Zadanie 10

Opis:

Wykres wygenerowany za pomocą programu GNUplot. Program wykonany w języku C++

Kompilacja/uruchomienie programu:

Aby skompilować program, którego kod znajduje się na końcu tego dokumentu można skopiować go do wybranego IDE np.: Visual Studio, Code Block itd. Bądź uruchomić z poziomu konsoli wybranymi komendami, tak jak zwykły program w C++. Program po takim uruchomieniu wypisze w słupku rozwiązania układu równań.

Metoda rozwiązania/ dyskusja:

Zadanie 10 polegalo na rozwiązaniu równania det(A-λ1) trzema metodami poszukiwania miejsc zerowych.Z podanych metod wybrałem metody : Newtona,Bisekcji i Reguły Falsi.

Najpierw określiłem jak wygląda ten wyznacznik:

det(A-λ1) = $\begin{vmatrix} 4-\lambda & -1 & 0 \\ -1 & 4-\lambda & -1 \\ 0 & -1 & 4-\lambda \end{vmatrix}$ z czego otrzymujemy: det(A-λ1) = $-x^3+12x^2$-46x + 56.

Std::cout zastąpiłem mniej zasobożernymi operacjami printf.

Zamiast wektorów (drogie operacje push) użyłem zwykłych tablic.

No i brak pow(), tylko zwykle mnożenie.

Metoda Newtona:

W pierwszym kroku metody wybierany jest punkt startowy z którego następnie wyprowadzana jest styczna w **f(x_1)**. Odcięta punktu przecięcia stycznej z osią OX jest pierwszym przybliżeniem rozwiązania (ozn. **x_2**).

Jeśli to przybliżenie nie jest satysfakcjonujące, wówczas punkt **x_2** jest wybierany jako nowy punkt startowy i wszystkie czynności są powtarzane. Proces jest kontynuowany, aż zostanie uzyskane wystarczająco dobre przybliżenie pierwiastka

Kolejne przybliżenia są dane rekurencyjnym wzorem:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

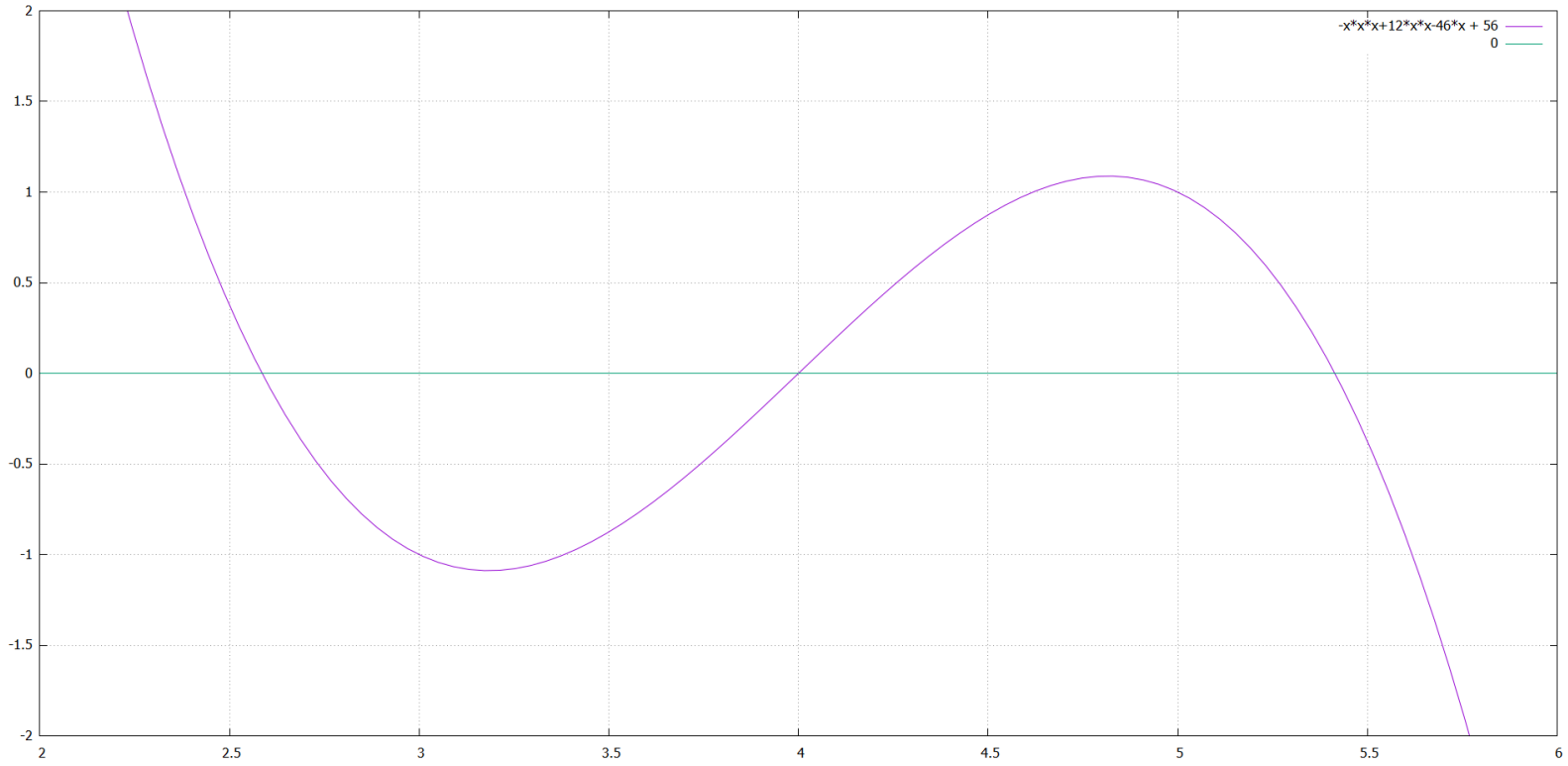
Metoda Bisekcji

Jest zgodna z twierdzeniem, że jeżeli funkcja na końcach przedziału domkniętego ma wartości o rożnych znakach to wewnątrz tego przedziału istnieje co najmniej jeden pierwiastek równania. Algorytm dzieli przedział na pół i sprawdza znaki w przedziałach. Koniec nastepuje wtedy kiedy algorytm znajdzie rozwiązanie o zadanej przez programistę dokladnosci.

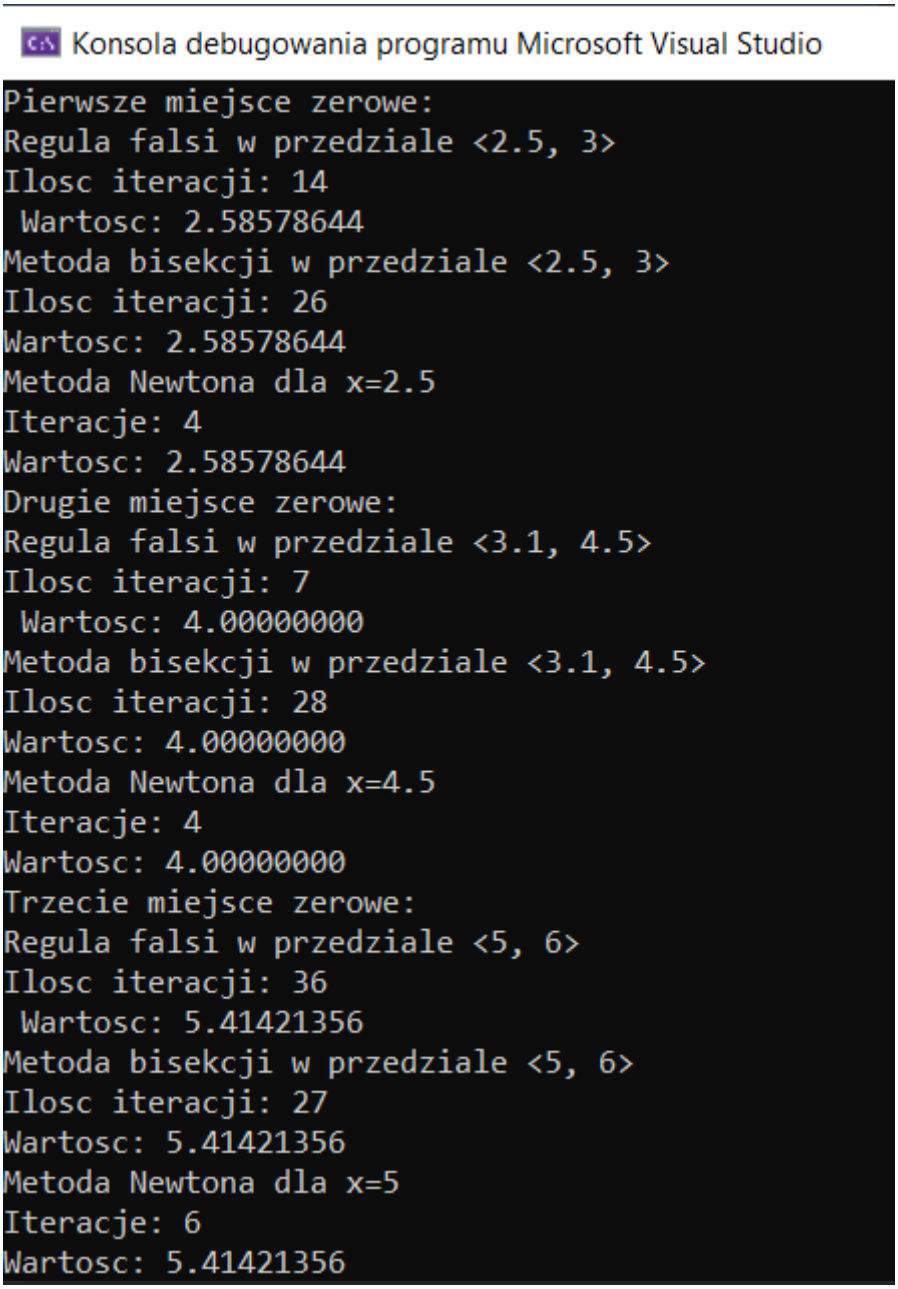
Reguła Falsi:

Metoda Regula Falsi jest rozwinięciem metody Bisekcji, wykorzystującym wiekszą ilość informacji o funkcji niż tylko jej znak. Metoda ta wykorzystuje interpolację liniową funkcji której zero jest poszukiwane. Prosta ta przechodzi przez punkty graniczne obszaru poszukiwań.

Graficzne przedstawienie funkcji:



Wyniki działania programów:



Opis:

Wybrałem powyższe przedziały biorąc pod uwagę wykres funkcji, aby znajdywały się w miarę bliskim położeniu od danego miejsca zerowego. Ilość iteracji mocno zależy od tego jaki przedział wybierzemy. W dwóch pierwszych przedzialach największą ilość iteracji miała metoda bisekcji, lecz gdy ustawiłem przedział od 5 do 6 gdzie funkcja bardzo mocno maleje wtedy najwięcej iteracji miała Regula Falsi. Otóż w Regule Falsi wykorzystywany jest przebieg funkcji. W metodzie bisekcji x_0 było wyznaczane zawsze w środku przedziału poszukiwań pierwiastka. Tutaj punkt ten jest wyliczany jako punkt przecięcia fałszywej prostej z osią OX. Gdy przedział maleje, prosta ta zaczyna coraz bardziej przypominać faktyczny przebieg funkcji.Dlatego algorytm wymaga mniej kroków , ale w tym przypadku więcej. Wszystkie trzy miejsca zerowe z najmniejszą liczbą iteracji policzyła metoda Newtona.

sKod programu:

```
#include <iostream>

using namespace std;
double static epsilon =  1e-8;

double funkcja(double x)
{
	return -x * x * x + 12 * x * x - 46 * x + 56;
}
double poch1(double x)
{
	return -3 * x * x + 24 * x - 46;
}

double metoda_falsi(double a, double b)
{
	double x = 0, fa = 0, fb = 0;
	double x1=0, xn, f;
	int counter = 0;
	fb = funkcja(b);
	fa = funkcja(a);

	//printf("wartosc x1: %d\n", x1);
	do {
		x1 = ((b * funkcja(a) - a * funkcja(b)) / (funkcja(a) - funkcja(b)));
		f = funkcja(x1);
		if (funkcja(b) * f > 0)
		{
			b = x1;
			fb = f;
			if (x == -1)
			{
				fa /= 2;
				x = -1;
			}
		}
		else if (fa * f > 0) {
			a = x1;
			fa = f;
			if (x == +1)
			{
				fb /= 2;
				x = +1;
			}
		}
		else {
			break;
		}counter++;



	} while ((abs(b - a)) > (1e-8 * abs(b + a)));

	printf("Ilosc iteracji: %d\n ", counter);
	return x1;
}
	double metoda_bisekcji(double a, double b)
	{
		double x1, x0 = 0;
		int counter = 0;
```

```
		while (abs(a - b) > epsilon)
		{
			x1 = ((a + b) / 2);
			if (funkcja(x1) == 0)
			{
				break;
			}
			else if ((funkcja(x1) * funkcja(a)) < 0)
			{
				b = x1;
			}
			else {
				a = x1;
			}
			counter++;
		}
		printf("Ilosc iteracji: %d\n", counter);
		return x1;

	}
double metoda_newtona(double xn)
{
	int counter = 0;

	double x1 = xn;
	while (abs(funkcja(x1)) > epsilon) {
		xn = x1 - funkcja(x1) / poch1(x1);
		x1 = xn;
		counter++;
	}
	printf("Iteracje: %d\n", counter);
	return x1;
}
int main()
{
	printf("Pierwsze miejsce zerowe: \n");
	printf("Regula falsi w przedziale <2.5, 3>\n");
	printf("Wartosc: %.8f\n", metoda_falsi(2.5, 3));
	printf("Metoda bisekcji w przedziale <2.5, 3>\n");
	printf("Wartosc: %.8f\n", metoda_bisekcji(2.5, 3));
	printf("Metoda Newtona dla x=2.5\n");
	printf("Wartosc: %.8f\n", metoda_newtona(2.5));

	printf("Drugie miejsce zerowe: \n");
	printf("Regula falsi w przedziale <3.1, 4.5>\n");
	printf("Wartosc: %.8f\n", metoda_falsi(3.1, 4.5));
	printf("Metoda bisekcji w przedziale <3.1, 4.5>\n");
	printf("Wartosc: %.8f\n", metoda_bisekcji(3.1, 4.5));
	printf("Metoda Newtona dla x=4.5\n");
	printf("Wartosc: %.8f\n", metoda_newtona(4.5));

	printf("Trzecie miejsce zerowe: \n");
	printf("Regula falsi w przedziale <5, 6>\n");ss
	printf("Wartosc: %.8f\n", metoda_falsi(5, 6));
	printf("Metoda bisekcji w przedziale <5, 6>\n");
	printf("Wartosc: %.8f\n", metoda_bisekcji(5, 6));
	printf("Metoda Newtona dla x=5\n");
	printf("Wartosc: %.8f\n", metoda_newtona(5));


	return 0;
}
```