Bogdan Chwaliński
Zestaw 2
Zadanie 5N a,b

Opis:

W zadaniu mieliśmy rozwiązać rownienia $Ax = e$ za pomocą metody Gaussa-Seidela oraz za pomocą gradientów sprzężonych. Otrzymane poniżej wyniki zostały wypisane po 50 iteracja dla każej z metody.

Nie stworzyłem wykresu ale sprawdzając program dla kilku różnych wartości iteracji, metoda gradientów sprzężonych jest szybciej zbieżna do poprawnych wyników od metody Gaussa-Seidela.

Jeżeli jest to niezbędne aby uzyskać jakiekolwiek pkt z tych zadań postaram się zaimplementować ten wykres np. w Mathematice (tylko będę musiał nad tym długo posiedzieć), ale zrobie co w mojej mocy.

Wyniki:

Wyniki Gaussa Seidel'a znajdują się pod [tym] linkiem .

Wyniki Gradientów sprzężonych znajdują się pod [tym] linkiem.

Kody programów:

Gradienty sprzężone lub pod [tym] linkiem

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

#define N 100                   //rozmiar macierzy, wektora itd.
#define iteracje 50             //liczba iteracji

using namespace std;

int main(){
        double A[N][N], x[N], e[N], norma[N], p[N], r[N], Ap[N], rr[N], pp[N];

        //uzupelnienie danych wejsciowych
        for(int i=0; i<N; i++){
                x[i]=e[i] = norma[i] = 1;

                        for(int j=0; j<N; j++){
                                if(i==j){
                                        A[i][j] = 4;
                                        if(j<N-1) A[i+1][j] = A[i][j+1] = 1;
                                        if(j<N-4) A[i+4][j] = A[i][j+4] = 1;
                                }
                        }
        }

        //implementacja algorytmu
        r[0] = e[0] + (-A[0][0] * x[0] - A[0][1] * x[1] - A[0][4] * x[4]);
        r[1] = e[1] + (-A[1][0] * x[0] - A[1][1] * x[1] - A[1][2] * x[2] - A[1][5] *
x[5]);
        r[2] = e[2] + (-A[2][1] * x[1] - A[2][2] * x[2] - A[2][3] * x[3] - A[2][6] *
x[6]);
        r[3] = e[3] + (-A[3][2] * x[2] - A[3][3] * x[3] - A[3][4] * x[4] - A[3][7] *
x[7]);
```

```
                for(int j=4; j<N-4; j++){
                        r[j] = e[j]-(A[j][j-4] * x[j-4] + A[j][j-1] * x[j-1] + A[j][j] *
x[j] + A[j][j+1] * x[j+1] + A[j][j+4] * x[j+4]);
                }

        r[N-4] = e[N-4] + (-A[N-4][N-8] * x[N-8] - A[N-4][N-5] * x[N-5] - A[N-4][N-4] *
x[N-4] - A[N-4][N-3] * x[N-3]);
        r[N-3] = e[N-3] + (-A[N-3][N-7] * x[N-7] - A[N-3][N-4] * x[N-4] - A[N-3][N-3] *
x[N-3] - A[N-3][N-2] * x[N-2]);
        r[N-2] = e[N-2] + (-A[N-2][N-6] * x[N-6] - A[N-2][N-3] * x[N-3] - A[N-2][N-2] *
x[N-2] - A[N-2][N-1] * x[N-1]);
        r[N-1] = e[N-1] + (-A[N-1][N-5] * x[N-5] - A[N-1][N-2] * x[N-2] - A[N-1][N-1] *
x[N-1]);

                for(int i=0; i<N; i++){
                        p[i] = r[i];
                }

                for(int i=0; i<iteracje; i++){
                        Ap[0] = A[0][0] * p[0] + A[0][1] * p[1] + A[0][4] * p[4];
                        Ap[1] = A[1][0] * p[0] + A[1][1] * p[1] + A[1][2] * p[2] + A[1]
[5] * p[5];
                        Ap[2] = A[2][1] * p[1] + A[2][2] * p[2] + A[2][3] * p[3] + A[2]
[6] * p[6];
                        Ap[3] = A[3][2] * p[2] + A[3][3] * p[3] + A[3][4] * p[4] + A[3]
[7] * p[7];

                                for(int j=4; j<N-4; j++){
                                        Ap[j] = A[j][j-4] * p[j-4] + A[j][j-1] * p[j-1] +
A[j][j] * p[j] + A[j][j+1] * p[j+1] + A[j][j+4] * p[j+4];
                                }

                        Ap[N-4] = A[N-4][N-8] * p[N-8] + A[N-4][N-5] * p[N-5] + A[N-4][N-
4] * p[N-4] + A[N-4][N-3] * p[N-3];
                        Ap[N-3] = A[N-3][N-7] * p[N-7] + A[N-3][N-4] * p[N-4] + A[N-3][N-
3] * p[N-3] + A[N-3][N-2] * p[N-2];
                        Ap[N-2] = A[N-2][N-6] * p[N-6] + A[N-2][N-3] * p[N-3] + A[N-2][N-
2] * p[N-2] + A[N-2][N-1] * p[N-1];
                        Ap[N-1] = A[N-1][N-5] * p[N-5] + A[N-1][N-2] * p[N-2] + A[N-1][N-
1] * p[N-1];

                double a, b, l, m;
                a=b=l=m=0;

                        for(int j=0; j<N; j++){
                                l = l + r[j] * r[j];
                                m = m + p[j] * Ap[j];
                        }

                a = l/m;

                        for(int j=0; j<N; j++){
                                rr[j] = r[j] - a * Ap[j];
                                x[j] = x[j] + a * p[j];
                        }

                m = l;
                l = 0;

                        for(int j=0; j<N; j++){
                                l = l + rr[j] * rr[j];
                        }

                b = l/m;

                        for(int j=0; j<N; j++){
                                pp[j] = rr[j] + b * p[j];
                        }

                        for(int j=0; j<N; j++){
```

```
                                p[j] = pp[j];
                                r[j] = rr[j];
                        }
                }

        //wypisanie wynikow
        for(int i=0; i<N; i++){
                cout << "x" << i+1 << " = " << x[i] << endl;
        }

return 0;
}
```

Gauss Seilder lub pod [tym] linkiem

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

#define N 100                   //rozmiar macierzy, wektora itd.
#define iteracje 50             //liczba iteracji
using namespace std;

int main(){
        double A[N][N], x[N], e[N], norma[N]; //dane wejsciowe
        //uzupelnienie danych wejsciowych
        for(int i=0; i<N; i++){
                x[i]=e[i] = norma[i] = 1;

                        for(int j=0; j<N; j++){
                                if(i==j){
                                        A[i][j] = 4;
                                        if(j<N-1) A[i+1][j] = A[i][j+1] = 1;
                                        if(j<N-4) A[i+4][j] = A[i][j+4] = 1;
                                }
                        }
        }

        //implementacja algorytmu
        for(int i=0; i<iteracje; i++){
                x[0] = (e[0] - A[0][1] * x[1] - A[0][4] * x[4]) / 4;
                x[1] = (e[1] - A[1][0] * x[0] - A[1][2] * x[2] - A[1][5] * x[5]) / 4;
                x[2] = (e[2] - A[2][1] * x[1] - A[2][3] * x[3] - A[2][6] * x[6]) / 4;
                x[3] = (e[3] - A[3][2] * x[2] - A[3][4] * x[4] - A[3][7] * x[7]) / 4;

                for(int j=4; j<N-4; j++){
                        x[j] = (e[j] - A[j][j-4] * x[j-4] - A[j][j-1] * x[j-1] - A[j]
[j+1] * x[j+1] - A[j][j+4] * x[j+4]) / 4;
                }

                        x[N-4] = (e[N-4] - A[N-4][N-8] * x[N-8] - A[N-4][N-5] * x[N-5] -
A[N-4][N-3] * x[N-3]) / 4;
                        x[N-3] = (e[N-3] - A[N-3][N-7] * x[N-7] - A[N-3][N-4] * x[N-4] -
A[N-3][N-2] * x[N-2]) / 4;
                        x[N-2] = (e[N-2] - A[N-2][N-6] * x[N-6] - A[N-2][N-3] * x[N-3] -
A[N-2][N-1] * x[N-1]) / 4;
                        x[N-1] = (e[N-1] - A[N-1][N-5] * x[N-5] - A[N-1][N-2] * x[N-2]) /
4;
        }

        //wypisanie wynikow
        for(int i=0; i<N; i++){
                cout << "x" << i+1 << " = " << x[i] << endl;
        }
```

```
return 0;
}
```