

<b>Imię i nazwisko</b> Meg Paskowski	<b>Temat laboratorium</b> Interpolacja Newtona	<b>Data oddania</b> 21.03.2024r.	<b>Data ćwiczeń</b> 15.03.2014r.
<b>Prowadzący</b> dr hab. inż. Marcin Hojny		<b>Grupa laboratoryjna</b> 4	

### 1. Cel ćwiczenia

Celem laboratorium nr. 3 było zapoznanie z Interpolacją Newtona oraz implementacja zagadnienia w wybranym przez siebie języku programowania.

### 2. Wstęp teoretyczny

Interpolacja Newtona jest metodą numeryczną stosowaną do przybliżania funkcji poprzez wielomian stopnia  $n$ , wykorzystując tzw. wielomian Newtona. W tej metodzie wielomian ten jest konstruowany tak, aby przechodził przez zestaw punktów danych należących do analizowanej funkcji. Interpolacja Newtona stanowi jedną z popularnych metod interpolacji wielomianowej, która opiera się na ilorazach różnicowych, aby utworzyć wielomian stopnia  $n$ . W porównaniu do omawianej na poprzednich laboratoriach interpolacji Lagrange'a ta metoda jest bardziej elastyczna tzn. umożliwia efektywniejsze oraz łatwiejsze obliczeniowo dodawanie nowych punktów danych do już istniejącego wielomianu interpolacyjnego. Do utworzenia algorytmu tej interpolacji należy zastosować wzory:

Wzór interpolacyjny Newtona dla  $n$ -tego rzędu:

$$W_n(x) = y_0 + [x_0, x_1] \cdot (x - x_0) + [x_0, x_1, x_2] \cdot (x - x_0) \cdot (x - x_1) + \dots + [x_0, x_1, \dots, x_n] \cdot (x - x_0) \cdot (x - x_1) \cdot \dots \cdot (x - x_{n-1}) \quad (1)$$

Wzór na ilorazy różnicowe dla  $k$ -tego rzędu:

$$[x_i, x_{i+1}, \dots, x_{i+k}] = \frac{[x_{i+1}, x_{i+2}, \dots, x_{i+k}] - [x_i, x_{i+1}, \dots, x_{i+k-1}]}{x_{i+k} - x_i} \quad (2)$$

Rozpatrując zbiór  $k+1$  punktów, jesteśmy w stanie obliczyć  $k$  ilorazów różnicowych, przy czym liczba wyników zmniejsza się o jeden w każdym kolejnym rzędzie. Poprzez wyznaczenie pierwszych wartości z każdego rzędu, możemy określić wartość wielomianu interpolacyjnego Newtona.

Przykładowa tabela pięciu punktów przedstawiająca wyznaczenie ilorazów różnicowych:

$x_i$	$y_i$	Rząd 1	Rząd 2	Rząd 3	Rząd 4	Rząd 5
$x_0$	$y_0$					
		$[x_0, x_1]$				
$x_1$	$y_1$		$[x_0, x_1, x_2]$			
		$[x_1, x_2]$		$[x_0, x_1, x_2, x_3]$		
$x_2$	$y_2$		$[x_1, x_2, x_3]$		$[x_0, x_1, x_2, x_3, x_4]$	
		$[x_2, x_3]$		$[x_1, x_2, x_3, x_4]$		$[x_0, x_1, x_2, x_3, x_4, x_5]$
$x_3$	$y_3$		$[x_2, x_3, x_4]$		$[x_1, x_2, x_3, x_4, x_5]$	
		$[x_3, x_4]$		$[x_2, x_3, x_4, x_5]$		
$x_4$	$y_4$		$[x_3, x_4, x_5]$			
		$[x_4, x_5]$				
$x_5$	$y_5$					

Tabela 1. Przykład wyznaczenia ilorazów różnicowych

### 3. Implementacja

W ramach ćwiczeń zaimplementowano metodę interpolacji Newtona w języku C++. Kluczowymi elementami implementacji są funkcje „*findNewton*” oraz „*differentialQuotient*” do których szczegółowo odniosę się poniżej.

Napisany program wczytuje dane z pliku „NM1.txt” do zmiennej (*read*) oraz deklaruje ilość punktów dla których będzie wykonywać zadanie (*size*).

```
int size;  
fstream read("NM1.txt");
```

Następnie sprawdzamy, czy plik został poprawnie otwarty

```
if (read.is_open()) {...}  
  
double differentialQuotient(int size, double **points, int i) {  
    if (size == 1)  
        return points[i][1];    // Zwracamy Y  
    else {  
        double quotient = (differentialQuotient(size - 1, points, i + 1) -  
differentialQuotient(size - 1, points, i)) / (points[i + size - 1][0] - points[i][0]);  
        return quotient;  
    }  
}
```

Gdy plik nie zostanie poprawnie otwarty informuje, o wystąpieniu konfliktu poprzez wyświetlony w terminalu komunikat.

```
if (read.is_open()) {...}  
else  
    cout << "Nie udało sie otworzyc pliku" << endl;
```

Natomiast po pozytywnym otwarciu pliku wczytywana jest liczba punktów funkcji, dla których będziemy wykonywać interpolacje do wcześniej zadeklarowanej tablicy dwuwymiarowej (*points2*). Po przypisaniu punktów do tablicy wypisuję dane z niej przy pomocy pętli *for*.

```
double** points2 = new double* [size];    // Tablica wskaźników dla punktów  
  
// Alokacja pamięci dla każdej tablicy wewnętrznej  
for (int i = 0; i < size; i++) {  
    points2[i] = new double[2];  
}  
  
// Odczytanie danych do tablicy dwuwymiarowej  
for (int i = 0; i < size; i++)  
    read >> points2[i][0] >> points2[i][1];  
  
cout << "Print points" << endl;  
for (int i = 0; i < size; i++)  
    cout << "X: " << points2[i][0] << " Y: " << points2[i][1] << endl;
```

W kolejnym kroku tworzona jest zmienna „*value*”, która będzie przechowywać wartość *X* dla którego szukamy wartości *Y* funkcji.

```
double value;  
cout << "Enter the X value for which you are looking for the Y value." << endl;  
cin >> value;
```

Następnie przejdę do omówienia głównych funkcji przy interpolacji Newtona.

Jako pierwszą mamy funkcję „*findNewton*”, która odpowiada za obliczenie wartości wielomianu w punkcie na podstawie podanych punktów. Przyjmuje ona ilość punktów dla podanej funkcji (*size*), wartość *X* dla jakiej szukamy *Y* (*value*) oraz tablicę dwuwymiarową zawierającą współrzędne punktów (*points2*). Na samym początku funkcja inicjalizuje zmienną „*result\_y*” na 0. Ta zmienna będzie przechowywać ostateczną wartość obliczonego wielomianu. Następnie funkcja przechodzi przez każdy punkt interpolacji oraz wylicza część wielomianu Newtona. Dla każdego punktu interpolacji, funkcja oblicza iloczyn różnicowy, który jest iloczynem różnicowym wartości  $(x - x_i)$  opisanym wzorem (1) dla wszystkich punktów. Funkcja sumuje wszystkie iloczyny różnicowe, mnożąc je przez odpowiednie ilorazy różnicowe, obliczone za pomocą funkcji „*differentialQuotient*”. W ten sposób uzyskuje wartość wielomianu Newtona dla punktu „*value*”, którą zwraca.

```
double findNewton(int size, double value, double **points) {
    double result_y = 0;    // Zmienna przechowująca wynik

    for (int i = 0; i < size; i++) { // Przechodzimy przez każdy punkt i liczymy wartości
        double quotient = 1; // Początkowy iloraz równy 1

        for (int j = 0; j < i; j++) { // Obliczanie ilorazów różnicowych
            quotient *= (value - points[j][0]);
        }
        result_y += quotient * differentialQuotient(i + 1, points, 0);
    }

    return result_y;
}
```

Drugą kluczową funkcją jest „*differentialQuotient*”, która odpowiedzialna jest za obliczenie kroków ilorazów. Wykorzystywane są one w wielomianie Newtona do interpolacji wartości funkcji różnicowych opisanych wzorem (2). Ta funkcja jest wykorzystywana przez powyższą „*findNewton*”.

Na początku funkcja sprawdza warunek bazowy, czy rozmiar „*size*” jest równy 1. Jeśli tak, oznacza to, że mamy do czynienia z jednym punktem, więc funkcja zwraca odpowiadającą mu wartość *Y*. Natomiast, gdy rozmiar „*size*” nie jest równy 1 funkcja rekurencyjnie wywołuje siebie dla mniejszego rozmiaru „*size - 1*” i oblicza iloraz różnicowy na podstawie punktów o indeksach „*i*” i „*i+1*”.

```
double differentialQuotient(int size, double **points, int i) {
    if (size == 1)
        return points[i][1];    // Zwracamy Y
    else {
        double quotient = (differentialQuotient(size - 1, points, i + 1) -
differentialQuotient(size - 1, points, i)) / (points[i + size - 1][0] - points[i][0]);
        return quotient;
    }
}
```

Na sam koniec należy pamiętać także o zwolnieniu pamięci oraz zamknięciu odczytu pliku.

```
// Zwolnienie zaalokowanej pamięci
for (int i = 0; i < size; i++) {
    delete[] points2[i];
}
delete[] points2;
```

```
read.close();
```

Poniżej przedstawione są zdjęcia całej implementacji algorytmu wraz z odczytem danych z pliku oraz zwolnieniem pamięci.

```
#include <fstream>

using namespace std;

double differentialQuotient(int size, double **points, int i) {
    if (size == 1)
        return points[i][1];    // Zwracamy Y
    else {
        double quotient = (differentialQuotient(size - 1, points, i + 1) - differentialQuotient(size - 1, points, i)) / (points[i + size - 1][0] - points[i][0]);
        return quotient;
    }
}

double findNewton(int size, double value, double **points) {
    double result_y = 0;    // Zmienna przechowująca wynik

    for (int i = 0; i < size; i++) { // Przechodzimy przez każdy punkt i liczymy wartości
        double quotient = 1; // początkowy iloraz równy 1

        for (int j = 0; j < i; j++) { // obliczanie ilorazów różnicowych
            quotient *= (value - points[j][0]);
        }
        result_y += quotient * differentialQuotient(i + 1, points, 0);    // Wynikiem jest suma wartości
    }

    return result_y;
}
```

Rysunek 1 fragment kodu implementacji Newtona - dwie główne funkcje

```
int main() {
    // Laboratoria 2 - Przybliżanie funkcji przez wielomian Newtona

    int size;
    ifstream read("NM1.txt");

    if (read.is_open()) {
        read >> size; // Liczba punktów
        cout << "The number of points for which we complete the task: " << size << endl;

        double** points2 = new double* [size]; // Tablica wskaźników

        // Allokacja pamięci dla każdej tablicy wewnętrznej
        for (int i = 0; i < size; i++) {
            points2[i] = new double[2];
        }

        // Odczytanie danych do tablicy dwuwymiarowej
        for (int i = 0; i < size; i++)
            read >> points2[i][0] >> points2[i][1];

        cout << "Print points" << endl;
        for (int i = 0; i < size; i++)
            cout << "X: " << points2[i][0] << " Y: " << points2[i][1] << endl;

        double value;
        cout << "Enter the X value for which you are looking for the Y value." << endl;
        cin >> value;

        cout << "For the given x= " << value << " the y value is: " << findNewton(size, value, points2) << endl;

        // Zwolnienie zaalokowanej pamięci
        for (int i = 0; i < size; i++) {
            delete[] points2[i];
        }
        delete[] points2;
    }
    else
        cout << "Nie udało się otworzyć pliku" << endl;

    read.close();

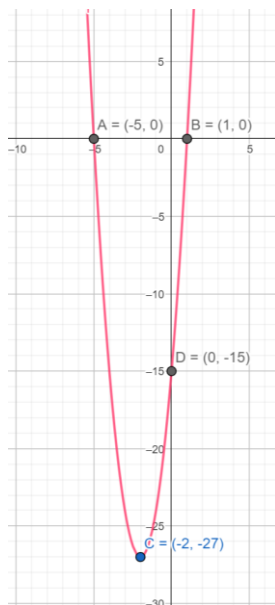
    return 0;
}
```

Rysunek 2 fragment implementacji - odczyt z pliku, zwolnienie pamięci

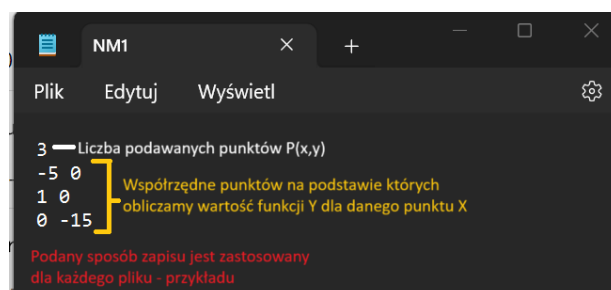
#### 4. Testy jednostkowe

Testy zostały wykonane dla danych z pliku „NM1.txt”. Zawartość pliku:

a. Dla funkcji kwadratowej:  $f(x) = 3x^2 + 12x - 15$



Rysunek 3 Wykres funkcji  $f(x) = 3x^2 + 12x - 15$



Rysunek 4 zawartość pliku „NM1.txt” dla przykładu a.

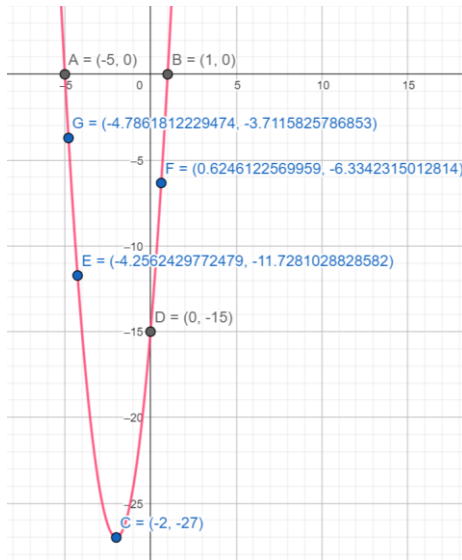
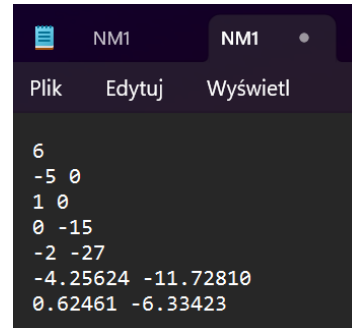
Szukany punkt C dla  $x = -2$

- Oczekiwany wynik: -27
- Otrzymany wynik: -27

```
The number of points for which we complete the task: 3
Print points
X: -5 Y: 0
X: 1 Y: 0
X: 0 Y: -15
Enter the X value for which you are looking for the Y value.
-2
For the given x= -2 the y value is: -27
```

Rysunek 5 Wyniki programu dla przykładu a

b. Ponownie dla funkcji:  $f(x)=3x^2+12x-15$

Rysunek 5 Wykres funkcji  $f(x)=3x^2+12x-15$  po dodaniu pkt.

Rysunek 6 zawartość pliku „NM1.txt” dla przykładu b.

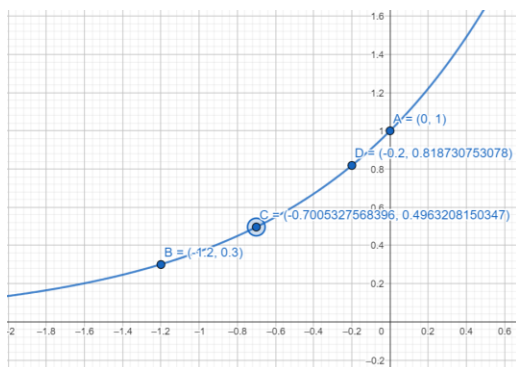
Szukany punkt G dla ok.  $x = -4.78618$

- Oczekiwany wynik: -3.7116
- Otrzymany wynik: -3.7116

```
The number of points for which we complete the task: 6
Print points
X: -5 Y: 0
X: 1 Y: 0
X: 0 Y: -15
X: -2 Y: -27
X: -4.25624 Y: -11.7281
X: 0.62461 Y: -6.33423
Enter the X value for which you are looking for the Y value.
-4.78618
For the given x= -4.78618 the y value is: -3.7116
```

Rysunek 7 Wyniki programu dla przykładu b

c. Dla funkcji eksponencjalnej:  $f(x)=e^x$



Rysunek 8 Wykres funkcji  $f(x) = e^x$

```

NM1
Plik  Edytuj  Wyświetl

3
-1.2 0.3
0 1
-0.2 0.818730

```

Rysunek 9 zawartość pliku „NM1.txt” dla przykładu c.

Szukany punkt C dla ok.  $x = -0.70053$

- Oczekiwany wynik: ok. 0.4963
- Otrzymany wynik: ok. 0.47834 → występuje niewielka różnica wyników

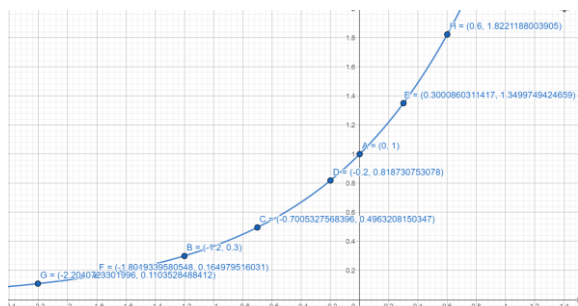
```

The number of points for which we complete the task: 3
Print points
X: -1.2 Y: 0.3
X: 0 Y: 1
X: -0.2 Y: 0.81873
Enter the X value for which you are looking for the Y value.
-0.70053
For the given x= -0.70053 the y value is: 0.478336

```

Rysunek 10 Wyniki programu dla przykładu c.

d. Ponownie dla funkcji eksponencjalnej:  $f(x) = e^x$



Rysunek 11 Wykres funkcji  $f(x) = e^x$  po dodaniu pkt.

```

NM1
Plik  Edytuj  Wyświetl

7
-1.2 0.3
0 1
-0.2 0.818730
0.6 1.822118
0.30008 1.34997
-1.80193 0.164979
-2.20407 0.110352

```

Rysunek 12 zawartość pliku „NM1.txt” dla przykładu d.

Szukany punkt C dla ok.  $x = -0.70053$

- Oczekiwany wynik: ok. 0.4963
- Otrzymany wynik: ok. 0.495939 → uzyskany wynik przy większej licznie podanych punktów jest jeszcze bardziej zbliżony do oczekiwanego.

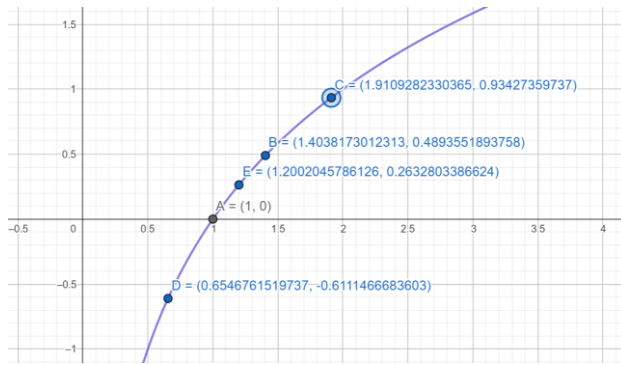
```

The number of points for which we complete the task: 7
Print points
X: -1.2 Y: 0.3
X: 0 Y: 1
X: -0.2 Y: 0.81873
X: 0.6 Y: 1.82212
X: 0.3 Y: 1.34997
X: -1.80193 Y: 0.164979
X: -2.20407 Y: 0.110352
Enter the X value for which you are looking for the Y value.
-0.70053
For the given x= -0.70053 the y value is: 0.495936

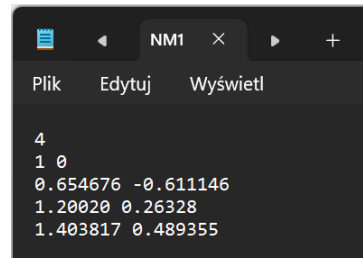
```

Rysunek 13 Wyniki programu dla przykładu d.

e. Dla funkcji logarytmicznej:  $f(x)=\log_2 x$



Rysunek 14 Wykres funkcji  $f(x)=\log_2 x$



Rysunek 15 zawartość pliku „NM1.txt” dla przykładu e.

Szukany punkt C dla ok.  $x = 1.910928$

- Oczekiwany wynik: ok. 0.93427
- Otrzymany wynik: ok. 1.01265

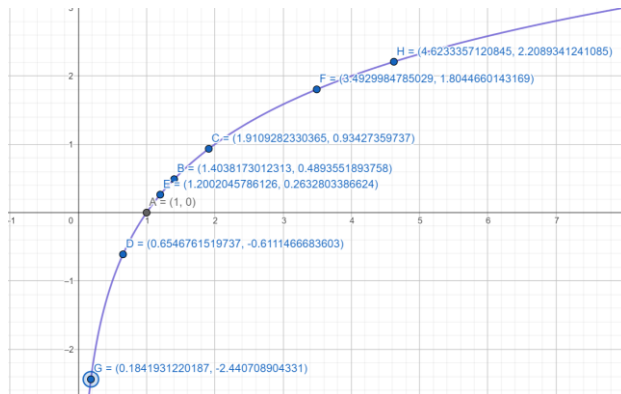
```

The number of points for which we complete the task: 4
Print points
X: 1 Y: 0
X: 0.654676 Y: -0.611146
X: 1.2002 Y: 0.26328
X: 1.40382 Y: 0.489355
Enter the X value for which you are looking for the Y value.
1.910928
For the given x= 1.91093 the y value is: 1.01265

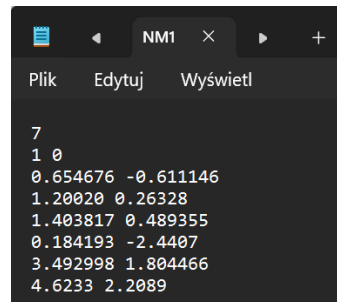
```

Rysunek 16 Wyniki programu dla przykładu e.

f. Ponownie dla funkcji logarytmicznej:  $f(x)=\log_2 x$



Rysunek 17 Wykres funkcji  $f(x)=\log_2 x$  po dodaniu pkt.



Rysunek 18 zawartość pliku „NM1.txt” dla przykładu f.

Szukany punkt C dla ok.  $x = 1.910928$

- Oczekiwany wynik: ok. 0.93427
- Otrzymany wynik: ok. 0.845964

```
The number of points for which we complete the task: 7
Print points
X: 1 Y: 0
X: 0.654676 Y: -0.611146
X: 1.2002 Y: 0.26328
X: 1.40382 Y: 0.489355
X: 0.184193 Y: -2.4407
X: 3.493 Y: 1.80447
X: 4.6233 Y: 2.2089
Enter the X value for which you are looking for the Y value.
1.910928
For the given x= 1.91093 the y value is: 0.845964
```

*Rysunek 3 Wyniki programu dla przykładu f.*

Jak można zauważyć w przeprowadzonych testach, wpływ na dokładność wyniku oczekiwanego względem otrzymanego ma ilość punktów oraz ich gęstość ułożenia na danym odcinku. Uzyskane wyniki są podobne do obliczania za pomocą interpolacji Lagrange'a, którą przedstawialiśmy na poprzednich laboratoriach.

## 5. Wnioski

Jak wykazały przeprowadzone powyżej testy, dokładność otrzymanego wyniku w interpolacji Newtona jest uzależniona od ilości punktów oraz ich gęstości na danym odcinku. Obserwujemy, że większa liczba punktów oraz ich równomierniejsze rozmieszczenie prowadzą do bardziej precyzyjnych wyników. Otrzymane rezultaty są w ogólności zgodne z oczekiwaniami. Przeprowadzona analiza podkreśla także znaczenie wyboru odpowiedniej metody interpolacji, uwzględniając charakter danych i funkcji. Metoda Newtona oferuje większą dokładność w porównaniu z metodą Lagrange'a.

## 6. Źródła

- Prezentacja autorstwa dr hab. inż. Marcina Hojnego „Metody numeryczne Interpolacja Newtona”.