Maciej Kozub

Laboratorium 4 – interpolacja

1. Funkcja testowa i struktury

Jako funkcja testowa wybrana została $f(x) = e^{\cos x(x)}$, przedział którym będziemy się zajmować to [-5, 5]. Stworzona została również struktura reprezentująca punkt (argument, wartość funkcji dla tego argumentu) oraz zdefiniowana pochodna funkcji f. Wartości funkcji w poniższych funkcjach będą obliczane w 20000 punktach. Kolejne testowane liczby węzłów interpolacyjnych to: 2, 3, 5, 7, 10, 12, 15, 20, 30, 35.

```
struct point {
    double x;
    double y;
};

double function(double x) {
    return exp(cos(x));
}

double derivative(double x) {
    return -sin(x) * exp(cos(x));
}
```

W kolejnym kroku zaimplementowano funkcję zwracającą dokładną wartość funkcji testowej w podanej liczbie punktów, tak aby można ją również było wykorzystać do wyznaczenia wartości węzłów interpolacyjnych dla rozmieszczenia równoodległego.

```
std::vector<point> getPoints(double function(double x), int pointsNumber) {
    std::vector<point> values;
    double interval = (END - START) / (pointsNumber - 1);
    for (int i = 0; i < pointsNumber; i++) {
        point point{};
        point.x = START + interval * i;
        point.y = function(point.x);
        values.push_back(point);
    }
    return values;
}
```

2. Interpolacja Lagrange'a

Metoda opiera się na utworzeniu wielomianu na podstawie sumy iloczynów funkcji liniowych, przechodzących przez kolejne węzły interpolacyjne.

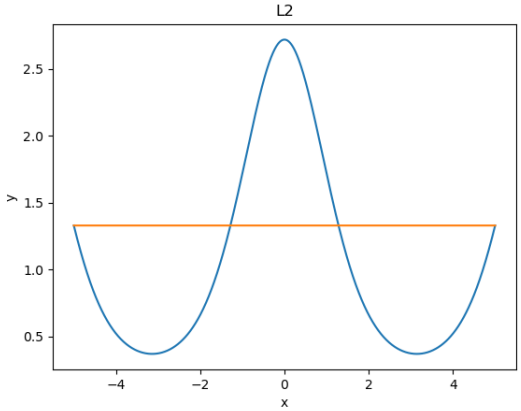
Dla węzłów równoodległych zaimplementowano dwie funkcje, odpowiednio wyliczającą wartość funkcji interpolacyjnej w danym punkcie oraz wyliczającą wartość funkcji na całym przedziale.

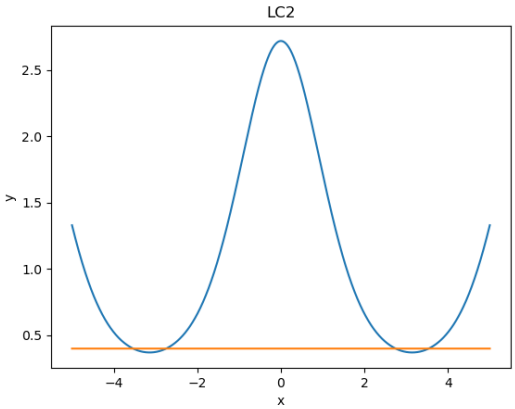
```
double lagrangeInPoint(std::vector<point> values, double xi) {
    int n = values.size();
    double result = 0;
    for (int i = 0; i < n; i++) {
        double yi = values[i].y;
        for (int j = 0; j < n; j++) {
            if (i != j) {
                yi = yi * (xi - values[j].x) / (values[i].x - values[j].x);
            }
        }
        result += yi;
    }
    return result;
}

std::vector<point> lagrange(int pointsNumber, bool chebyshev) {
    std::vector<point> results;
    std::vector<point> values;
    if (!chebyshev)
        values = getPoints(function, pointsNumber);
    else
        values = getChebyshevPoints(function, pointsNumber);
    double interval = LENGTH / (STEPS - 1);
    for (int i = 0; i < STEPS; i++) {
        point point{};
        point.x = interval * i - END;
        point.y = lagrangeInPoint(values, point.x);
        results.push_back(point);
    }
    return results;
}
```

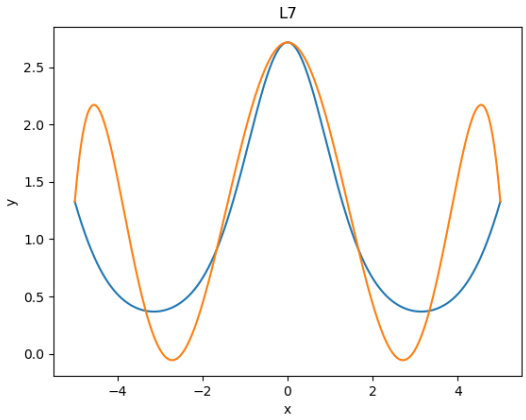
Dla węzłów Czebyszewa potrzebna jest dodatkowo funkcja wyznaczająca miejsce zerowe wielomianu Czebyszewa o danym stopniu.

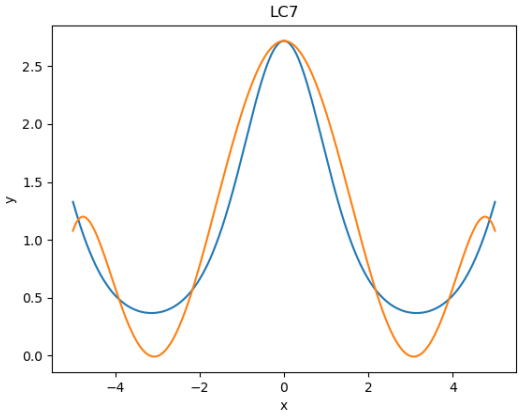
```
std::vector<point> getChebyshevPoints(double function(double), int pointsNumber) {
    std::vector<point> values;
    for (int k = pointsNumber - 1; k >= 0; k--) {
        point point{};
        point.x = LENGTH / 2 * cos( X: (M_PI * (2 * k + 1)) / (2 * (pointsNumber)));
        point.y = function(point.x);
        values.push_back(point);
    }
    return values;
}
```

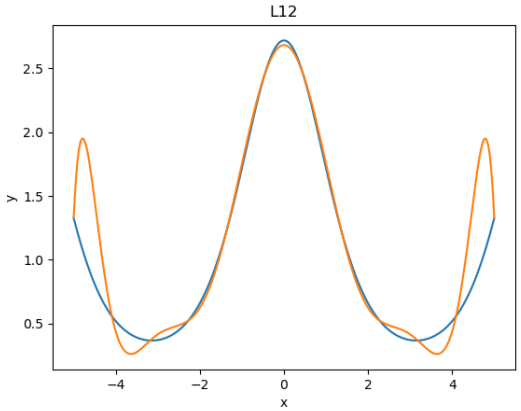
Poniżej znajduje się porównanie (niektórych) wyników, tj. wykresu funkcji interpolowanej (kolor niebieski) i interpolującej (kolor pomarańczowy) dla różnej liczby węzłów zarówno równoodległych jak i Czebyszewa.

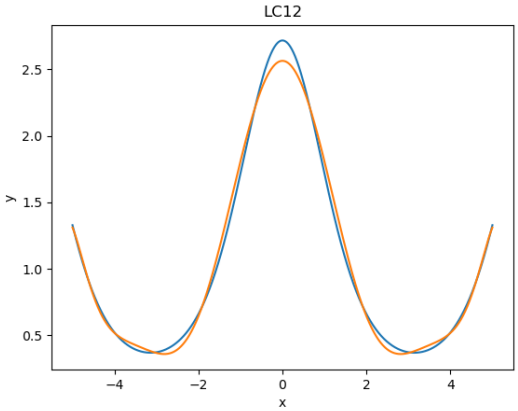


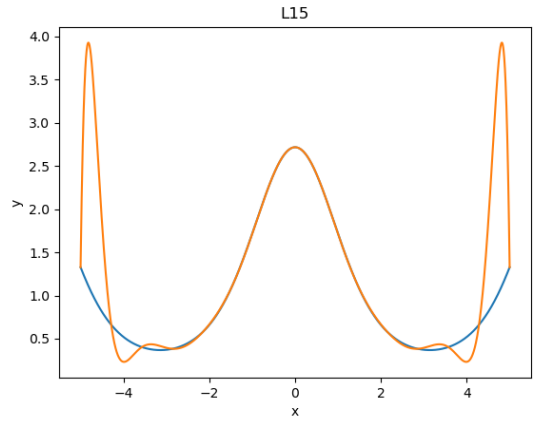


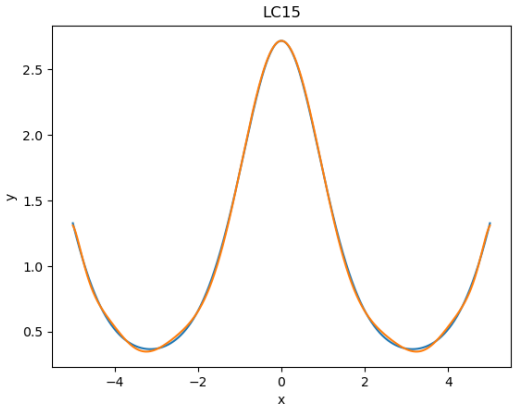


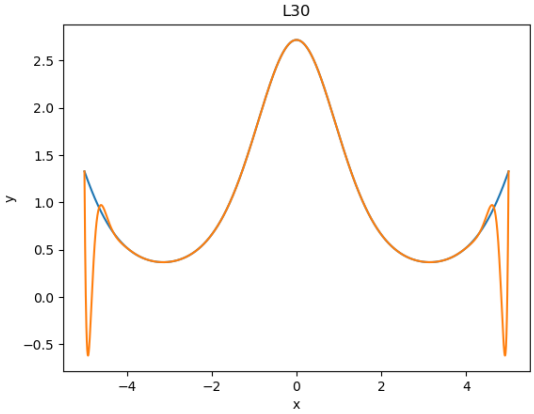


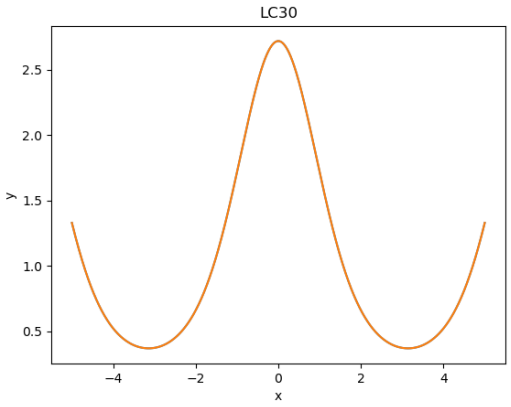


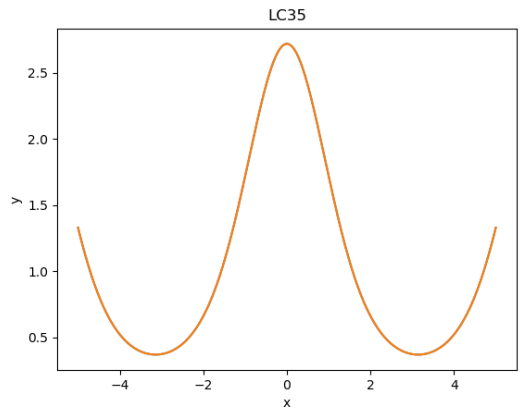


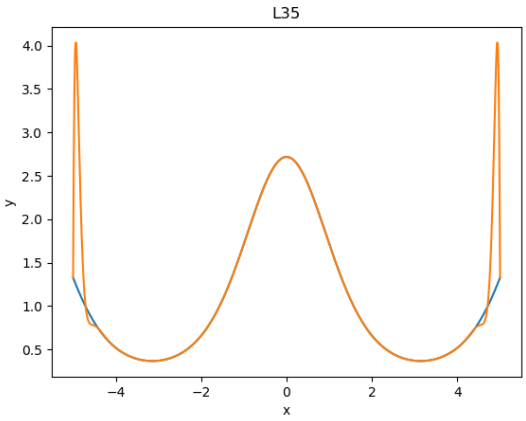












Jak można zauważyć w porównywanych przypadkach interpolacja z węzłami Czebyszewa jest dokładniejsza. Dla liczby węzłów większej od 15 wielomian Czebyszewa prawie pokrywa się z funkcją interpolowaną na przedziale (-3, 3), oddalając się od punktu (0,0) na osi można zauważyć efekt Runge'go.

W celu analizy błędu interpolacji zaimplementowano poniższą funkcję liczącą sumę wartości błędu interpolacji na 20000 punktach przedziału.

```
double totalError(std::vector<point> interpolation(int n, bool ch), int n, bool chebyshev) {
    double error = 0.0;
    double interval = LENGTH / (STEPS - 1);
    std::vector<point> results;
    results = interpolation(n, chebyshev);
    for (int i = 0; i < STEPS; i++) {
        error += std::abs( x: function( x: i * interval - END) - results[i].y);
    }
    return error;
}
```

Funkcja ta dokonuje obliczeń dla ilości węzłów z przedziału (1, 100). Częściowe wyniki znajdują się poniżej.

Stopień wielomianu	w. równoodległe	w. Czebyszewa
3	2435.11	1888.32
10	320.002	182.027
20	211.836	4.05293
36	88.3905	0.00225225
46	32.5192	1.16845e-05
58	6.40933	1.38835e-08
66	646.556	1.26385e-10
76	492534	1.39427e-12
77	1.02828e+06	1.23801e-12
90	6.07482e+09	1.61854e-12
99	3.14893e+12	1.44962e-12

W przypadku rozmieszczenia równoodległego błąd początkowo maleje, następnie od liczby węzłów większej niż 58 zaczyna gwałtownie rosnąć. W przypadku węzłów Czebyszewa błąd maleje wraz ze wzrostem stopnia wielomianu. W metodzie węzłów równoodległych najlepszym wielomianem interpolującym okazał się wielomian o stopniu 58.

Dobrym przykładem wystąpienia efektu Runge'go jest wielomian stopnia 15. Pomimo zwiększenia ilości węzłów nastąpiło pogorszenie dokładności interpolacji.

3. Interpolacja Newtona

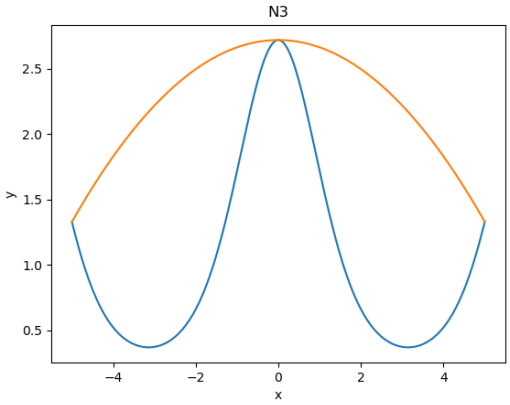
Metoda ta polega na wyznaczaniu kolejnych ilorazów różnicowych i sumowaniu iloczynów tych ilorazów i funkcji liniowych. W celu realizacji tej metody zaimplementowano dwie funkcje pomocnicze, obliczającą iloraz n funkcji liniowych – newtonHelper oraz newtonDivDiffTable – obliczającą tablicę wypełnioną ilorazami różnicowymi.

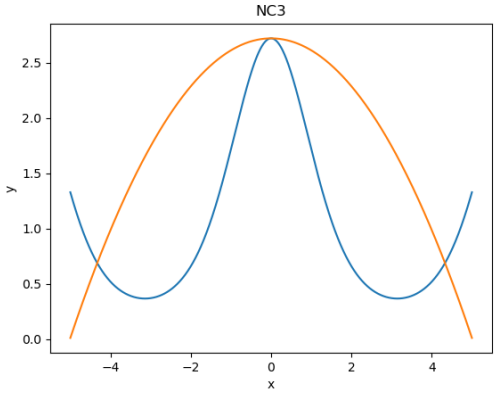
```
double newtonHelper(int n, double x, std::vector<point> values) {
    double result = 1;
    for (int i = 0; i < n; i++) {
        result = result * (x - values[i].x);
    }
    return result;
}

std::vector<std::vector<double>> newtonDivDiffTable(std::vector<point> values, int n) {
    std::vector<std::vector<double>> result;
    std::vector<double> empty_vec;
    empty_vec.resize(n, x: -1.0);
    result.resize(n, empty_vec);

    for (int i = 0; i < n; i++) {
        result[i][0] = values[i].y;
    }
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n - i; j++) {
            result[j][i] = 0.0;
            result[j][i] = (result[j][i - 1] - result[j + 1][i - 1]) / (values[j].x - values[i + j].x);
        }
    }
    return result;
}
```

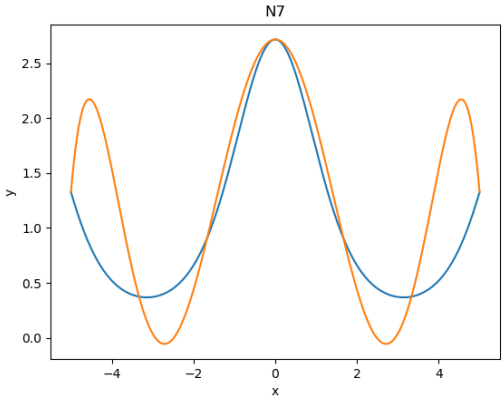
Następnie zaimplementowano właściwy algorytm interpolujący Newtona (wartość w punkcie oraz na całym przedziale).

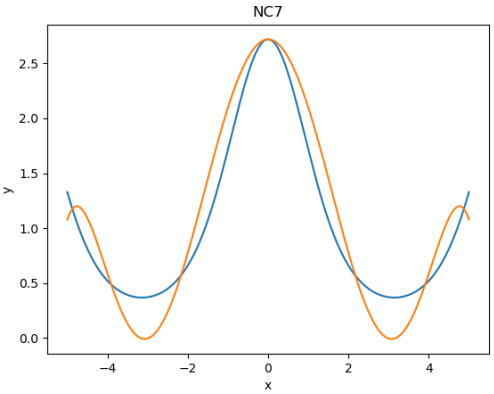
```
double newtonInPoint(const std::vector<point>& values, int n, double x) {
    std::vector<std::vector<double>> y = newtonDivDiffTable(values, n);
    double sum = y[0][0];
    for (int i = 1; i < n; i++) {
        sum = sum + (newtonHelper(i, x, values) * y[0][i]);
    }
    return sum;
}

std::vector<point> newton(int pointsNumber, bool chebyshev) {
    std::vector<point> results;
    std::vector<point> values;
    if(!chebyshev)
        values = getPoints(function, pointsNumber);
    else
        values = getChebyshevPoints(function, pointsNumber);
    double interval = LENGTH / (STEPS - 1);
    for (int i = 0; i < STEPS; i++) {
        point point{};
        point.x = interval * i - END;
        point.y = newtonInPoint(values, pointsNumber, point.x);
        results.push_back(point);
    }
    return results;
}
```

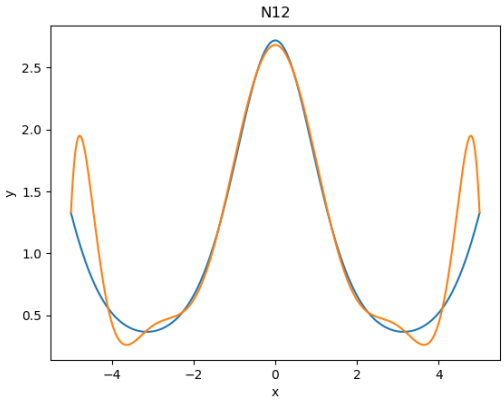
Analogicznie do metody Lagrange’a przedstawiono poniżej niektóre wyniki uzyskane za pomocą tej metody.

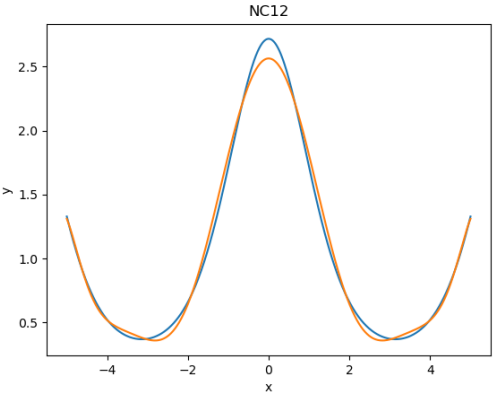

N3
x
y

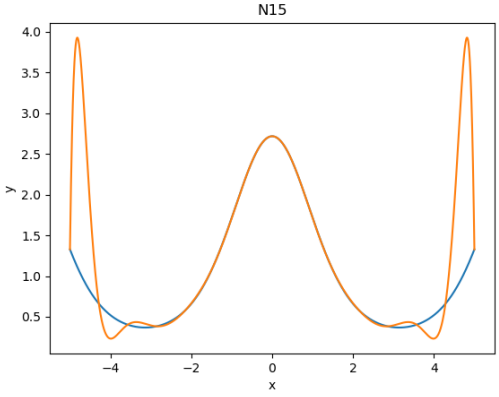

NC3
x
y

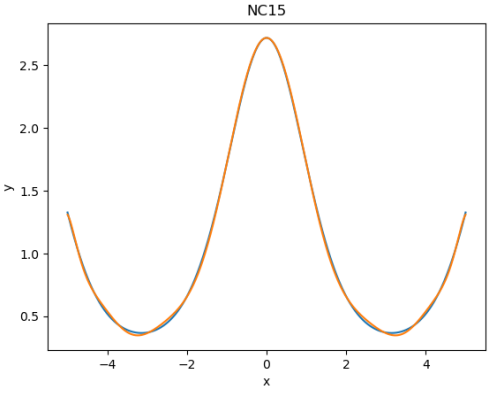

N7
x
y

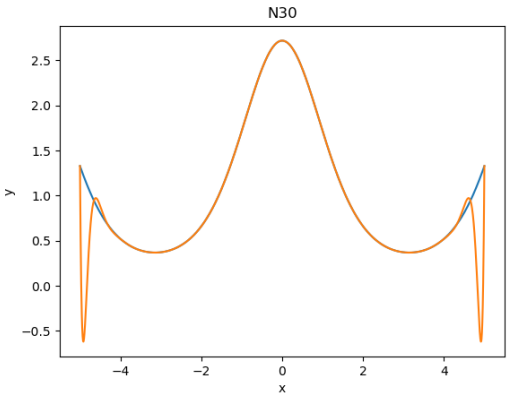

NC7
x
y

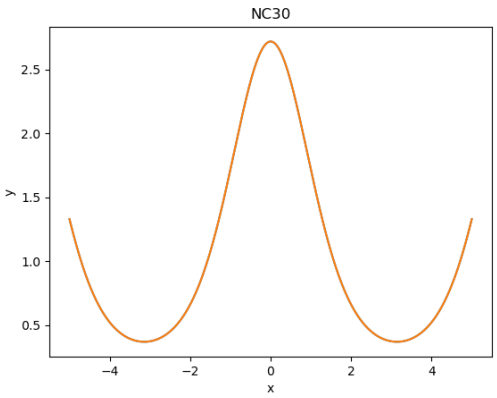

N12
x
y

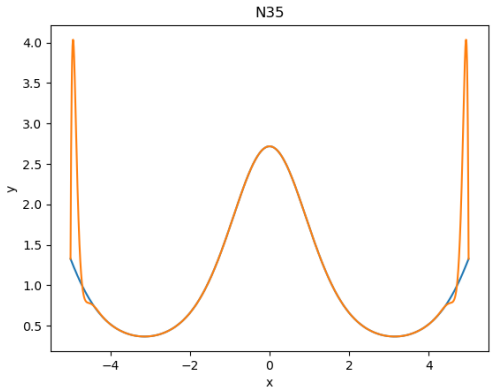

NC12
x
y

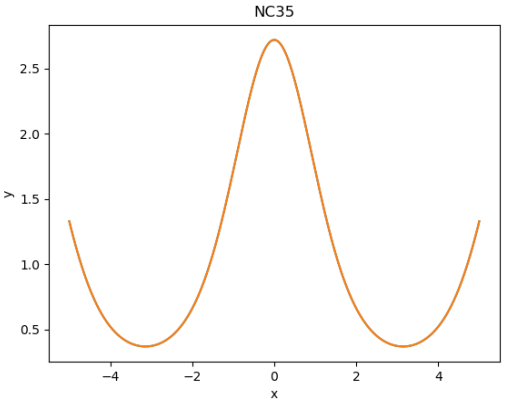

N15
x
y


NC15
x
y









Ponownie wykorzystując węzły Czebyszewa osiągnięto lepszą dokładność. Wyznaczanie błędu interpolacji przebiega analogicznie do metody Lagrange'a. Częściowe wyniki przedstawiono poniżej.

Stopień wielomianu	w. równoodległe	w. Czebyszewa
3	2435.11	1888.32
14	245.061	44.2026
15	512.807	25.2397
36	88.3905	0.00225222
43	75.7566	4.41131e-05
56	8.94849	0.000869227
66	18.5592	1.05279
78	43405.6	140908
79	210651	1.02342e+07
99	1.21095e+14	5.31523e+16

W tym przypadku najlepiej dopasowanym wielomianem okazał się wielomian o stopniu 62. Ponownie w przypadku węzłów równoodległych wartość błędu początkowo niestabilnie maleje a następnie gwałtownie rośnie począwszy od liczby węzłów równej 80. W przypadku węzłów Czebyszewa inaczej niż w interpolacji Lagrange'a po początkowym zwiększaniu precyzji interpolacji następuje gwałtowny wzrost błędu. Ma to miejsce przy liczbie węzłów większej bądź równej 49.

Efekt Runge'go można zauważyć na przykład analizując wykres dla ilości węzłów 30 i 35.

4. Interpolacja Hermite'a

Interpolacja Hermite'a jesy podobna do metody Newtona, również wykorzystujemy w niej ilorazy różnicowe. Różnica polega na używaniu pochodnej funkcji w celu zwiększenia precyzji przybliżenia. Do tworzonej tablicy ilorazów różnicowych każdy węzeł dodajemy n+1 razy, gdzie n to stopień najwyższej używanej pochodnej. Wykorzystując zdefiniowaną wcześniej pochodną zaimplementowano metodę Hermite'a dla jednej pochodnej.

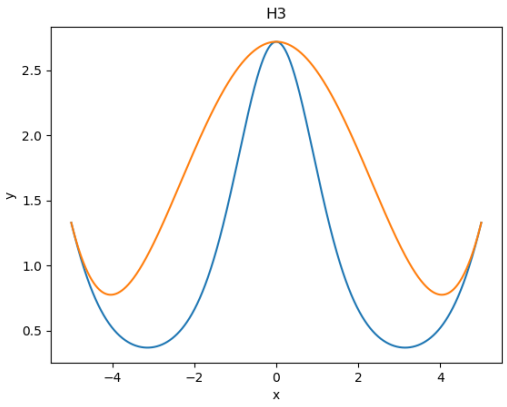
```
std::vector<std::vector<double>> hermiteDivDiffTable(std::vector<point> values, int n) {
    std::vector<std::vector<double>> result;
    std::vector<double> empty_vec;
    empty_vec.resize( new_size: 2 * n,  x: -1.0);
    result.resize( new_size: 2 * n, empty_vec);

    std::vector<point> values2;

    for (int i = 0; i < n; i++) {
        values2.push_back(values[i]);
        values2.push_back(values[i]);
    }

    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < i + 1; j++) {
            if (j == 0) result[i][j] = function(values2[i].x);
            else if (j == 1 && i % 2 == 1) result[i][j] = derivative(values2[i].x);
            else {
                result[i][j] = result[i][j - 1] - result[i - 1][j - 1];
                result[i][j] = result[i][j] / (values2[i].x - values2[i - j].x);
            }
        }
    }

    return result;
}
```

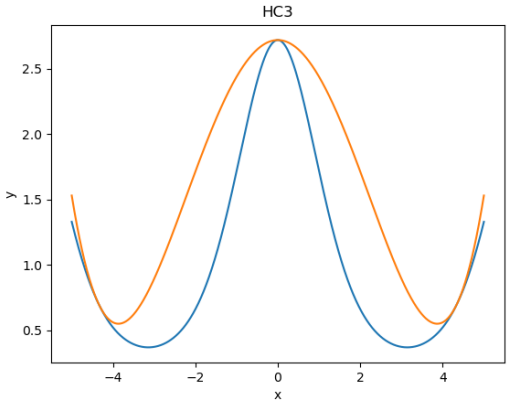
Następnie zaimplementowano właściwy algorytm interpolujący Newtona (wartość w punkcie oraz na całym przedziale).

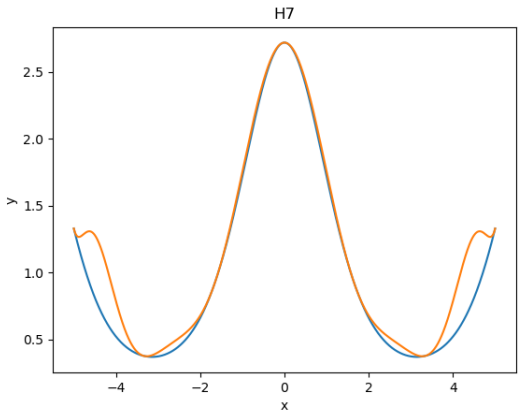
```
double hermiteInPoint(std::vector<point> values, int n, double x) {
    std::vector<point> values2;

    for (int i = 0; i < n; i++) {
        values2.push_back(values[i]);
        values2.push_back(values[i]);
    }

    std::vector<std::vector<double>> y = hermiteDivDiffTable(values, n);
    double result = 0.0;
    double component = 1.0;
    for (int i = 0; i < 2 * n; i++) {
        result = result + y[i][i] * component;
        component = component * (x - values2[i].x);
    }
    return result;
}
```

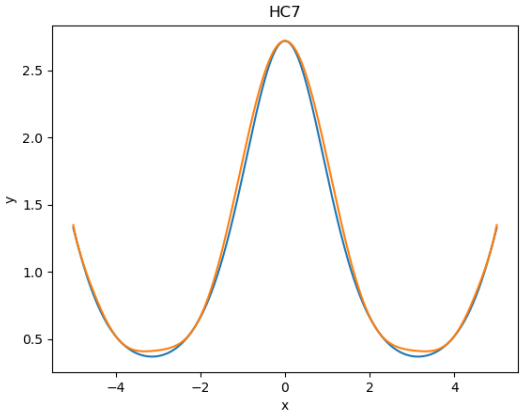
```cpp
std::vector<point> hermite(int pointsNumber, bool chebyshev) {
    std::vector<point> results;
    std::vector<point> values;
    if(!chebyshev)
        values = getPoints(function, pointsNumber);
    else
        values = getChebyshevPoints(function, pointsNumber);
    double interval = LENGTH / (STEPS - 1);
    for (int i = 0; i < STEPS; i++) {
        point point{};
        point.x = interval * i - END;
        point.y = hermiteInPoint(values, pointsNumber, point.x);
        results.push_back(point);
    }
    return results;
}
```

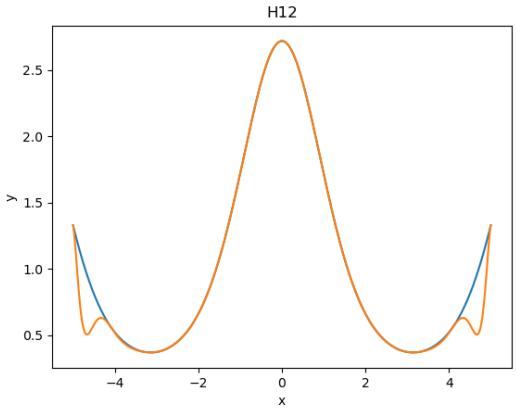
Analogicznie do poprzednich punktów poniżej przedstawiono (wybrane) porównania funkcji interpolującej i interpolowanej zarówno dla węzłów Czebyszewa jak i równoodległych dla różnej liczby węzłów interpolacyjnych.

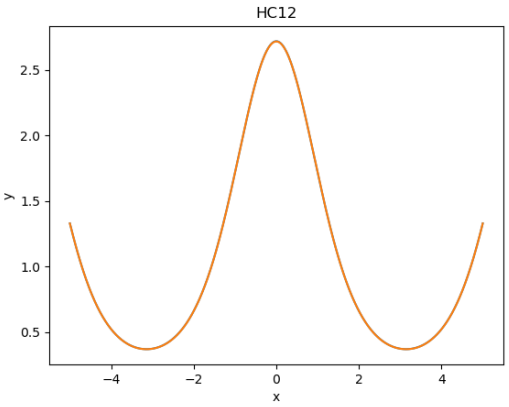


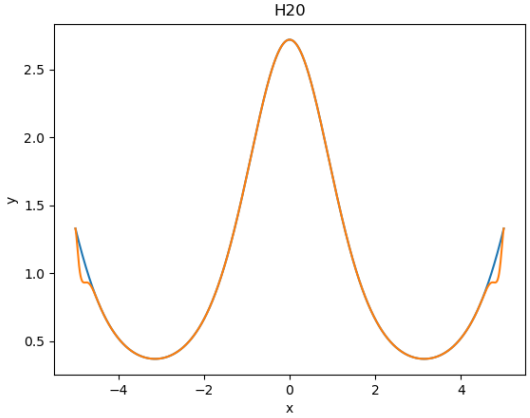


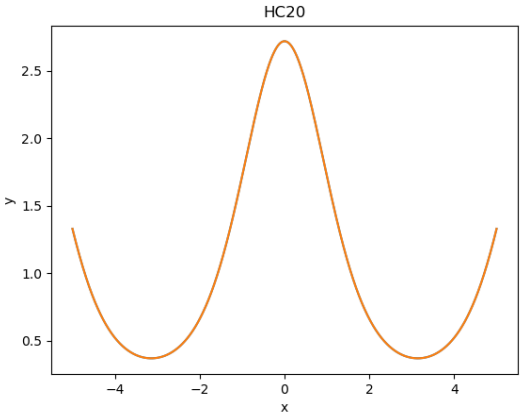


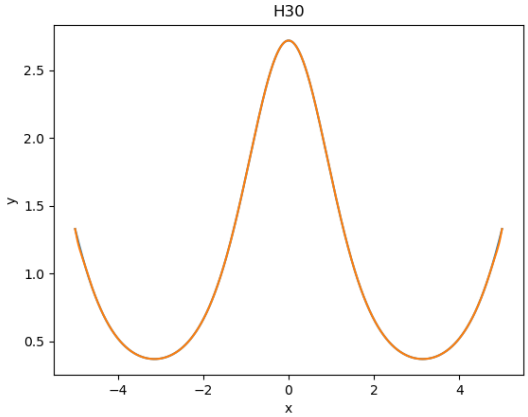


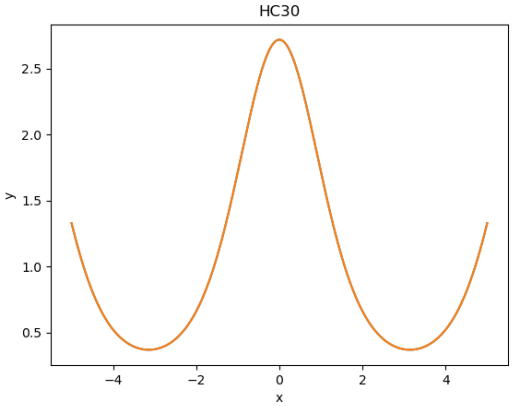

H12
y
x

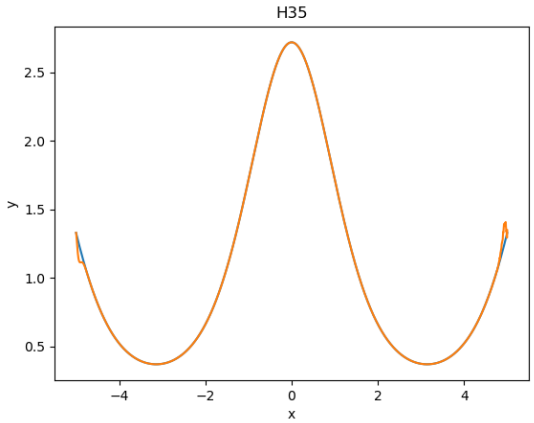

HC12
y
x

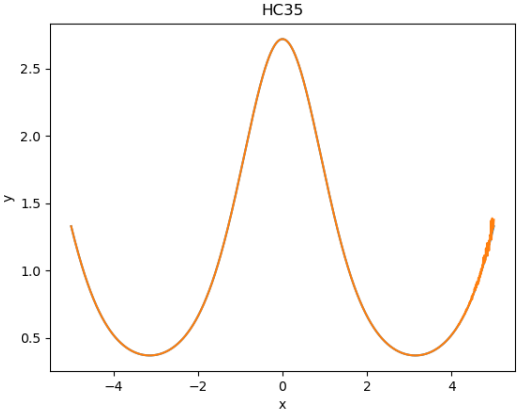

H20
y
x


HC20
y
x


H30
y
x


HC30
y
x


H35
y
x


HC35
y
x

Podobnie do wcześniejszych punktów interpolacja z wykorzystaniem węzłów Czebyszewa jest dokładniejsza niż dla węzłów równoodległych. Z analizy wykresów można również wnioskować że metoda Hermite'a jest dokładniejsza niż wcześniej testowana metoda Lagrange'a i Newtona, niedokładność interpolacji Hermite'a dla 12 węzłów Czebyszewa jest niedostrzegalna dla ludzkiego oka.

Analizy błędów dokonano analogicznie do poprzednich punktów z wykorzystaniem tej samej funkcji. Częściowe wyniki przedstawiono poniżej.

Stopień wielomianu	w. równoodległe	w. Czebyszewa
2	3660.36	1458.63
3	1216.02	952.048
20	17.2632	0.000449297
23	8.20448	4.34842e-05
30	1.03029	0.00800894
32	0.512896	0.050478
40	10926.1	756591
41	33299.5	3.69744e+07
50	1.39163e+13	1.07551e+16
90	1.86345e+51	1.49207e+56
99	2.84534e+59	3.1108e+65

W przypadku węzłów równoodległych najlepiej dopasowanym wielomianem okazał się wielomian o stopniu 32. Wartość błędu początkowo maleje dla obydwóch sposobów rozłożenia węzłów interpolacyjnych. Następnie dla węzłów równoodległych od n = 33 a dla węzłów Czebyszewa n = 24 błąd zaczyna gwałtownie rosnąć. Dla węzłów Czebyszewa zjawisko to wystąpiło już przy interpolacji metodą Newtona.

Efekt Runge'go można zauważyć obserwując na wykresach powyżej funkcję interpolującą dla liczby węzłów 30 i 35.