Metody numeryczne - N6

DAMIAN PORADYŁO

N6 *Zadanie numeryczne*
Zaimplementować metodę gradientów sprzężonych dla układu z zadania 6 z poprzedniego zestawu.

6. Dla układu równań $Ax = b$, gdzie

$$A = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 6 \\ 2 \end{pmatrix}$$

1 OPIS METODY

Metodę gradientów sprzężonych możemy zastosować, gdy:

- macierz $A \epsilon R^{NxN}$ jest symetryczna
- x_1 – początkowe przybliżenie rozwiązania równania $0 < \varepsilon \leq 1$

Wówczas mamy poniższy algorytm:

Algorytm:

$r_0 := b - Ax_0$
$p_0 := r_0$
$k := 0$
while $\|\mathbf{r}_k\| > \varepsilon$

$$\alpha_k := \frac{r_k^\top r_k}{p_k^\top A p_k}$$

$$x_{k+1} := x_k + \alpha_k p_k$$

$$r_{k+1} := r_k - \alpha_k A p_k$$

$$\beta_k := \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$$

$$p_{k+1} := r_{k+1} + \beta_k p_k$$

$$k := k + 1$$

W zadaniu była podana macierz 3x3, jak widać jest to macierz trójdiagonalną. Aby nie iterować po całej macierzy kwadratowej (chociaż w przypadku macierzy tak małych rozmiarów nie ma to większego znaczenia) zapisałem ją w sposób właśnie trójdiagonalny (w postaci 3 wektorów). Dzięki temu uniknąłem niepotrzebnego mnożenia przez 0.

W arytmetyce dokładnej metoda zbiega się po N krokach, zatem jej koszt wynosi O(N · koszt jednego kroku). Należy zwrócić uwagę, że koszt jednego kroku jest tutaj zdominowany przez obliczanie iloczynu Ap_k

2 Wyniki

Wynikiem jest zbiór rozwiązań:

$$x = \begin{bmatrix} 1.0 \\ 2.0 \\ 1.0 \end{bmatrix}$$

Rozwiązanie otrzymujemy już w **drugiej** iteracji.

3 KOD PROGRAMU

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

#define precision 1e-10

double getActualNorm(std::vector<double> v) {
    double result = 0;
    for(int i=0; i<v.size(); i++) {
        result += v[i] * v[i];
    }
    return std::sqrt(result);
}
std::vector<double> multiplyVector(std::vector<double> A1, std::vector<double> A2, std::vector<double>
A3, std::vector<double> y) {

    std::vector<double> returnVector(y.size(), 0.0);
    returnVector[0] = ((A2[0] * y[0]) + (A1[0] * y[1]));
    returnVector[1] = ((A3[0] * y[0]) + (A2[1] * y[1]) + (A1[1] * y[2]));
    returnVector[2] = ((A3[1] * y[1]) + (A2[2] * y[2]));

    return returnVector;
}
double multiplyVectorT(std::vector<double> x, std::vector<double> y) {
    double result = 0;
    for(int i=0; i<x.size(); i++) {
        result += x[i] * y[i];
    }
    return result;
}
std::vector<double> multiplyVectorScalar(double x, std::vector<double> y) {
    std::vector<double> returnVector(y.size(), 0.0);
    for(int i=0; i<y.size(); i++) {
        returnVector[i] = x * y[i];
    }
    return returnVector;
}
std::vector<double> addVector(std::vector<double> x, std::vector<double> y) {
    std::vector<double> returnVector(x.size(), 0.0);
    for(int i=0; i<x.size(); i++) {
        returnVector[i] = x[i] + y[i];
    }
    return returnVector;
}
std::vector<double> substractVector(std::vector<double> x, std::vector<double> y) {
    std::vector<double> returnVector(x.size(), 0.0);
    for(int i=0; i<x.size(); i++) {
        returnVector[i] = x[i] - y[i];
    }
    return returnVector;
}

std::vector<double> solve(std::vector<double> A1, std::vector<double> A2, std::vector<double> A3,
std::vector<double> b) {
    std::vector<double> r(b.size(), 0.0);
    std::vector<double> x(b.size(), 1.0);

    r = substractVector(b, multiplyVector(A1, A2, A3, x));
    std::vector<double> p(r);
    int iterator = 0;

    while(getActualNorm(r) > precision) {
        double rt_r = multiplyVectorT(r,r);
        std::vector<double> Ap = multiplyVector(A1, A2, A3, p);
        double alfa = rt_r / multiplyVectorT(p, Ap);

        r = substractVector(r, multiplyVectorScalar(alfa, Ap));
        double beta = multiplyVectorT(r,r) / rt_r;
        x = addVector(x, multiplyVectorScalar(alfa, p));
        p = addVector(r, multiplyVectorScalar(beta, p));
        iterator++;
    }
    return x;
}
int main() {
    std::vector<double> A1 {-1.0, -1.0};
    std::vector<double> A2 {4.0, 4.0, 4.0};
    std::vector<double> A3 {-1.0, -1.0};

    std::vector<double> b {2.0, 6.0, 2.0};
    std::vector<double> wynik = solve(A1, A2, A3, b);

    for(int i=0; i<wynik.size(); i++) {
        std::cout <<  std::fixed << std::setprecision(10) << wynik[i] << " \n";
    }
}
```

4 URUCHOMIENIE

>> **make run**

W skład zestawu wchodzą:

- Plik programu
- Plik zawierający otrzymane wyniki
- Opracowanie
- Makefile