

Wydział WIMiIP	Imię i nazwisko Mateusz Witkowski	Rok II	Grupa 4
Temat: Rozwiązywanie równań nieliniowych metodą bisekcji i Newtona-Raphsona.			Prowadzący dr hab. inż. Hojny Marcin, prof. AGH
Data ćwiczenia 30.04.2020	Data oddania 07.05.2020	Data zaliczenia	OCENA

1. Cel ćwiczenia

Celem ćwiczenia było zapoznanie się oraz implementacja metod rozwiązywania równań nieliniowych za pomocą metody bisekcji i Newtona-Raphsona.

2. Wprowadzenie teoretyczne

Często występującym, ważnym problemem obliczeniowym jest numeryczne poszukiwanie rozwiązań równań nieliniowych, np. algebraicznych (wielomiany), przestępnych (funkcje trygonometryczne), które przybierają postać:

$$f(x) = 0$$

x – jest zmienną skalarną (w równaniu jednej zmiennej) lub wektorem wymiaru (w równaniu o więcej niż jednej zmiennej).

Liczba spełniająca to równanie jest rozwiązaniem (bądź pierwiastkiem) równania.

Charakterystyczną cech tych równań jest to, że może nie istnieć rozwiązanie lub jest ich nieskończenie wiele, z czego wynikają niejednoznaczne reguły postępowania przy ich rozwiązywaniu. Jednymi ze sposobów wyznaczania tego typu równań są metody bisekcji i Newtona-Raphsona.

Metoda bisekcji – nazywana również metodą połowienia jest najprostszą możliwą metodą. W pierwszej kolejności wybieramy przedział do analizy i na podstawie jego granic wyznaczany jest punkt x_0 , taki że:

$$x_0 = \frac{a + b}{2}$$

a, b – granice przedziału.

Jeśli wyznaczony punkt okaże się miejscem zerowym algorytm zakończy działanie. W przeciwnym wypadku przedział zostanie podzielony na dwie części oraz algorytm sprawdzi, która z połów spełnia następujący warunek istnienia rozwiązania:

$$f(a) \cdot f(b) < 0$$

Przedział spełniający ten warunek zostaje przedziałem, w którym należy kontynuować poszukiwanie rozwiązania. Zakończenie algorytmu jest równoznaczne z osiągnięciem żądanej dokładności.

Istnieją warunki zastosowania tej metody:

- Funkcja f oraz jej pierwsza i druga pochodna są ciągłe w analizowanym przedziale $[a, b]$
- Różne znaki na krańcach przedziału: $f(a) \cdot f(b) < 0$
- Wewnątrz $[a, b]$ znajduje się dokładnie jeden pierwiastek

Metoda **Newtona-Raphsona** – nazywana również metodą stycznych. Warunki potrzebne do zastosowania nie różnią się od tych w poprzedniej metodzie:

- Funkcja f oraz jej pierwsza i druga pochodna f mają stały znak w przedziale $[a, b]$
- Różne znaki na krańcach przedziału: $f(a) \cdot f(b) < 0$
- Wewnątrz $[a, b]$ znajduje się dokładnie jeden pierwiastek

Metodę tę rozpoczynamy od wyznaczenia punktu startowego – zwykle wartości 0, 1, lub jednego z krańców przedziałów. Kolejnym krokiem jest wyprowadzenie stycznej w $f(x_1)$, której punkt przecięcia z osią O_x jest pierwszym przybliżeniem rozwiązania (x_2). W przypadku jeśli otrzymane rozwiązanie nie spełnia warunku stopu, czyli wartość $f(x_2)$ jest daleka od zera, algorytm wykonuje kolejną iterację. Punkt przecięcia stycznej z osią O_x zostaje nowym punktem startowym i algorytm wykonuje się ponownie według wzoru:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Kryteria określające moment zatrzymania algorytmu:

- Wartość funkcji w wyznaczonym punkcie jest bliska 0:
 $|f(x_n)| \leq \text{zadana_dokładność}$
- Odległość między kolejnymi przybliżeniami jest wystarczająco mała:
 $|x_{n+1} - x_n| \leq \text{zadana_dokładność}$

3. Kod programu

Zdefiniowano globalne zmienne odpowiedzialne za dokładność obliczeń i liczbę wyświetlanych miejsc po przecinku oraz pomocnicze funkcje mające na celu obliczanie wartości zadanej funkcji oraz jej pochodnej w przekazanym punkcie. W funkcjach wykorzystano metodę pow() skracając zapis niektórych wzorów funkcji.

```
double Dokladnosc = 0.0000001; //Dokladnosc obliczen
int prec = 11; //Liczba wyswietlanych miejsc po przecinku

double f(double x) { //Pomocnicza funkcja obliczajaca wartosc w punkcie x
    return 3*pow(x,5) - 4*pow(x,4) + x*x - 7*x + 3;
}

double d(double x) { //Pomocnicza funkcja obliczajaca wartosc pochodnej w punkcie x
    return 15*pow(x,4) - 16*pow(x,3) + 2*x - 7;
}
```

Rysunek 1. Globalne zmienne, funkcje pomocnicze.

W funkcji main zadeklarowano odpowiednie zmienne i pobrano wartości granic przedziałów i punktu początkowego oraz wywołano odpowiednie funkcje.

```
int main() {  
    double a, b;  
    double start;  
  
    cout << "Obliczanie pierwiastka funkcji" << endl;  
    cout << endl;  
    cout << "-----" << endl;  
    cout << "Metoda Bisekcji" << endl;  
    cout << "Podaj zakres poszukiwan pierwiastka: " << endl;  
    cout << "a = "; cin >> a;    //Pobranie granic przedzialu  
    cout << "b = "; cin >> b;  
    Bisekcja(a, b, Dokladnosc);  
  
    cout << endl;  
    cout << "-----" << endl;  
    cout << "Metoda Newtona-Raphsona" << endl;  
    cout << "Podaj punkt startowy (a, b, 0 lub 1):" << endl;  
    cout << "Start = "; cin >> start;    //Pobranie punktu startowego  
    NewtonRalphson(a, b, Dokladnosc, start);  
  
    getchar(); getchar();  
    return 0;  
}
```

Rysunek 2. Funkcja main

Definicję algorytmu bisekcji rozpoczęto od obliczenia wartości na granicach przedziałów oraz sprawdzeniu warunku istnienia rozwiązania w przedziale. Rozpoczęto pętlę while kończącą się w przypadku gdy odległość między granicami będzie mniejsza od zadanej dokładności bądź gdy wartość naszego punktu środkowego dostatecznie zbliży się do zera (w granicach przesłanej dokładności). W pętli sprawdzamy który z podprzedziałów spełnia warunek istnienia pierwiastka, a następnie kontynuujemy obliczenia na tym przedziale.

```
double Bisekcja(double a, double b, double precision) {
    double Fa = f(a); //wartosc funkcji na lewej granicy
    double Fb = f(b); //wartosc funkcji na prawej granicy
    double l_granica = a;
    double p_granica = b;
    double Xo;
    double Fo;
    if (Fa * Fb > 0) { //sprawdzenie warunku
        cout << "Funkcja nie spelnia zalozen" << endl;
        return -1;
    }
    while (fabs(l_granica - p_granica) > precision)
    {
        Xo = (l_granica + p_granica) / 2; //wyznaczenie punktu srodkowego
        Fo = f(Xo); //wartosc punktu srodkowego

        if (fabs(Fo) < precision) { //sprawdzenie drugiego warunku
            cout << "Pierwiastek Funkcji w przedziale: " << setprecision(prec) << Xo << endl;
            return 0; //spełnienie warunku konczy algorytm
        }
        if (Fa * Fo < 0) //sprawdzenie warunku dla podprzedzialow
        {
            p_granica = Xo;
        }
        else
        {
            l_granica = Xo;
            Fa = Fo;
        }
    }
    cout << "Pierwiastek Funkcji w przedziale: " << setprecision(prec) << Xo << endl;
    return 1;
}
```

Rysunek 3. Funkcja realizująca algorytm Bisekcji

Implementację metody Newtona-Raphsona rozpoczęto dokładnie w ten sam sposób co poprzednią – obliczono wartości granic, sprawdzono warunek, rozpoczęto pętlę while zakończoną w przypadku dostatecznego zbliżenia się do zera bądź zmniejszanie odległości kroku między punktami do wartości poniżej zadanej dokładności. Na podstawie wartości funkcji oraz jej pochodnej w punkcie początkowym wyznaczano kolejne przybliżenia rozwiązania, aż do uzyskania ustalonej precyzji.

```
double NewtonRaphson(double a, double b, double precision, double start) {
    double Fa = f(a); //wartosc funkcji na lewej granicy
    double Fb = f(b); //wartosc funkcji na prawej granicy

    if (Fa * Fb > 0) { //sprawdzenie warunku
        cout << "Funkcja nie spelnia zalozen" << endl;
        return -1;
    }

    double X1 = start;
    double X0;
    double f1 = f(X1); //wartosc funkcji dla punktu startowego
    while (fabs(f1) >= precision) //warunek zakonczenia algorytmu
    {
        double df = d(X1); //obliczenie pochodnej
        X0 = X1;
        X1 = X1 - (f1 / df); //obliczenie kolejnego punktu startowego
        f1 = f(X1);
        if (fabs(X1 - X0) <= precision) break; //warunek zakonczenia algorytmu
    }
    cout << "Pierwiastek Funkcji w przedziale: " << setprecision(prec) << X1 << endl;
    return 1;
}
```

Rysunek 4. Funkcja realizująca metodę Newtona-Raphsona

Cały kod

```
using namespace std;
double Dokladnosc = 0.0000001; //Dokladnosc obliczen
int prec = 11; //Liczba wyswietlanych miejsc po przecinku

double f(double x) { //Pomocnicza fukncja obliczajaca wartosc w punkcie x
    return 3*pow(x,5) - 4*pow(x,4) + x*x - 7*x + 3;
}

double d(double x) { //Pomocnicza fukncja obliczajaca wartosc pochodnej w punkcie x
    return 15*pow(x,4) - 16*pow(x,3) + 2*x - 7;
}

double Bisekcja(double a, double b, double precision) {
    double Fa = f(a); //wartosc funkcji na lewej granicy
    double Fb = f(b); //wartosc funkcji na prawej granicy
    double l_granica = a;
    double p_granica = b;
    double Xo;
    double Fo;
    if (Fa * Fb > 0) { //sprawdzenie warunku
        cout << "Funkcja nie spelnia zalozen" << endl;
        return -1;
    }
    while (fabs(l_granica - p_granica) > precision)
    {
        Xo = (l_granica + p_granica) / 2; //wyznaczenie punktu srodkowego
        Fo = f(Xo); //wartosc punktu srodkowego

        if (fabs(Fo) < precision) { //sprawdzenie drugiego warunku
            cout << "Pierwiastek Funkcji w przedziale: " << setprecision(prec) << Xo << endl;
            return 0; //spelnienie warunku konczy algorytm
        }
        if (Fa * Fo < 0) //sprawdzenie warunku dla podprzedzialow
        {
            p_granica = Xo;
        }
        else
        {
            l_granica = Xo;
            Fa = Fo;
        }
    }
    cout << "Pierwiastek Funkcji w przedziale: " << setprecision(prec) << Xo << endl;
    return 1;
}
```

Rysunek 5. Całość kodu cz.1

```

double NewtonRaphson(double a, double b, double precision, double start) {
    double Fa = f(a);    //wartosc funkcji na lewej granicy
    double Fb = f(b);    //wartosc funkcji na prawej granicy

    if (Fa * Fb > 0) {    //sprawdzenie warunku
        cout << "Funkcja nie spelnia zalozen" << endl;
        return -1;
    }

    double X1 = start;
    double X0;
    double f1 = f(X1);    //wartosc funkcji dla punktu startowego
    while (fabs(f1) >= precision)    //warunek zakonczenia algorytmu
    {
        double df = d(X1);    //obliczenie pochodnej
        X0 = X1;
        X1 = X1 - (f1 / df);    //obliczenie kolejnego punktu startowego
        f1 = f(X1);
        if (fabs(X1 - X0) <= precision) break;    //warunek zakonczenia algorytmu
    }
    cout << "Pierwiastek Funkcji w przedziale: " << setprecision(prec) << X1 << endl;
    return 1;
}

int main() {

    double a, b;
    double start;
    cout << "Obliczanie pierwiastka funkcji" << endl;
    cout << endl;
    cout << "-----" << endl;
    cout << "Metoda Bisekcji" << endl;
    cout << "Podaj zakres poszukiwan pierwiastka: " << endl;
    cout << "a = "; cin >> a;    //Pobranie granic przedzialu
    cout << "b = "; cin >> b;
    Bisekcja(a, b, Dokladnosc);
    cout << endl;
    cout << "-----" << endl;
    cout << "Metoda Newtona-Raphsona" << endl;
    cout << "Podaj punkt startowy (a, b, 0 lub 1):" << endl;
    cout << "Start = "; cin >> start;    //Pobranie punktu startowego
    NewtonRaphson(a, b, Dokladnosc, start);

    getchar(); getchar();
    return 0;
}

```

Rysunek 6. Całość kodu cz.2

4. Testy

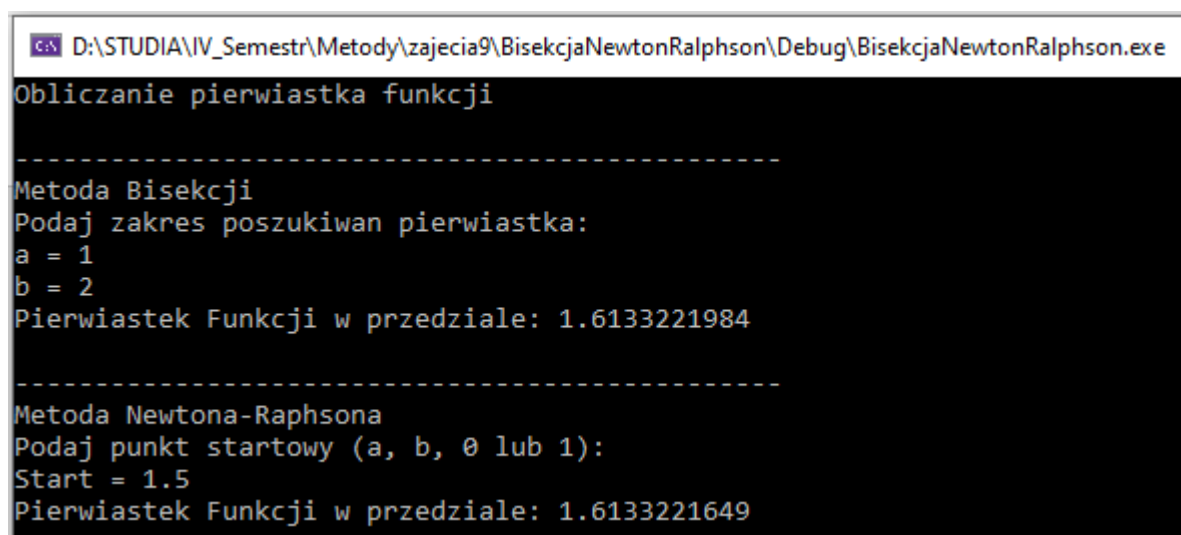
W celu zweryfikowania wyników programu dokonano trzech testów z wykorzystaniem kalkulatora znajdującego się na stronie <https://obliczone.pl/kalkulatory/1081-kalkulator-miejsc-zerowych-minimum-maksimum-funkcji>. W każdym przypadku obliczono błąd względny. Wszystkie obliczenia w programie wykonywano z dokładnością do 0.0000001 .

Test 1.

Tabela 1. Dane dla testu 1

Wzór funkcji	Lewa granica przedziału	Prawa granica przedziału	Punkt początkowy
$3x^5 - 4x^4 + x^2 - 7x + 3$	1	2	1.5

Wynik programu:



```
D:\STUDIA\IV_Semestr\Metody\zajecia9\BisekcjaNewtonRaphson\Debug\BisekcjaNewtonRaphson.exe
Obliczanie pierwiastka funkcji
-----
Metoda Bisekcji
Podaj zakres poszukiwan pierwiastka:
a = 1
b = 2
Pierwiastek Funkcji w przedziale: 1.6133221984
-----
Metoda Newtona-Raphsona
Podaj punkt startowy (a, b, 0 lub 1):
Start = 1.5
Pierwiastek Funkcji w przedziale: 1.6133221649
```

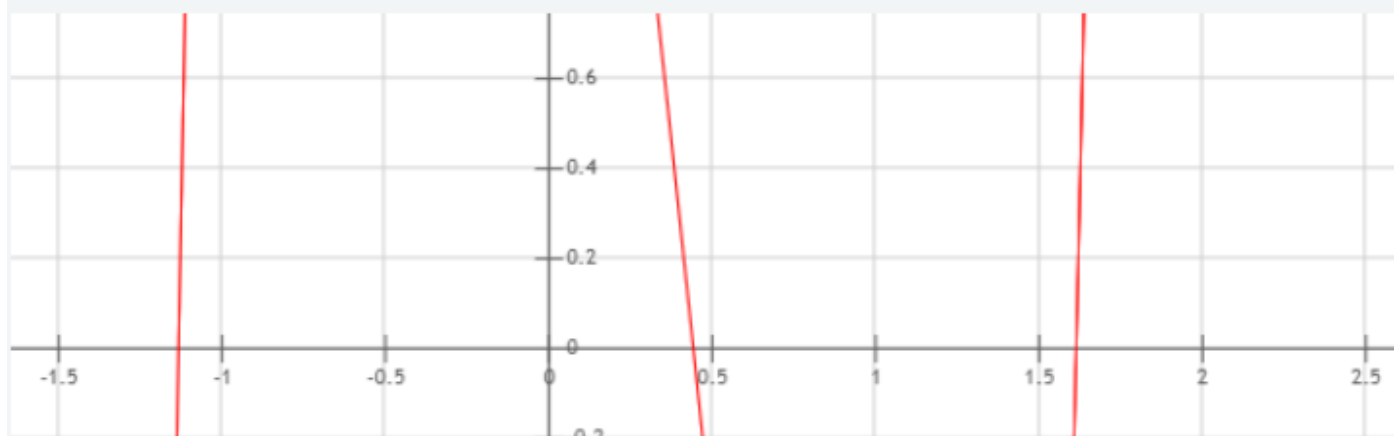
Rysunek 7. Wynik działania programu

Wynik ze strony:

Miejsca zerowe funkcji:

$$\begin{bmatrix} -1.13357247374 \\ 0.441899667066 \\ 1.6133221643 \end{bmatrix}$$

Poniżej wykres funkcji $f(x) = 3 \cdot x^5 - 4 \cdot x^4 + x^2 - 7 \cdot x + 3$



Rysunek 8. Wynik ze strony

Błędy względne

- Dla metody bisekcji:

$$\delta = \left| \frac{1.6133221643 - 1.6133221984}{1.6133221643} \right| * 100\% = 2.11365 * 10^{-6}\%$$

- Dla metody Newtona-Raphsona:

$$\delta = \left| \frac{1.6133221643 - 1.6133221649}{1.6133221643} \right| * 100\% = 3.71903 * 10^{-8}\%$$

Test 2.

Tabela 2. Dane dla testu 2

Wzór funkcji	Lewa granica przedziału	Prawa granica przedziału	Punkt początkowy
$3\cos(2 - 2x)$	1	2	2

Wynik programu:

```
D:\STUDIA\IV_Semestr\Metody\zajecia9\BisekcjaNewtonRalphson\Debug\BisekcjaNewtonRalphson.exe
Obliczanie pierwiastka funkcji
-----
Metoda Bisekcji
Podaj zakres poszukiwan pierwiastka:
a = 1
b = 2
Pierwiastek Funkcji w przedziale: 1.7853981853
-----
Metoda Newtona-Raphsona
Podaj punkt startowy:
Start = 2
Pierwiastek Funkcji w przedziale: 1.7853981634
```

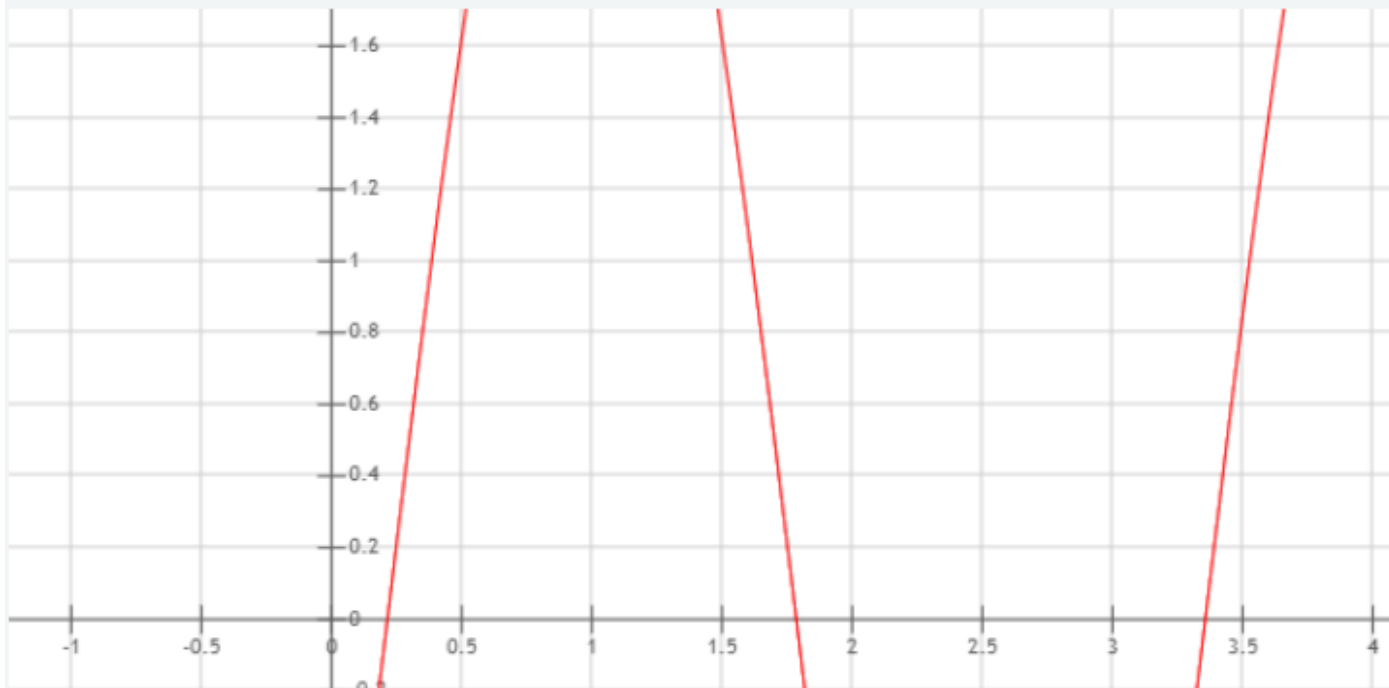
Rysunek 9. Wynik działania programu

Wynik ze strony:

Miejsca zerowe funkcji:

$$\begin{bmatrix} \frac{-1}{4} \cdot (\pi - 4) \\ \frac{-1}{4} \cdot (-\pi - 4) \end{bmatrix}$$

Poniżej wykres funkcji $f(x) = 3 \cdot \cos(2 - 2 \cdot x)$



Rysunek 10. Wynik ze strony

Błędy względne

- Dla metody bisekcji:

$$\delta = \left| \frac{1.785398163 - 1.7853981853}{1.785398163} \right| * 100\% = 1.22677 * 10^{-6}\%$$

- Dla metody Newtona-Raphsona:

$$\delta = \left| \frac{1.785398163 - 1.7853981634}{1.785398163} \right| * 100\% = 1.54029 * 10^{-10}\%$$

Test 3.

Tabela 3. Dane dla testu 3

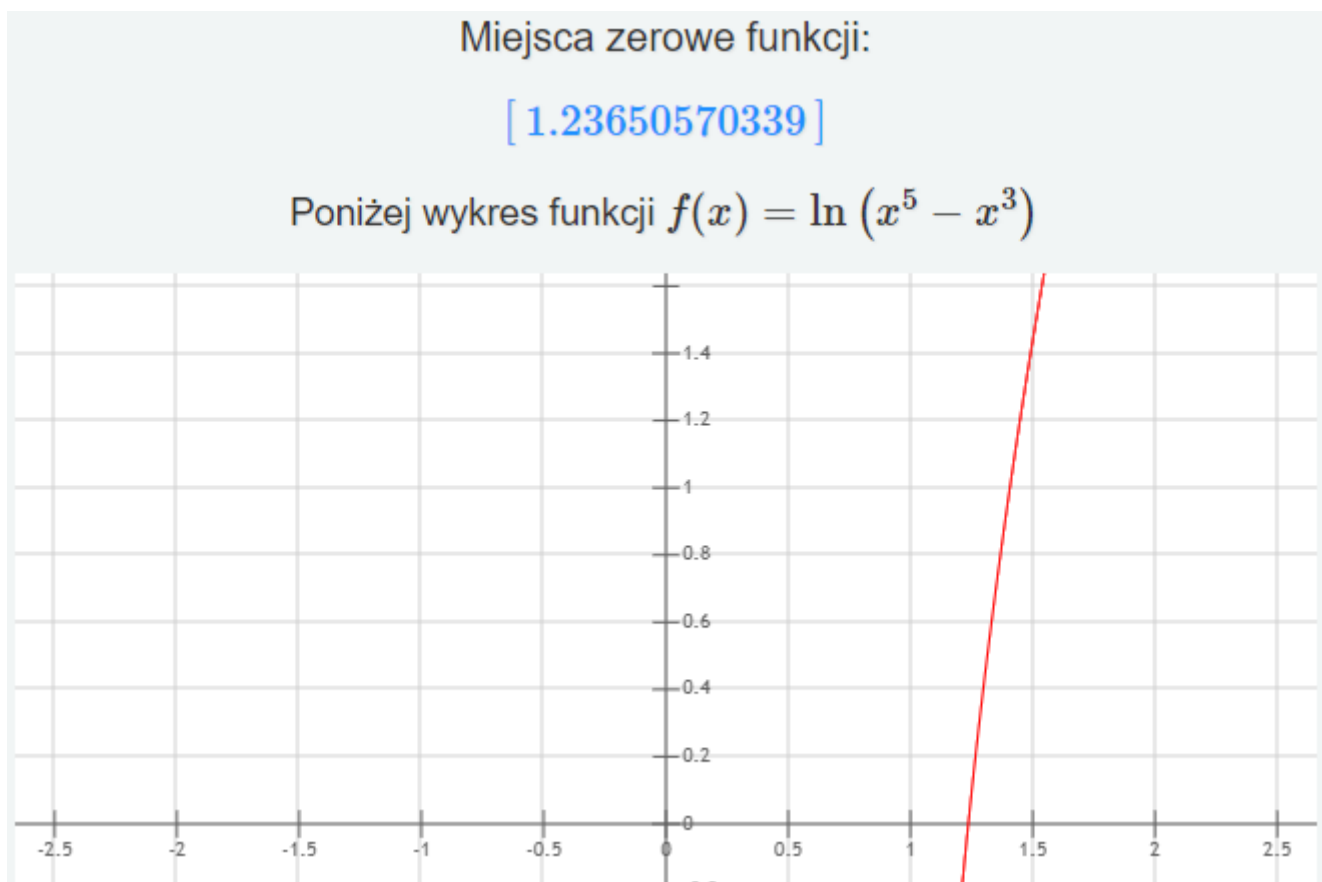
Wzór funkcji	Lewa granica przedziału	Prawa granica przedziału	Punkt początkowy
$\ln(x^5 - x^3)$	1	2	1.5

Wynik programu:

```
D:\STUDIA\IV_Semestr\Metody\zajecia9\BisekcjaNewtonRalphson\Debug\BisekcjaNewtonRalphson.exe
Obliczanie pierwiastka funkcji
-----
Metoda Bisekcji
Podaj zakres poszukiwan pierwiastka:
a = 1
b = 2
Pierwiastek Funkcji w przedziale: 1.2365056872
-----
Metoda Newtona-Raphsona
Podaj punkt startowy:
Start = 1.5
Pierwiastek Funkcji w przedziale: 1.2365056974
```

Rysunek 11. Wynik działania programu

Wynik ze strony:



Rysunek 12. Wynik ze strony

Błędy względne

- Dla metody bisekcji:

$$\delta = \left| \frac{1.23650570339 - 1.2365056872}{1.23650570339} \right| * 100\% = 1.30933 * 10^{-6}\%$$

- Dla metody Newtona-Raphsona:

$$\delta = \left| \frac{1.23650570339 - 1.2365056974}{1.23650570339} \right| * 100\% = 4.8443 * 10^{-7}\%$$

5. Wnioski

Poznane podczas ćwiczenia metody Newtona-Raphsona i bisekcji są skutecznymi sposobami rozwiązywania równań nieliniowych. Metody te zapewniają bardzo dokładne wyniki, a ich precyzję możemy swobodnie zmieniać modyfikując dokładność za jaką mają być wykonane obliczenia. Analizując wyniki testów można zauważyć, że metoda stycznych jest nieco bardziej dokładna, przy czym dzięki wykorzystaniu pochodnej do obliczania kolejnych przybliżeń jest ona skuteczniejsza niż metoda bisekcji, jej zaletą za to jest bardzo duża prostota i łatwość implementacji. Kolejną przewagą metody połowienia jest to, że nie wymaga ona punktu startowego, gdzie w metodzie Newtona nieprawidłowe obranie tego punktu może skutkować błędnym wynikiem. Mimo wymienionych wad metody prawidłowo spełniają swoje zadanie obliczania pierwiastków równań.