Metody numeryczne

Wojciech Chrobak

20 listopada 2017

Zadanie 2 - obowiązkowe

Kod

```
#include <iostream>
#include <cmath>
#include <vector>
#include <iomanip>

using namespace std;


const int sizeMatrix = 128;
double D = 4;
double E = 1;
double F = 1;
double G = 1;
double H = 1;

vector<double> b;

void printX(vector<double> A) {
  for (int i = 0; i < A.size(); i++) {
    cout << A[i] << "\\\\\n";
  }
  cout << endl;
}

double norm(vector<double> u) {
  double a = 0;
  double norm;
  for (int i = 0; i < u.size(); ++i) {
    a += u[i] * u[i];
  }
  norm = sqrt(a);
  return norm;
}

vector<double> subtract(vector<double> A, vector<double> B) {
  vector<double> res;
  res.assign(A.size(), 0);

  for (int i = 0; i < res.size(); i++) {
    res[i] = A[i] - B[i];
  }

  return res;
}

```

```
double multiVV(vector<double> A, vector<double> B) {
  double res = 0;
  for (int i = 0; i < A.size(); i++) {
    res = res + A[i] * B[i];
  }
  return res;
}

// metoda Gaussa-Seidela
vector<double> GS() {
  vector<double> x;
  x.assign(sizeMatrix,0);

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

  //////////////////////////////////////////////////

  int counter = 0;
  while(true) {

    temp_x = x; // poprzednie wartosci przyblizen

    for (int i = 0; i < sizeMatrix; ++i) {
      double sumL = 0;
      double sumR = 0;
      // lewa strona sumy
      if(i > 0) {
        sumL = F * x[i-1];
        if(i>=4) {
          sumL = sumL + H * x[i-4];
        }
      }

      // prawa strona sumy
      if(i < sizeMatrix-1) {
        sumR = E * temp_x[i+1];
        if(i < sizeMatrix-4) {
          sumR = sumR + G * temp_x[i+4];
        }
      }

      x[i] = (b[i] - sumL - sumR)/D;
    }
    if( norm(subtract(x, temp_x)) < 0.000001)
      break;
    counter++;
  }
  return x;
}

// metoda gradientow sprzezonych
vector<double> CG() {
  vector<double> x;
  x.assign(sizeMatrix, 0);

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

  vector<double> r = b; // r_k+1
  vector<double> temp_r = r; // r_k

  vector<double> p = b; // p_k+1
  vector<double> temp_p = p; // p_k
```

```

    double alfa = 0;
    double beta = 0;

    vector<double> Ap; // Ap = A*p
    Ap.assign(sizeMatrix,0);


    // /////////////////////////////////////////////////////////////////////////////////////////////


    int counter = 0;
    while (true) {

      temp_x = x; // poprzednie wartosci przyblizen
      temp_r = r;
      temp_p = p;

      ////////////////////////////////////////
      // alfa //

      // licze Ap
      for (int i = 0; i < sizeMatrix; ++i) {
        if (i == 0) {
          Ap[i] = temp_p[i] * D + temp_p[i + 1] * F + temp_p[i + 4] * H;
        }
        if (i > 0 && i < 4) {
          Ap[i] = temp_p[i] * D + temp_p[i - 1] * E + temp_p[i + 1] * F + temp_p[i + 4] * H;
        }
        if (i >= 4 && i < sizeMatrix - 4) {
          Ap[i] = temp_p[i] * D + temp_p[i - 1] * E + temp_p[i - 4] * G + temp_p[i + 1] * F + temp_p[i + 4] * H;
        }
        if (i >= sizeMatrix-4 && i < sizeMatrix - 1) {
          Ap[i] = temp_p[i] * D + temp_p[i - 1] * E + temp_p[i - 4] * G + temp_p[i + 1] * F;
        }
        if (i == sizeMatrix - 1) {
          Ap[i] = temp_p[i] * D + temp_p[i - 1] * E + temp_p[i - 4] * G;
        }
      }

      alfa = multiVV(temp_r, temp_r) / multiVV(temp_p, Ap);

      ////////////////////////////////////////

      for (int i = 0; i < sizeMatrix; ++i) {
        x[i] = temp_x[i] + alfa * temp_p[i];
      }

      ////////////////////////////////////////

      for (int i = 0; i < sizeMatrix; ++i) {
        r[i] = temp_r[i] - alfa * Ap[i];
      }

      ////////////////////////////////////////

      beta = multiVV(r,r) / multiVV(temp_r, temp_r);

      ////////////////////////////////////////
```

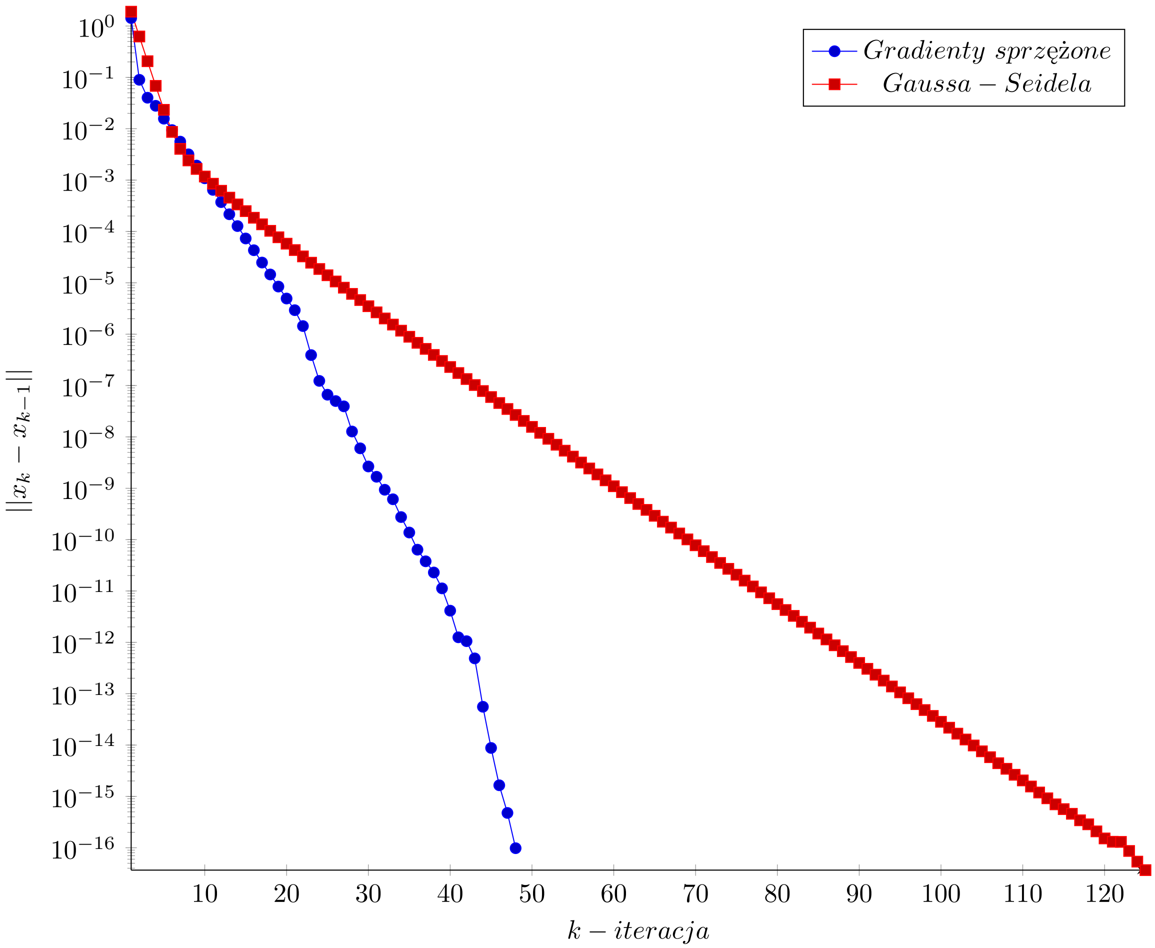
```
    for (int i = 0; i < sizeMatrix; ++i) {
      p[i] = r[i] + beta * temp_p[i];
    }

    if( norm(subtract(x, temp_x)) < 0.000001)
      break;
    counter++;
  }
  return x;
}

int main() {
  cout.setf(ios::fixed, ios::floatfield);
  cout.precision(30);

  // wyrazy wolne
  b.assign(sizeMatrix, 1);


  vector<double> x1;
  vector<double> x2;

  x1 = GS();
  x2 = CG();

  return 0;
}
```

Wynik

Wypisuję tylko kilka dla sprawdzenia wyniku.

$$\begin{array}{cc} \text{Gaussa-Seidela} & \text{Gradientów sprzężonych} \\ x = \begin{bmatrix} 0.194276795444196875850551009535 \\ 0.130930202478144686040195665555 \\ 0.146794908343523766713900613468 \\ ... \\ 0.146794908343523738958324997839 \\ 0.130930202478144686040195665555 \\ 0.194276795444196875850551009535 \end{bmatrix} & x = \begin{bmatrix} 0.194276795444196792583824162648 \\ 0.130930202478144630529044434297 \\ 0.146794908343523738958324997839 \\ ... \\ 0.146794908343523738958324997839 \\ 0.130930202478144658284620049926 \\ 0.194276795444196764828248547019 \end{bmatrix} \end{array}$$

Zależność normy



W metodzie Gaussa-Seidela $||x_k - x_{k-1}||$ potrzebujemy dużo więcej kroków iteracji niż w metodzie gradientów sprzężonych aby otrzymać przybliżone wyniki z oczekiwaną dokładnością.

Przykładowo, jeśli ustalimy $\epsilon = 10^{-6}$ dla naszej macierzy to musimy wykonać 35 iteracji algorytmu Gaussa-Seidela. Metoda gradientów sprzężonych jest dużo szybsza i oczekiwane przybliżenie otrzymamy już po 23 krokach iteracji.

Macierz A po rozkładzie Cholesky'ego nie zmienia swojej postaci i nadal jest to macierz z 4 dodatkowymi diagonalami. Zatem koszt takiego rozkładu wynosi $O((2P+1)N) = O(9N)$