

## Laboratorium 4

Artem Buhera  
GĆ01 135678

27.03.2021

W tym programie użyjemy uogólnionej metody Newtona do rozwiązywania poniższego układu równań:

$$\begin{cases} x^2 + y^2 + z^2 = 2, \\ x^2 + y^2 = 1, \\ x^2 = y \end{cases} \Leftrightarrow \begin{cases} x^2 + y^2 + z^2 - 2 = 0, \\ x^2 + y^2 - 1 = 0, \\ x^2 - y = 0 \end{cases}$$

Ustawmy, że musimy rozwiązać układ równań w poniższej postaci:

$$\vec{f}(\vec{x}) = \vec{0}, \text{ gdzie } \vec{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \vec{f}(\vec{x}) = \begin{bmatrix} f_1(\vec{x}) \\ f_2(\vec{x}) \\ f_3(\vec{x}) \end{bmatrix} = \begin{bmatrix} x^2 + y^2 + z^2 - 2, \\ x^2 + y^2 - 1, \\ x^2 - y \end{bmatrix}, \vec{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Na ćwiczeniach znaleźliśmy do tego układu rozwiązanie w postaci równania

$\vec{x}_{n+1} = \vec{x}_n - \vec{\delta}_{n+1}$ , gdzie  $\vec{x}_{n+1}, \vec{x}_n$  - to aktualne oraz następne przybliżenia liczonych rozwiązań, a  $\vec{\delta}_{n+1}$  - wektor poprawkowy do obliczania kolejnego przybliżenia

$$\vec{x}_{n+1} = \begin{bmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \end{bmatrix}, \vec{x}_n = \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix}, \vec{\delta}_{n+1} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix} = \begin{bmatrix} \frac{2x_n^2 y_n + x_n^2 - y_n^2 - 1}{2x_n(2y_n + 1)} \\ \frac{y_n^2 + y_n - 1}{2y_n + 1} \\ \frac{z_n}{2} - \frac{1}{2z_n} \end{bmatrix}$$

Obliczanie będziemy zaczynać od pewnych początkowych wyrazów  $\vec{x}_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} 7.5 \\ 5 \\ 2.5 \end{bmatrix}$ ,

a iteracje kończyć w przypadku zajścia przynajmniej jednego z warunków:

- przekroczenie maksymalnej liczby iteracji  $n_{max} = 30$
- estymator błędów, ustalony jako największa wartość absolutna z różnic między obliczonymi miejscami zerowymi w dwóch kolejnych iteracjach dla każdej ze zmiennych, czyli normę maksimum  $e_n = \|\vec{x}_{n+1} - \vec{x}_n\|_\infty$ , jest na moduł mniejsze od tolerancji  $TOLX = 2^{-52}$
- podobnie residuum równań  $\|\vec{f}(\vec{x}_{n+1})\|_\infty$  jest na moduł mniejsze od tolerancji  $TOLF = 2^{-52}$

```

const int NMAX = 30;
const double TOLX = DBL_EPSILON;
const double TOLF = DBL_EPSILON;

double f1(vector<double> xn) {
    double x = xn[0], y = xn[1], z = xn[2];
    return x*x + y*y + z*z - 2.0;
}

double f2(vector<double> xn) {
    double x = xn[0], y = xn[1];
    return x*x + y*y - 1.0;
}

double f3(vector<double> xn) {
    double x = xn[0], y = xn[1];
    return x*x - y;
}

double delta1(vector<double> xn) {
    double x = xn[0], y = xn[1];
    return (2.0*x*x*y + x*x - y*y - 1.0) / (2.0*x*(2.0*y + 1.0));
}

double delta2(vector<double> xn) {
    double y = xn[1];
    return (y*y + y - 1.0) / (2.0*y + 1.0);
}

double delta3(vector<double> xn) {
    double z = xn[2];
    return z/2.0 - 1.0/(2.0*z);
}

```

```

// pomocnicze funkcje do printowania
void printHeader(int vector_size) {
    cout << "  n";
    for (int i = 0; i < vector_size; i++)
        cout << setw(20) << "xn_" << i;
    cout << setw(26) << "estymator" << setw(26) << "residuuum" << endl;
}

void printIteration(int vector_size, const vector<double> &xn1, double max_residuuum,
double max_estimator, int n) {
    cout << setw(3) << n << "  ";
    for (int i = 0; i < vector_size; i++)
        cout << setw(19) << setprecision(17) << xn1[i] << "  ";
    cout << setw(24) << max_estimator << "  " << setw(24) << max_residuuum << endl;
}

void printTOLX(int n, double estimator) {
    cout << endl << "przekroczono TOLX po " << n
        << " iteracjach - max estymator = " << estimator << endl << endl;
}

void printTOLF(int n, double residuum) {
    cout << endl << "przekroczono TOLF po " << n
        << " iteracjach - max residuum = " << residuum << endl << endl;
}

void printTOLXAndTOLF(int n, double estimator, double residuum) {
    cout << endl << "przekroczono TOLX oraz TOLF po " << n
        << " iteracjach - max estymator = " << estimator
        << ", max residuum = " << residuum << endl << endl;
}

void printNMAX() {
    cout << "przekroczono maksymalną liczbę iteracji - " << NMAX << endl << endl;
}

```

```

vector<double> solve_system(vector<double> x0, vector<double (*) (vector<double>)> delta,
                           vector<double (*) (vector<double>)> f, int vector_size) {

    // wektor obliczonych miejsc zerowych układu równań f
    vector<double> xn1(vector_size);
    vector<double> xn = x0;

    vector<double> residuum(vector_size);
    vector<double> estimator(vector_size);
    double max_residuum;
    double max_estimator;

    printHeader(vector_size);

    // arbitralne ograniczenie na liczbę iteracji
    for (int n = 1; n ≤ NMAX; n++) {
        // iterowanie po elementach wektora xn
        for (int i = 0; i < vector_size; i++) {
            xn1[i] = xn[i] - delta[i](xn);

            estimator[i] = abs(xn1[i] - xn[i]);
            residuum[i] = abs(f[i](xn1));
        }

        // wyznaczenie największego estymatora i residuum
        max_estimator = *max_element(estimator.begin(), estimator.end());
        max_residuum = *max_element(residuum.begin(), residuum.end());

        printIteration(vector_size, xn1, max_residuum, max_estimator, n);

        // kryterium dokładności wyznaczenia xn1
        // oraz wiarygodności xn1 jako przybliżenia pierwiastków
        if (max_estimator ≤ TOLX && max_residuum ≤ TOLF) {
            printTOLXAndTOLF(n, max_estimator, max_residuum);
            return xn1;
        } else {
            // kryterium dokładności wyznaczenia xn1
            if (max_estimator ≤ TOLX) {
                printTOLX(n, max_estimator);
                return xn1;
            }

            // kryterium wiarygodności xn1 jako przybliżenia pierwiastków
            if (max_residuum ≤ TOLF) {
                printTOLF(n, max_residuum);
                return xn1;
            }
        }

        xn = xn1;
    }

    printNMAX();
    return xn1;
}

```

```

int main() {
    int vector_size = 3;

    vector<double> xn = {7.5, 5.0, 2.5};
    vector<double (*) (vector<double>)> delta = {delta1, delta2, delta3};
    vector<double (*) (vector<double>)> f = {f1, f2, f3};

    vector<double> roots = solve_system(xn, delta, f, vector_size);

    for (int i = 0; i < vector_size; i++)
        cout << "x_" << i << " = " << roots[i] << endl;
}

```

Wynik działania programu:

n	xn_0	xn_1	xn_2	max estymator	max residuum
1	3.9075757575757577	2.3636363636363638	1.45	3.5924242424242423	19.85592516069789
2	2.1009471690258819	1.15007215007215	1.0698275862068964	1.8066285885498758	10.103255866542
3	1.2179712259450848	0.70380739117074942	1.0022788213065104	0.88297594308079708	1.9506509218110248
4	0.86395458461746077	0.62108974194463751	1.000002590609737	0.354016641327624	0.24632520378767309
5	0.78965706175822636	0.61803815328161693	1.0000000000033555	0.074297522859234411	0.0093159239596767307
6	0.78615915950766957	0.61803398875765092	1	0.0034979022505567947	1.7382996268189288e-05
7	0.78615137779593691	0.6180339887498949	1	7.7817117326572927e-06	7.014211433897799e-11
8	0.78615137775742328	0.6180339887498949	1	3.851363672424668e-11	1.1102230246251565e-16

przekroczono TOLF po 8 iteracjach - max residuum = 1.1102230246251565e-16

```

x_0 = 0.78615137775742328
x_1 = 0.6180339887498949
x_2 = 1

```

Uzyskaliśmy wystarczająco dokładne wyniki po dosyć małej ilości iteracji. Wnioskujemy z tego, że metoda jest efektywna i poprawnie zaimplementowana. Możemy też zaobserwować zbieżność estymatora błędów oraz residuum rozwiązania układu równań.