

Paulina Grabowska	Metody numeryczne
Interpolacja Newtona	

1. Wstęp

Interpolacja jest techniką matematyczną umożliwiającą oszacowanie wartości funkcji w punktach, dla których nie można dokonać bezpośrednich pomiarów. Podczas interpolacji tworzona jest funkcja lub krzywa przechodząca przez podane punkty danych, co pozwala ocenić wartości argumentów pomiędzy tymi punktami.

Istnieje wiele różnych metod interpolacji: wielomiany, funkcje wymierne, funkcje trygonometryczne, a nawet funkcje sklejane. Interpolację wykorzystuje się w: metodach numerycznych i funkcjach konstrukcyjnych bazujących na danych pomiarowych ograniczonej liczby punktów.

2. Interpolacja Newtona

Interpolacja Newtona to technika numeryczna wykorzystywana do znalezienia wielomianu, który przechodzi przez dane punkty danych. Jest to metoda użyteczna w analizie danych, w grafice komputerowej, w numerycznym rozwiązywaniu równań różniczkowych i wielu innych dziedzinach.

Metoda ta zakłada, że znana jest lista n punktów danych (x_i, y_i) , gdzie x_i są różnymi argumentami funkcji, a y_i to odpowiadające im wartości. Interpolacja Newtona używa ilorazów różnicowych, które są iteracyjnie obliczane na podstawie punktów danych. Ilorazy te oblicza się w analogiczny sposób, ale jest to uzależnione od liczby rzędów; wzór dla ilorazów różnicowych k -tego rzędu (dla $k, i \in \mathbb{C}$) (2) oblicza się rekurencyjnie, wykorzystując wcześniej obliczone ilorazy różnicowe niższego stopnia (1).

$$[x_{n-1}, x_n] = \frac{y_n - y_{n-1}}{x_n - x_{n-1}} \quad (1)$$

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

Po obliczeniu ilorazów różnicowych należy skonstruować wielomian interpolacyjny Newtona (3), gdzie y_0 to wartość funkcji w pierwszym danym punkcie, a $[y_0, y_1]$ to iloraz różnicowy pierwszego stopnia między dwoma pierwszymi punktami danych; $[y_0, y_1, y_2]$ jest ilorazem różnicowym drugiego stopnia między trzema pierwszymi punktami danych, i tak dalej.

$$W_n(x) = y_0 + [x_0, x_1](x - x_0) + [x_0, x_1, x_2](x - x_0)(x - x_1) + \dots \\ + [x_0, x_1, \dots, x_n](x - x_0)(x - x_1) \cdot \dots \cdot (x - x_{n-1}) \quad (3)$$

Po utworzeniu wielomianu interpolacyjnego (3), używany jest on do obliczenia wartości funkcji $W_n(x)$ w dowolnym podanym punkcie. Wówczas podaną wartość x podstawia się do wielomianu celem obliczenia jej wartości.

Złożoność obliczeniowa metody interpolacji Newtona wzrasta wraz ze wzrostem liczby punktów i jest równa złożoności obliczeniowej metody interpolacji Lagrange'a wynosząc tym samym $O(n^2)$.

Interpolacja Newtona, podobnie zresztą jak interpolacja Lagrange'a jest przykładem interpolacji wielomianowej, co niesie za sobą ryzyko wystąpienia efektu Rungego. Jest to zjawisko pogorszenia jakości interpolacji, podczas zwiększania liczby węzłów. – prowadzi to do pogorszenia dokładności interpolacji na krańcach przedziału, nawet gdy punkty danych są równomiernie rozłożone.

3. Implementacja metody numerycznej

Rozpoczęto od implementacji funkcji *difference_quotient*, która oblicza ilorazy różnicowe do interpolacji Newtona. Funkcja ta przyjmuje dwie tablice typu *double* – $x[]$ oraz $y[]$ – które przechowują współrzędne wyjściowych punktów; oraz dwie zmienne: n będącą liczbą tych punktów oraz stopień wielomianu k .

Najpierw sprawdzany jest warunek, czy stopień interpolacji nie jest równy zero – czyli czy interpolujemy po jedynie jednym punkcie. W takim przypadku zwracana jest wartość funkcji w pierwszym punkcie, czyli w $y[0]$.

Następnie następuje inicjalizacja zmiennej *result*, która ustawiona jest na wartość 0.0. To właśnie ta zmienna przechowuje iloraz różnicowy. Zainicjowana później pętla *for* iteruje po i , po punktach od 0 do k , ponieważ nie można przekroczyć stopnia interpolacji. W tej pętli

inicjowana jest kolejna zmienna, *numerator*, ustawiona początkowo na wartość 1.0, która z kolei przechowuje licznik ilorazu różnicowego.

Inicjowana jest następna pętla *for*, również iterująca po *j* od 0 do *k*, w której następuje obliczenie mianownika ilorazu różnicowego. Pętla ta zawiera warunek sprawdzający różność *j* oraz *i* – czyli sprawdzenie, czy obecny punkt nie jest tym samym, który obliczamy iloraz różnicowy. W takim przypadku aktualizowana jest zmienna *numerator* poprzez pomnożenie jej przez różnicę $x[i]$ i $x[j]$.

Następnie do zmiennej *result* dodawany jest iloraz różnicowy równy wartości funkcji w punkcie $y[i]$ podzielonej przez zmienną *numerator*. Po wykonaniu iteracji wszystkich pętli funkcja *difference_quotient* zwraca obliczony iloraz różnicowy.

```
double difference_quotient(double x[], double y[], int n, int k) {
    if (k == 0) {
        return y[0];
    }

    double result = 0.0;
    for (int i = 0; i <= k; ++i) {
        double numerator = 1.0;
        for (int j = 0; j <= k; ++j) {
            if (j != i) {
                numerator *= (x[i] - x[j]);
            }
        }
        result += y[i] / numerator;
    }
    return result;
}
```

Rys. 1 Fragment kodu przedstawiający funkcję *difference_quotient*

Następnie zaimplementowano funkcję *newton_interpolation*, która służy do obliczania wartości wielomianu Newtona w danym punkcie *input_data* na podstawie danego zestawu punktów. Na samym początku funkcji następuje inicjalizacja zmiennej *result*, która przechowuje obliczoną wartość wielomianu. Wartość początkową tej zmiennej ustawiono na 0.0.

Następnie inicjalizowana jest zmienna *term*, która przechowuje kolejne wyrazy iloczynu wielomianu interpolacyjnego. Wartość początkowa tej zmiennej wynosi 1.0. Rozpoczyna się pętla *for* iterująca po *i* po wszystkich punktach danych aż do *n* (liczby danych punktów). W tej pętli do zmiennej *result* dodawany jest iloraz różnicowy (wywołanie funkcji *difference_quotient* dla punktu $x[i]$ oraz $y[i]$), która zostaje pomnożona przez zmienną *term*.

Na koniec w pętli aktualizowana jest zmienna *term* poprzez pomnożenie jej przez różnicę danej *input_data* oraz $x[i]$, co odpowiada czynnikowi $(x - x_n)$ we wzorze (3). Po wykonaniu wszystkich iteracji pętli *for* zwracana jest wartość zmiennej *result*.

```
double newton_interpolation(double x[], double y[], int n, double input_data) {
    double result = 0.0;
    double term = 1.0;

    for (int i = 0; i < n; ++i) {
        result += difference_quotient(x, y, n, i) * term;
        term *= (input_data - x[i]);
    }

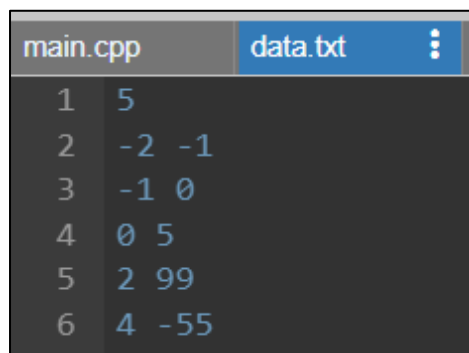
    return result;
}
```

Rys. 2 Fragment kodu przedstawiający funkcję *newton_interpolation*

Dane, na których pracuje funkcja umieszczono w pliku tekstowym, a w funkcji *main* dołączono funkcjonalność czytania z pliku. Następuje to przy pomocy inicjalizacji obiektu *ifstream*, po czym sprawdzany jest warunek otwarcia pliku, celem uniknięcia ewentualnego błędu.

Struktura pliku tekstowego *data.txt* wygląda następująco: pierwsza linijka zawiera jedynie liczbę punktów *n*. Każda następna linijka zawiera już dwie liczby: pierwsza z nich to zmienna *x* danego punktu, a druga jest zmienną *y*. W taki też sposób następuje odczyt z pliku – najpierw zaczytywana jest liczba *n*, a na jej podstawie tworzone są tablice $x[n]$ i $y[n]$, które są później uzupełniane danymi z pliku tekstowego poprzez zastosowanie pętli.

Po zakończeniu odczytywania następuje zamknięcie pliku tekstowego, celem zwolnienia zasobów systemowych i uniknięcia późniejszych ewentualnych problemów z dostępem do pliku, gdyż z dostępu do niego mogą później korzystać różne aplikacje.



main.cpp	data.txt
1	5
2	-2 -1
3	-1 0
4	0 5
5	2 99
6	4 -55

Rys. 3 Plik tekstowy *data.txt* zawierający zadane dane

Na koniec jeszcze wprowadzana z poziomu konsoli jest jedna dana wejściowa - *input_data* - która określa, dla jakiego argumentu funkcji chcemy obliczyć wartość funkcji za pomocą interpolacji Lagrange'a. Dla wprowadzonej liczby wywołujemy funkcję *newton_interpolation* i wynik interpolacji wyświetlamy na ekranie konsoli.

```
int main() {  
  
    ifstream file("data.txt");  
  
    if (!file.is_open()) {  
        cout << "no such file exists" << endl;  
        return 1;  
    }  
  
    int n;  
    file >> n;  
  
    double x[n];  
    double y[n];  
  
    for (int i = 0; i < n; ++i) {  
        file >> x[i] >> y[i];  
    }  
  
    double input_data;  
  
    cout << "input data x for intepolation: ";  
    cin >> input_data;  
  
    double interpolatedValue = newton_interpolation(x, y, n, input_data);  
  
    cout << "interpolated data in point " << input_data << " : " << interpolatedValue << endl;  
  
    file.close();  
  
    return 0;  
}
```

Rys. 4 Fragment kodu przedstawiający funkcję *newton_interpolation*

Po zaimplementowaniu algorytmu sprawdzono poprawność jego działania dla zadanej wcześniej wartości (równej 1). Wynik zgadzał się, co pozwala na stwierdzenie, że algorytm działa poprawnie. Następnie przystąpiono do napisania testów jednostkowych, które mają upewnić się, co do poprawnego działania algorytmu.

```
input data x for intepolation: 1  
interpolated data in point 1 : 44
```

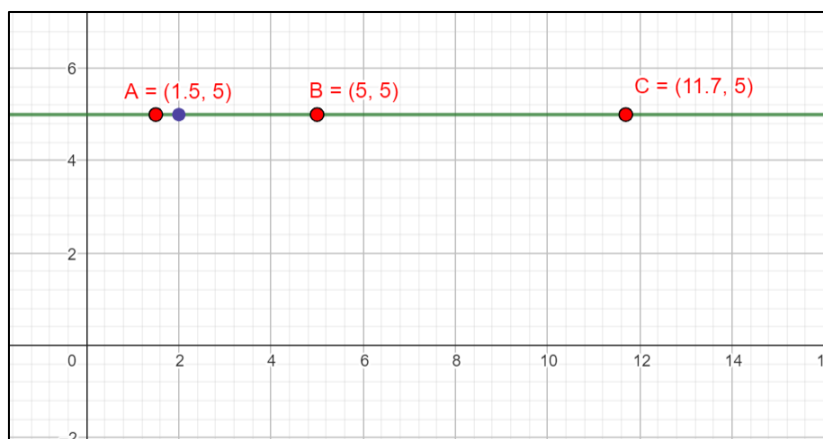
Rys. 5 Zrzut ekranu konsoli programu po wprowadzeniu danej testowej

4. Testy jednostkowe

Obok kompilacji programu sprawdzano poprawność otrzymanych obliczeń w programie PlanetCalc, a załączone niżej wykresy wielomianu wygenerowano w programie Geogebra. Punkty zaznaczone na tych wykresach na niebiesko to punkty wprowadzone do pliku tekstowego, dla algorytmu, natomiast te zaznaczone na czerwono to punkty dla których liczona była interpolacja

4.1. Test jednopunktowy

Test, którego celem jest sprawdzenie, czy funkcja *newton_interpolation* poprawnie zwraca wartość funkcji dla jednego punktu danych ($f(x) = 5$). Wprowadzono zamiany do pliku tekstowego w pierwszej linijce wpisując liczbę 1 (liczba n) a w drugiej kolejno 2 i 5 (współrzędne x i y).



Rys. 6a Wykres funkcji utworzonej dla testu jednostkowego dla jednego punktu z zaznaczonymi punktami poddawanych interpolacji.

Newton Polynomial	
$P_n(x) = 5$	
Newton Polynomial after simplification	
5	
Interpolated Points	
x	8
y	5

Rys. 6b Wartości funkcji w określonych punktach policzone w programie PlanetCalc.

```
input data x for interpolation: 1.5
interpolated data in point 1.5 : 5
```

Rys. 6c Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla jednego punktu dla punktu 1,5.

```
input data x for interpolation: 5
interpolated data in point 5 : 5
```

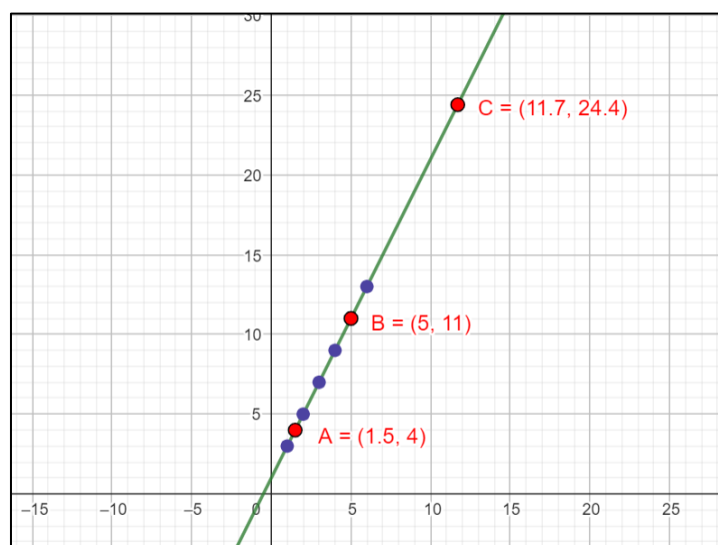
Rys. 6d Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla jednego punktu dla punktu 5.

```
input data x for interpolation: 11.7
interpolated data in point 11.7 : 5
```

Rys. 6e Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla jednego punktu dla punktu 11.7.

4.2. Test liniowy

W tym teście sprawdzane jest, czy funkcja interpolacji działa poprawnie dla liniowego zestawu punktów danych. Oczekiwanym wynikiem są wartości interpolowane rosnące w sposób liniowy zgodnie z danymi wejściowymi. Wybrano 5 punktów wejściowych o współrzędnych: (1, 3), (2, 5), (3, 7), (4, 9), (6, 13), tworząc tym samym funkcję $f(x) = 2x + 1$.



Rys. 7a Wykres funkcji utworzonej dla testu jednostkowego dla danych liniowych z zaznaczonymi punktami poddawanych interpolacji

Newton Polynomial after simplification $2x + 1$			
Interpolated Points			
x	1.5	5	11.7
y	4	11	24.40

Rys. 7b Wartości funkcji w określonych punktach policzone w programie PlanetCalc

```
input data x for interpolation: 1.5
interpolated data in point 1.5 : 4
```

Rys. 7c Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla danych liniowych dla punktu 1,5.

```
input data x for interpolation: 5
interpolated data in point 5 : 11
```

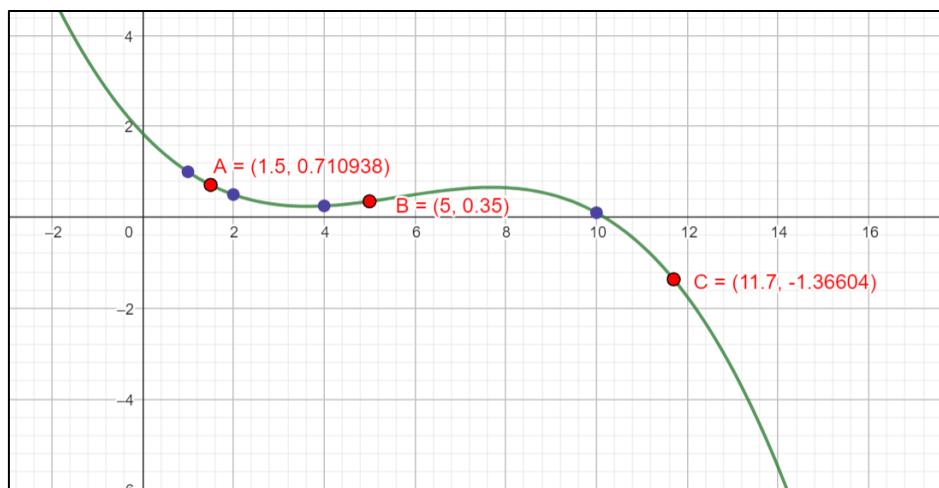
Rys. 7d Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla danych liniowych dla punktu 5.

```
input data x for interpolation: 11.7
interpolated data in point 11.7 : 24.4
```

Rys. 7e Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla danych liniowych dla punktu 11,7.

4.3. Test wielomianowy

Testowana będzie funkcja interpolacji dla zestawu punktów danych pochodzących z wielomianu czwartego stopnia. Wybrano cztery punkty o współrzędnych odpowiednio (1, 1), (2, 0.5) oraz (4, 0.25), (10, 0.1). Utworzono funkcję o wzorze $f(x) = -0,0125x^3 + 0,2125x^2 - 1,05x + 1,85$.



Rys. 8a Wykres funkcji utworzonej dla testu jednostkowego dla testu wielomianowego z zaznaczonymi punktami poddawanych interpolacji

Newton Polynomial after simplification $-0.0125x^3 + 0.2125x^2 - 1.05x + 1.85$			
Interpolated Points			
x	1.5	5	11.7
y	0.7109374999999998	0.35	-1.3660375000000045

Rys. 8b Wartości funkcji w określonych punktach policzone w programie PlanetCalc

```
input data x for interpolation: 1.5
interpolated data in point 1.5 : 0.710938
```

Rys. 8c Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla testu wielomianowego dla punktu 1,5.

```
input data x for interpolation: 5
interpolated data in point 5 : 0.35
```

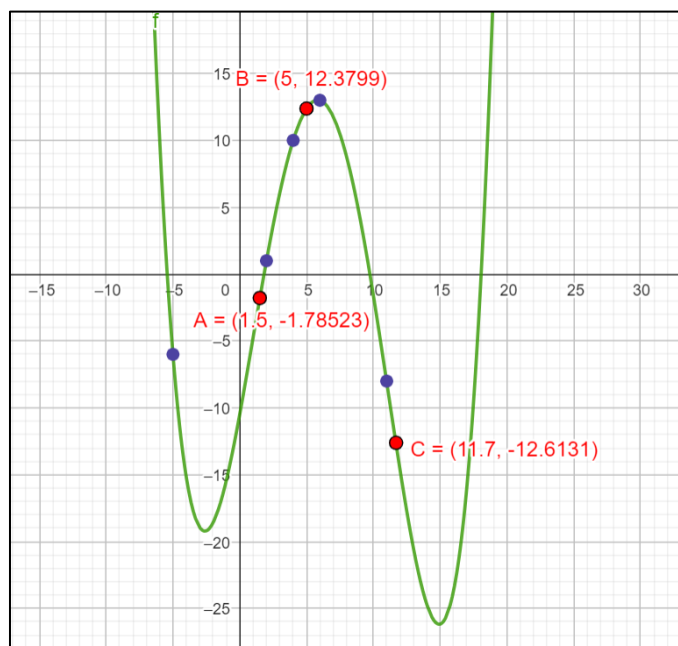
Rys. 8d Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla testu wielomianowego dla punktu 5.

```
input data x for interpolation: 11.7
interpolated data in point 11.7 : -1.36604
```

Rys. 8e Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla testu wielomianowego dla punktu 11,7.

4.4. Test dla danych nieregularnych

W tym teście sprawdzone zostanie, czy funkcja interpolacji radzi sobie poprawnie z nieregularnie rozproszonymi punktami danych. Wybrano pięć punktów o współrzędnych $(-5, -6)$, $(2, 1)$, $(4, 10)$, $(6, 13)$, $(11, -8)$ tworząc funkcję $f(x) = 0,00602x^4 - 0,14571x^3 + 0,39603x^2 + 5.48066x - 10.47619$.



Rys. 9a Wykres funkcji utworzonej dla testu jednostkowego dla danych nieregularnych z zaznaczonymi punktami poddawanych interpolacji

Newton Polynomial after simplification $0.00602x^4 - 0.14571x^3 + 0.39603x^2 + 5.48066x - 10.47619$			
Interpolated Points			
x	1.5	5	11.7
y	-1.825385551948053	12.37987	-12.613060833333321

Rys. 9b Wartości funkcji w określonych punktach policzone w programie PlanetCalc

```
input data x for interpolation: 1.5
interpolated data in point 1.5 : -1.82539
```

Rys. 9c Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla danych nieregularnych dla punktu 1,5.

```
input data x for interpolation: 5
interpolated data in point 5 : 12.3799
```

Rys. 9d Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla danych nieregularnych dla punktu 5.

```
input data x for interpolation: 11.7
interpolated data in point 11.7 : -12.6131
```

Rys. 9e Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla danych nieregularnych dla punktu 11,7.

4.5. Test dla dużego stopnia wielomianu

Podobnie, jak podczas analizy interpolacji Lagrange'a zrezygnowano z odczytu z pliku i wprowadzono do funkcji main pętlę, przydziela wartości tablicowej $x[n]$ i $y[n]$. Liczbę punktów zwiększono do tysiąca. W przypadku tablicy $x[n]$ przydzielane wartości są liczbami całkowitymi równymi liczbie iteracji pętli, natomiast dla tablicy $y[n]$ wartości te są obliczane na podstawie iloczynu tejże liczby iteracji. W domyśle utworzono tym samym funkcję kwadratową rozpoczynającą się w punkcie (0,0) o tysiącu elementach. Poddano sprawdzeniu siedem punktów: 1.5, 5, 11.7, 101, 178, 367, 824.

```
int n = 1000;
double x[n];
double y[n];

for (int i = 0; i < n; ++i) {
    x[i] = i;
    y[i] = i * i;
}

double input_data;

cout << "input data x for interpolation: ";
cin >> input_data;

double interpolatedValue = newton_interpolation(x, y, n, input_data);

cout << "interpolated data in point " << input_data << " : " << interpolatedValue << endl;
```

Rys. 10a Funkcja *main* po zmianie

Interpolated Points							
x	1.5	5	11.7	101	178	367	824
y	2.25	25	136.89000	10201	31684	134689	678976

Rys. 10b Spodziewane wartości funkcji w określonych punktach policzone w programie PlanetCalc dla funkcji x^2

```
input data x for intepolation: 1.5
interpolated data in point 1.5 : -nan
```

Rys. 10c Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla dużych zestawów danych dla punktu 1,5.

```
input data x for intepolation: 5
interpolated data in point 5 : 25
```

Rys. 10d Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla dużych zestawów danych dla punktu 5.

```
input data x for intepolation: 11.7
interpolated data in point 11.7 : -nan
```

Rys. 10e Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla dużych zestawów danych dla punktu 11,7

```
input data x for intepolation: 101
interpolated data in point 101 : 3.99908e+33
```

Rys. 10f Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla dużych zestawów danych dla punktu 11,7

```
input data x for intepolation: 178
interpolated data in point 178 : -nan
```

Rys. 10g Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla dużych zestawów danych dla punktu 11,7

```
input data x for intepolation: 376
interpolated data in point 376 : -nan
```

Rys. 10h Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla dużych zestawów danych dla punktu 11,7

```
input data x for intepolation: 824
interpolated data in point 824 : -nan
```

Rys. 10i Wydruk z konsoli po skompilowaniu programu dla testu jednostkowego dla dużych zestawów danych dla punktu 11,7

5. Opracowanie wyników

Testy jednostkowe wykonane dla mniejszej ilości danych i rozmaitych funkcji zwracają wartości poprawne i o dobrym rozszerzeniu po przecinku; co potwierdzają wykonanie w PlanetCalc obliczenia, od których otrzymane wyniki nie odbiegają. Dla danych użytych w teście wielomianowym i teście dla danych nieregularnych można zauważyć lekkie odchylenia w liczbie miejsc po przecinku – różni się między przypadkami i często skraca się pomimo istnienia dalszego rozwinięcia; chociaż uzyskane tym samym przybliżenie jest prawidłowe.

Dopiero przy dużych zestawach danych widać niedoskonałości interpolacji Newtona. Tylko dla dwóch prób uzyskano wyniki, z czego tylko jeden prawidłowy (dla 5). Pozostałe wartości (1.5, 11.7, 178, 376, 824) nie dały wyniku a jedynie zwróciły błąd *-nan (not a number)*, co potwierdza występowanie efektu Rungego w interpolacji Newtona przy dużej ilości danych punktów.

6. Wnioski

Można zauważyć, że metoda interpolacji Newtona jest skutecznym narzędziem do przybliżania funkcji interpolującej na podstawie zestawu punktów danych. Dzięki prostej do zrozumienia formule oraz efektywnej implementacji, metoda ta umożliwia dokładne przybliżenie funkcji interpolującej.

Przeprowadzając interpolację Newtona na zbiorze danych, uzyskujemy funkcję, która przechodzi przez wszystkie podane punkty danych. Dzięki temu możliwe jest estymowanie wartości funkcji w dowolnym punkcie między danymi wejściowymi, co jest przydatne w wielu dziedzinach, takich jak analiza danych czy modelowanie matematyczne.

Metoda interpolacji Newtona jest stosunkowo łatwa do zrozumienia i implementacji, co czyni ją atrakcyjną opcją dla wielu zastosowań. Jednakże, złożoność obliczeniowa może wzrosnąć wraz z rosnącą liczbą punktów danych, co może być problematyczne dla bardzo dużych zestawów danych. W takich przypadkach, interpolacja Newtona może prowadzić do niepożądanych efektów, takich jak duże oscylacje funkcji interpolującej lub też brak wyniku.

7. Źródła

Wykłady z Metod Numerycznych autorstwa dr hab. Danuty Szeligi

Prezentacja Metody Numeryczne. Interpolacja Newtona autorstwa dr. Hab. Inż. Marcina Hojnego.

Wikipedia – artykuły o interpolacji, interpolacji Newtona, ilorazach różnicowych

<http://www.kosiorowski.edu.pl/wp-content/uploads/2014/09/IS-MetNum-W-S-6-Interpolacja-wielomianowa.pdf>

https://e.kul.pl/files/10382/public/aan_w3_1819_1.pdf

https://home.agh.edu.pl/~chwiej/mn/wyk/interpolacja_22_23.pdf

<https://home.agh.edu.pl/~kipo/dydak/interpolacja.pdf>

Program Geogebra: <https://www.geogebra.org/calculator>

Program PlanetCalc: <https://planetcalc.com>