При решаването на нелинейни уравнения се използват итерационни методи.
Най-често се прилагат методите на Нютон и метод на секущите.

За решаването на нелинейно уравнение **f(x)=0** чрез метода на Нютон (известен още като метод на допирателните), се използва итерационната формула:

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

Ще покажем един пример за конзолно приложение, готово за компилиране
В този пример ще решаваме уравнението $f(x) = x^2 - 2 = 0$ (което трябва да ни даде стойността на +/- 1.4142...).

Програмен код

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

using namespace std;

// Дефинираме функцията f(x)
double f(double x) {
	return x * x - 2;
}

// Дефинираме производната на функцията f'(x)
double f_prime(double x) {
	return 2 * x;
}

int main() {
	double x0, x1, eps;
	int max_iter, iter = 0;

	cout << "=== Metod na Newton za nelineini uravnenia ===" << endl;

	// Входни данни
	cout << "Vuvedete nachalno priblijenie (x0): ";
	cin >> x0;
	cout << "Vuvedete tochnost (eps, napr. 0.0001): ";
	cin >> eps;
	cout << "Vuvedete maksimalen broi iteracii: ";
	cin >> max_iter;

	cout << fixed << setprecision(6);
	cout << "\nIteracia\t x_n\t\t f(x_n)" << endl;
	cout << "-----------------------------------------------" << endl;

	do {
		double fx = f(x0);
		double dfx = f_prime(x0);

		// Проверка за делене на нула (ако производната е 0, методът се проваля)
```

```
            if (abs(dfx) < 1e-10) {
                cout << "Greshka: Proizvodnata e blizko do nula!" << endl;
                break;
            }

            x1 = x0 - fx / dfx; // Формула на Нютон

            iter++;
            cout << iter << "\t\t " << x1 << "\t " << f(x1) << endl;

            // Проверка дали сме достигнали желаната точност
            if (abs(x1 - x0) < eps) {
                cout << "-------------------------------------------" << endl;
                cout << "Nameren koren: " << x1 << " sled " << iter << " iteracii." << endl;
                break;
            }

            x0 = x1;

            if (iter >= max_iter) {
                cout << "Dostignat e maksimalniat broi iteracii bez shodimost." << endl;
            }

        } while (iter < max_iter);

        system("pause"); // Задържа конзолата отворена в Dev-C++
        return 0;
    }
```

Кратко упътване за работа с кода:

1. **Дефиниране на функцията**: В секцията f(double x) записвате уравнението, което искате да решите,
2. **Производна**: В f_prime(double x) трябва ръчно да въведете производната на вашата функция. Методът на Нютон изисква тя да е известна и непрекъсната.
3. **Начално приближение x_0**: Това е вашата „първоначална прогноза“. Колкото по-близо е до реалния корен, толкова по-бързо ще се извърши пресмятането.
4. **Условие за край**: Програмата спира, когато разликата между две последователни стъпки стане по-малка от зададената точност **eps**.

Неудобство за този метод, че се изисква да се въведе производната, която не винаги лесно се получава, особено а дълги изрази.

Когато искаме да избегнем ръчното пресмятане на производната, обикновено преминаваме към **Метода на Секущите (Secant Method)**.

При него производната $f'(x_n)$ се апроксимира чрез стойностите на функцията в две съседни точки:

$$f'(x_n) \approx f(x_n) - f(x_{n-1}) \; / \; (x_n - x_{n-1})$$

Така итерационната формула става:

$$x_{n+1}=x_n-f(x_n)\cdot(x_n-x_{n-1})\,/\,(f(x_n)-f(x_{n-1}))$$

Ето как изглежда кодът за **Dev-C++**, който не изисква дефиниране на производна:

Програмен код (C++ - Метод на Секущите)

C++

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

using namespace std;

// Дефинираме само основната функция f(x)
double f(double x) {
    // Пример: x^3 - x - 2 = 0
    return pow(x, 3) - x - 2;
}

int main() {
    double x0, x1, x2, eps;
    int max_iter, iter = 0;

    cout << "=== Metod na Sekushtite (bez proizvodna) ===" << endl;

    // Входни данни - изисква две начални приближения
    cout << "Vuvedete purvo nachalno priblijenie (x0): ";
    cin >> x0;
    cout << "Vuvedete vtoro nachalno priblijenie (x1): ";
    cin >> x1;
    cout << "Vuvedete tochnost (eps): ";
    cin >> eps;
    cout << "Vuvedete maksimalen broi iteracii: ";
    cin >> max_iter;

    cout << fixed << setprecision(6);
    cout << "\nIteracia\t x_n\t\t f(x_n)" << endl;
    cout << "-----------------------------------------------" << endl;

    while (iter < max_iter) {
        double f0 = f(x0);
        double f1 = f(x1);

        // Проверка за предотвратяване на делене на нула
        if (abs(f1 - f0) < 1e-12) {
            cout << "Greshka: Razlikata vuv funkciite e tvurde malka!" << endl;
            break;
        }

        // Формула на Секущите
        x2 = x1 - f1 * (x1 - x0) / (f1 - f0);

        iter++;
        cout << iter << "\t\t " << x2 << "\t " << f(x2) << endl;

        // Проверка за край (достигната точност)
        if (abs(x2 - x1) < eps) {
            cout << "-----------------------------------------------" << endl;
```

```
            cout << "Nameren koren: " << x2 << " sled " << iter << " iteracii."
<< endl;
            break;
        }

        // Подготовка за следващата итерация
        x0 = x1;
        x1 = x2;

        if (iter == max_iter) {
            cout << "Dostignat e maksimalniat broi iteracii." << endl;
        }
    }

    system("pause");
    return 0;
}
```

Основни разлики :

- **Без производна:** Вече няма нужда от функция `f_prime`. Програмата сама „оценява“ наклона на кривата чрез две точки.
- **Две начални точки:** За разлика от Нютон, тук трябва да въведете две стойности (x0 и x1), които са близо до очаквания корен.
- **Логика:** Този метод е изключително удобен за сложни функции, чието диференциране е трудно или невъзможно.
- **Сходимост:** Малко по-бавен е от оригиналния метод на Нютон, но много по-бърз от метода на делене на интервала (бисекция).

Този подход е стандартният избор, когато искаме „автоматизиран“ процес без математически анализ на производните.