jest układ równań liniowych $Ax=b$. Macierz A o wymiarze $n \times n$ jest określona wzorem: $[a_{ij}]=[1/(i+j-1)]$, $i,j=1,...,n$. Przyjmij wektor x jako dowolną n-elementową permutację ze zbioru $\{1,-1\}$ i oblicz wektor b (operując na wartościach wymiernych). Następnie metodą eliminacji Gaussa rozwiąż układ równań liniowych $Ax=b$ (przyjmując jako niewiadomą wektor x). Przyjmij różną precyzję dla wprowadzania znanych wartości macierzy A i wektora b oraz różną precyzję dla przechowywania tych wartości. Sprawdź jak błędy zaokrągleń zaburzają rozwiązanie.

Do rozwiązania tego zadania wykorzystałem funkcję **gaussj** z biblioteki *Numerical Recipes*.

Treść programu realizującego zadanie:

```
#include "nr.h"
#include "nrutil.h"
#include <math.h>

extern void gaussj(float **a, int n, float **b, int m);

void main(void)
{
  int n = 4;
  float vals[] = { 1.0, -1.0 };
  int i, j, *idx;
  float **a, **b, *x, d;

  a = matrix(1, n, 1, n);
  b = matrix(1, n, 1, 1);
  x = vector(1, n);
  idx = ivector(1, n);

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

  /* stworzenie macierzy A */
  for(j=1; j<=n; j++)
    for(i=1; i<=n; i++)
         a[j][i] = 1.0/((float)(i+j)-1.0);

  /* wymnozenie macierzy A przez wektor x */
  for(j=1; j<=n; j++)
  {
    b[j][1] = 0.0;
       for(i=1; i<=n; i++)
      b[j][1] += a[j][i]*x[i];
  }

  /* wypisanie macierzy A */
  printf("macierz A:\n");
  for(j=1; j<=n; j++)
  {
    for(i=1; i<=n; i++)
      printf("%f ", a[j][i]);
    printf("\n");
  }

  /* wypisanie wektora x */
  printf("\nwektor x:\n");
  for(i=1; i<=n; i++)
    printf("%f ", x[i]);
  printf("\n");

  /* wypisanie wektora b (A*x) */
  printf("\nwektor b (Ax):\n");
  for(i=1; i<=n; i++)
    printf("%f ", b[i][1]);
  printf("\n");
```

```
  /* rozwiazanie ukladu Ax=b */
  gaussj(a, n, b, 1);

  /* wypisanie wektora x po rozwiazaniu Ax=b */
  printf("\nwektor x (jako rozwiazanie ukladu Ax=b:\n");
  for(i=1; i<=n; i++)
    printf("%f ", b[i][1]);
  printf("\n");
}
```

Aby sprawdzić błędy wynikające z niedokładności reprezentacji zmiennoprzecinkowej liczb rzeczywistych, kompilowałem powyższy program dla pojedynczej precyzji (float) oraz dla podwójnej (double).

Dla macierzy 8 x 8 elementów, wyniki przedstawiają się następująco:

Pojedyncza precyzja:

```
macierz A:
1.000000 0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000
0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111
0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000
0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909
0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333
0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923
0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429
0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667

wektor x:
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000

wektor b (Ax):
-0.634524 -0.254365 -0.145635 -0.096789 -0.069877 -0.053200 -0.042039 -0.034152

wektor x (jako rozwiazanie ukladu Ax=b):
-1.000359 1.022853 -1.320735 2.801413 -5.956080 8.110349 -6.108272 2.451216
```

Podwójna precyzja:

```
macierz A:
1.000000 0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000
0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111
0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000
0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909
0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333
0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923
0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429
0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667

wektor x:
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000

wektor b (Ax):
-0.634524 -0.254365 -0.145635 -0.096789 -0.069877 -0.053200 -0.042039 -0.034152

wektor x (jako rozwiazanie ukladu Ax=b):
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000
```

Jak widać precyzja pojedyncza jest w takim przypadku absolutnie niewystarczająca, gdyż występują znaczne różnice od prawdziwych wartości rozwiązania (1.0 zamieniło się na 8.11...).

Spróbowałem zbadać, jak mała musi być macierz, aby precyzja pojedyncza dawała zadowalające wyniki. Przykładowo dla macierzy 4 x 4 wynik wygląda następująco:

```
macierz A:
1.000000 0.500000 0.333333 0.250000
0.500000 0.333333 0.250000 0.200000
0.333333 0.250000 0.200000 0.166667
0.250000 0.200000 0.166667 0.142857

wektor x:
-1.000000 1.000000 -1.000000 1.000000

wektor b (Ax):
-0.583333 -0.216667 -0.116667 -0.073810

wektor x (jako rozwiazanie ukladu Ax=b):
```

```
-1.000004 1.000045 -1.000106 1.000068
```

Mamy więc odchylenia dopiero na czwartym miejscu po przecinku.

Podobny eksperyment przeprowadziłem dla podwójnej precyzji, chcąc sprawdzić dla jak dużej macierzy wynik zacznie się pogarszać. Przykładowy wynik dla macierzy 10 x 10:

```
macierz A:
1.000000 0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000
0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909
0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333
0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923
0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429
0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667
0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667 0.062500
0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667 0.062500 0.058824
0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667 0.062500 0.058824 0.055556
0.100000 0.090909 0.083333 0.076923 0.071429 0.066667 0.062500 0.058824 0.055556 0.052632

wektor x:
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000

wektor b (Ax):
-0.645635 -0.263456 -0.153211 -0.103200 -0.075372 -0.057961 -0.046205 -0.037828 -0.031616 -
0.026864

wektor x (jako rozwiazanie ukladu Ax=b):
-1.000000 1.000000 -1.000000 0.999996 -0.999980 0.999944 -0.999909 0.999912 -0.999954 0.999990
```

Jak widać rozbieżności pokazały się dopiero na piątym miejscu po przecinku.

Zadanie 2: To samo, co w zadaniu 1, ale znaleźć rozwiązanie metodą Choleskiego.

W zadaniu tym posłużyłem się funkcjami **choldc** i **cholsl** z biblioteki *Numerical Recipes*, które wespół rozwiązują układ równań liniowych metodą Choleskiego.

Program wykonujący zadanie:

```
#include "nr.h"
#include "nrutil.h"
#include <math.h>

extern void choldc(float **a, int n, float p[]);
extern void cholsl(float **a, int n, float p[], float b[], float x[]);

void main(void)
{
  int n = 3;
  float vals[] = { 1.0, -1.0 };
  int i, j;
  float **a, *b, *x, *p;

  a = matrix(1, n, 1, n);
  b = vector(1, n);
  x = vector(1, n);
  p = vector(1, n);

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

  /* stworzenie macierzy A */
  for(j=1; j<=n; j++)
    for(i=1; i<=n; i++)
      a[j][i] = 1.0/((float)(i+j)-1.0);

  /* wymnozenie macierzy A przez wektor x */
  for(j=1; j<=n; j++)
  {
    b[j] = 0.0;
    for(i=1; i<=n; i++)
      b[j] += a[j][i]*x[i];
  }
```

```
  /* wypisanie macierzy A */
  printf("macierz A:\n");
  for(j=1; j<=n; j++)
  {
    for(i=1; i<=n; i++)
          printf("%f ", a[j][i]);
        printf("\n");
  }

  /* wypisanie wektora x */
  printf("\nwektor x:\n");
  for(i=1; i<=n; i++)
    printf("%f ", x[i]);
  printf("\n");

  /* wypisanie wektora b (A*x) */
  printf("\nwektor b (Ax):\n");
  for(i=1; i<=n; i++)
    printf("%f ", b[i]);
  printf("\n");

  /* rozwiazanie ukladu Ax=b */
  choldc(a, n, p);
  cholsl(a, n, p, b, x);

  /* wypisanie wektora x po rozwiazaniu Ax=b */
  printf("\nwektor x (jako rozwiazanie ukladu Ax=b):\n");
  for(i=1; i<=n; i++)
    printf("%f ", x[i]);
  printf("\n");
}
```

Sprawdzenie jak błędy zaokrągleń zaburzają rozwiązanie wykonałem podobnie jak w zadaniu poprzednim.

Precyzja pojedyncza:

```
macierz A:
1.000000 0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000
0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111
0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000
0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909
0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333
0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923
0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429
0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667

wektor x:
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000

wektor b (Ax):
-0.634524 -0.254365 -0.145635 -0.096789 -0.069877 -0.053200 -0.042039 -0.034152

wektor x (jako rozwiazanie ukladu Ax=b):
-0.999980 0.997745 -0.963225 0.782110 -0.391674 0.130879 -0.384401 0.828535
```

Precyzja podwójna:

```
macierz A:
1.000000 0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000
0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111
0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000
0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909
0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333
0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923
0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429
0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667

wektor x:
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000

wektor b (Ax):
-0.634524 -0.254365 -0.145635 -0.096789 -0.069877 -0.053200 -0.042039 -0.034152

wektor x (jako rozwiazanie ukladu Ax=b):
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000
```

Już na tym przykładzie widać, że metoda Choleskiego jest również bardzo czuła na błędy zaokrągleń (dla pojedynczej precyzji), aczkolwiek inaczej niż metoda Gaussa. W metodzie Gaussa rozwiązanie układu 8x8 „rozjeżdżało" się poza jedynki, natomiast w przypadku metody Choleskiego rozwiązanie wyraźnie „zjeżdża" się do zera (maksymalna różnica: z 1.0 zrobiło się 0.13...).
Precyzja podwójna jest w przypadku macierzy 8 x 8 w zupełności wystarczająca (tak jak w przypadku metody Gaussa).

Podobnie jak poprzednio zrobiłem jeszcze testy dla małych macierzy przy pojedynczej precyzji:

```
macierz A:
1.000000 0.500000 0.333333 0.250000
0.500000 0.333333 0.250000 0.200000
0.333333 0.250000 0.200000 0.166667
0.250000 0.200000 0.166667 0.142857

wektor x:
-1.000000 1.000000 -1.000000 1.000000

wektor b (Ax):
-0.583333 -0.216667 -0.116667 -0.073810

wektor x (jako rozwiazanie ukladu Ax=b):
-1.000003 1.000036 -1.000086 1.000056
```

Wynik jest lepszy niż dla metody Gaussa: odchylenia dopiero na piątym miejscu po przecinku.

Duża macierz (10 x 10) i podwójna precyzja:

```
macierz A:
1.000000 0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000
0.500000 0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909
0.333333 0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333
0.250000 0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923
0.200000 0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429
0.166667 0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667
0.142857 0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667 0.062500
0.125000 0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667 0.062500 0.058824
0.111111 0.100000 0.090909 0.083333 0.076923 0.071429 0.066667 0.062500 0.058824 0.055556
0.100000 0.090909 0.083333 0.076923 0.071429 0.066667 0.062500 0.058824 0.055556 0.052632

wektor x:
-1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000 -1.000000 1.000000

wektor b (Ax):
-0.645635 -0.263456 -0.153211 -0.103200 -0.075372 -0.057961 -0.046205 -0.037828 -0.031616 -
0.026864

wektor x (jako rozwiazanie ukladu Ax=b):
-1.000000 1.000000 -1.000000 0.999996 -0.999982 0.999951 -0.999920 0.999923 -0.999959 0.999991
```

Tu, podobnie jak w metodzie Gaussa maksymalne odchylenia dopiero na piątym miejscu po przecinku.

Zadanie 3: Układ liniowy

$$\begin{bmatrix} 10^{-5} & 10^{-5} & 1 \\ 10^{-5} & -10^{-5} & 1 \\ 1 & 1 & 2 \end{bmatrix} x = \begin{bmatrix} 2\cdot 10^{-5} \\ -2\cdot 10^{-5} \\ 1 \end{bmatrix}$$

ma dokładnie jedno rozwiązanie:

$$x_1 = -\frac{1}{1-2\cdot 10^{-5}},\ x_2 = 2,\ x_3 = \frac{10^{-5}}{1-2\cdot 10^{-5}}$$

Rozwiązać ten układ metodą eliminacji Gaussa

a. bez wyboru elementów głównych
b. z pełnym wyborem elementów głównych
c. z pełnym wyborem elementów głównych po skalowaniu $x_3' = 10^5 x_3$ i zrównoważeniu macierzy.

Do rozwiązania tego zadania wykorzystałem znów funkcję **gaussj** z biblioteki *Numerical Recipes*, rozwiązującej układ równań liniowych metodą eliminacji Gaussa z pełnym wyborem elementów głównych. Natomiast do rozwiązania bez wyboru elementów głównych napisałem własne funkcje **gauss** oraz **backsubstitute**.

Treść programu:

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

extern void gaussj(float **a, int n, float **b, int m);

/*
 * Eliminacja Gaussa bez wyboru elementow glownych
 */
void gauss(float **a, int n, float *b)
{
  int i, j, k;

  for(k=1; k<n; k++)
  {
    for(j=k+1; j<=n; j++)
    {
      float scaler = a[j][k]/a[k][k];
      for(i=1; i<=n; i++)
      {
        a[j][i] -= a[k][i]*scaler;
      }
      b[j] -= b[k]*scaler;
    }
  }
}

/*
 * Podstawienie wsteczne
 */
void backsubstitute(float **a, float *b, int n)
{
  int i, j;

  for(j=n; j>=1; j--)
  {
    float r = 0;
    for(i=n; i>j; i--)
      r += a[j][i]*b[i];
    b[j] = (b[j]-r)/a[j][j];
  }
}

/*
 * Czesc glowna programu
 */
void main(void)
{
  float mtx[3][3] =  { { 1.0e-5,  1.0e-5, 1.0 },
                       { 1.0e-5, -1.0e-5, 1.0 },
                      {    1.0,     1.0, 2.0 } };
  float vb[] =         { 2.0e-5, -2.0e-5, 1 };
  int n = 3, i,j;
  float **a = matrix(1, n, 1, n);
  float *b = vector(1, n);
  float **b2 = matrix(1, n, 1, 1);
  int *rows = ivector(1, n);
  int *cols = ivector(1, n);

  /* skopiowanie danych ukladu */
  for(j=1; j<=n; j++)
    for(i=1; i<=n; i++)
      a[j][i] = mtx[j-1][i-1];
  for(i=1; i<=n; i++)
    b[i] = vb[i-1];

  /* wyswietlenie macierzy ukladu */
  printf("A=\n");
  mat_show(a, n);

  /* wyswietlenie prawej strony ukladu */
  printf("\nb=");
  vec_show(b, n);

  /* rozwiazanie ukladu metoda eliminacji gaussa */
  gauss(a, n, b);
  backsubstitute(a, b, n);
```

```
  /* wyswietlenie rozwiazania */
  printf("\nbez wyboru elementow glownych:\n");
  vec_show(b, n);

  /* skopiowanie danych ukladu */
  for(j=1; j<=n; j++)
    for(i=1; i<=n; i++)
      a[j][i] = mtx[j-1][i-1];
  for(i=1; i<=n; i++)
    b2[i][1] = vb[i-1];

  /* rozwiazanie z pelnym wyborem el. glownych */
  gaussj(a, n, b2, 1);

  /* wypisanie rozwiazania */
  printf("\npelny wybor elementow glownych:\n");
  vec_show2(b2, n);

  /* skopiowanie danych ukladu */
  for(j=1; j<=n; j++)
    for(i=1; i<=n; i++)
      a[j][i] = mtx[j-1][i-1];
  for(i=1; i<=n; i++)
    b2[i][1] = vb[i-1];

  /* skalowanie */
  for(j=1; j<=n; j++)
    a[j][3] *= 1.0e-5;

  /* rozwiazanie z pelnym wyborem el. glownych */
  gaussj(a, n, b2, 1);

  b2[3][1] *= 1.0e-5;

  /* wypisanie rozwiazania */
  printf("\nskalowanie + pelny wybor elementow glownych:\n");
  vec_show2(b2, n);
}
```

Wynik działania programu:

```
A=
| 0.000010     0.000010       1.000000       |
| 0.000010     -0.000010      1.000000       |
| 1.000000     1.000000       2.000000       |

b=0.000020 -0.000020 1.000000

bez wyboru elementow glownych:
-1.000020 2.000000 0.000010

pelny wybor elementow glownych:
-0.998522 1.998502 0.000010

skalowanie + pelny wybor elementow glownych:
-1.000020 2.000000 0.000010
```

Jak widać w przypadku tej macierzy samo wyszukiwanie elementów głównych nic nie daje, a wręcz pogarsza rozwiązanie. Poprawne rozwiązanie otrzymujemy jeśli dokonamy skalowania macierzy. Co ciekawe, bez wyboru elementów głównych wynik jest również poprawny (obliczenia wykonywane były w precyzji pojedynczej, w przypadku precyzji podwójnej brak było bowiem jakichś ciekawych zachowań (wszystkie rozwiązania były poprawne)).