Metody numeryczne

Nikodem Kocjan

16.04.2024

Interpolacja funkcjami sklejanymi w bazie.

Spis treści

1 Cel ćwiczenia

Celem ćwiczenia jest interpolacja z pomocą funkcji sklejonymi w bazie.

2 Wstęp teorytyczny

2.1 Interpolacja funkcjami sklejonymi w bazie

Interpolacja funkcjami sklejanymi stanowi jedną z fundamentalnych metod w numerycznej analizie danych, szczególnie przydatną w przypadkach, gdy potrzebujemy wygładzić dane lub aproksymować funkcje w sposób ciągły przez określone przedziały. Metoda ta wykorzystuje tzw. funkcje sklejane, które lokalnie aproksymują funkcję, ale globalnie tworzą ciągłą i gładką krzywą interpolacyjną.

2.2 Definicja funkcji sklejanej

Funkcje sklejane, zwłaszcza kubiczne, są definiowane na przedziałach między węzłami interpolacji. Każdy fragment krzywej interpolacyjnej, określony jako $\Phi_{3i}(x)$, jest kubicznym wielomianem, który jest niezerowy tylko w ograniczonym przedziale, co redukuje problem do lokalnego zagadnienia.

$$\Phi_{3i}(x) = \frac{1}{h^3} \begin{cases} (x - x_{i-2})^3 & \text{dla } x \in [x_{i-2}, x_{i-1}) \\ h^3 + 3h^2(x - x_{i-1}) + 3h(x - x_{i-1})^2 - 3(x - x_{i-1})^3 & \text{dla } x \in [x_{i-1}, x_i) \\ h^3 + 3h^2(x_{i+1} - x) + 3h(x_{i+1} - x)^2 - 3(x_{i+1} - x)^3 & \text{dla } x \in [x_i, x_{i+1}) \\ (x_{i+2} - x)^3 & \text{dla } x \in [x_{i+1}, x_{i+2}) \\ 0 & \text{inny niż posotałe przypadki} \end{cases}$$

2.3 Interpolacja i warunki brzegowe

Interpolacja za pomocą funkcji sklejanych wymaga rozwiązania układu równań, który wykorzystuje zarówno wartości funkcji w węzłach, jak i dodatkowe warunki brzegowe, które zwykle dotyczą pierwszych pochodnych funkcji w skrajnych punktach przedziału interpolacji:

$$-c_0 + c_2 = \frac{h}{3}\alpha$$
$$-c_{n-1} + c_{n+1} = \frac{h}{3}\beta$$

gdzie α i β są pochodnymi funkcji interpolowanej na krańcach przedziałów interpolacji.

3 Deklaracja zmiennych oraz implementacja biliotek

Opisane zagadnienie rozwiązano za pomocą programu Visual Studio Code w językі C++. Wykorzystano zewnętrzną bibliotekę GSL w celu ułatwienia działania na macierzach, oraz rozwiązywania układów równań liniowych. Wykorzystano biblioteki

- <iostream> standardowa biblioteka C++
- <cmath> przydatne funkcje matematyczne
- <vector> obiekt vector dla niektórych zmiennych
- <gsl_math.h> obliczenia biblioteki GSL
- <gsl_linalg.h> rozwiązywanie układów liniowych GSL
- <gsl_eigen.h> zainkludowanie przestrzeni
- <gsl_sf_bessel.h>

Wykorzystano zewnętrzną bibliotekę plotly, do generowania wykresów.

Zadeklarowane stałe, macierze oraz wektory użyte w programie to:

```
double x_start = -5;    //poczatek przedzialu
double x_end = 5;       //koniec przedzialu
double x_delta = 0.01;  //delta
int n = 5; //liczba wezlow
double h = (x_end - x_start) / (n-1); //odleglosc miedzy sasiednimi wezlami
vector<double>xx(n+6); // tablica xx rozmiaru n+6
vector<double>x_tab(n+4); //tablica x rozmiaru n+4
vector<double>y_tab(n+4); //tablica y rozmiaru n+4
//macierz A ukladu rownan
gsl_matrix *a = gsl_matrix_alloc(n,n);
//wektor b ukladu rownan
gsl_vector *b = gsl_vector_alloc(n);
//wektor c rozwiazania ukladu rownan
gsl_vector *cold = gsl_vector_calloc(n);
vector<double>cnew; // rozszerzony wektor c
```

Fragment kodu 1: Deklaracja stalych

4 Opis działania

4.1 Krótki opis programu

Program podzielony jest na kilka funkcji :

- fi - zwraca wartość Φ
- f_basic - zwraca wartość s(x)

- f1 - zwraca wartość funkcji 1 dla argumenu x
- f1_poch - zwraca wartość pochodnej funkcji 1 dla argumenu x
- f2 - zwraca wartość funkcji 2 dla argumenu x
- f2_poch - zwraca wartość pochodnej funkcji 2 dla argumenu x
- main - główna funkcja

4.2 Funkcja fi

Funkcja jest odpowiednikiem wzoru z podpunktu 2.2, zwraca wartość Φ zależnie od wprowadzonych danych. Wykorzystywana do wyznaczania s(x).

```
double fi(double x, double x_tab_m2,double x_tab_m1,double x_tab, double x_tab_p1,double x_tab_p2 ,double h){
        if(x >= x_tab_m2 && x < x_tab_m1 ){
            double result = (x - x_tab_m2) * (x - x_tab_m2) * (x - x_tab_m2);
            result = (1.0 / (h * h * h)) * result;
            return result;
        }else if(x >= x_tab_m1 && x < x_tab){
            double result =( h * h * h )+ (3.0*h*h*(x - x_tab_m1)) + (3.0*h*(x-x_tab_m1)*(x-x_tab_m1)) - (3.0*(x-x_tab_m1)*(x - x_tab_m1)*(x - x_tab_m1));
            result = (1.0 / (h * h * h)) * result;
            return result;
        }else if(x >= x_tab && x < x_tab_p1){
            double result = h*h*h + 3.0*h*h*(x_tab_p1 - x) + 3.0*h*(x_tab_p1 - x)*(x_tab_p1 - x) - 3.0*(x_tab_p1 - x)*(x_tab_p1 - x)*(x_tab_p1 - x);
            result = (1.0 / (h * h * h)) * result;
            return result;
        }else if(x >= x_tab_p1 && x < x_tab_p2){
            double result = (x_tab_p2 - x)*(x_tab_p2 - x)*(x_tab_p2 - x);
            result = (1.0 / (h * h * h)) * result;
            return result;
        }else {
            return 0.0;
        }
}
```

Fragment kodu 2: Funkcja wyznaczająca wartość phi

4.3 Funkcja f_basic

Funkcja zwraca wartość $s(x)$.

```
f_basic(double h, vector<double> xtmp, vector<double> c, double x){
    double sum = 0.0;
    int n = c.size();
    for(int i = 0; i < n; ++i){
        sum += c[i] * fi(x, xtmp[i],xtmp[i+1],xtmp[i+2],xtmp[i+3], xtmp[i+4], h);
```

```
    }
    return sum;
}
```

Fragment kodu 3: Funkcja zwracająca wartość s(x)

4.4 Funkcja f1

```
double f1(double x){
    double result = 1 / (1 + x * x);
    return result;
}
```

Fragment kodu 4: Funkcja zwracająca wartość f1(x)

4.5 Pochodna f1

```
double f1_poch(double x){
    double result = (f1(x + x_delta) - f1(x - x_delta)) / (2 * x_delta);
    return result;
}
```

Fragment kodu 5: Funkcja zwracająca wartość pochodnej f1(x)

4.6 Funkcja f2

```
double f2(double x){
    double result = cos(2 * x);
    return result;
}
```

Fragment kodu 6: Funkcja zwracająca wartość f2(x)

4.7 Pochodna f2

```
double f2_poch(double x){
    double result = (f2(x + x_delta) - f2(x - x_delta)) / (2 * x_delta);
    return result;
}
```

Fragment kodu 7: Funkcja zwracająca wartość pochodnej f2(x)

4.8 Funkcja main

Na początku ustalono wartości n. Na jej podstawie obliczono h, oraz wyznaczono alpha oraz beta.

```
    int n = 5; //liczba wezlow
    double h = (x_end - x_start) / (n-1); //odleglosc miedzy
    sasiednimi wezlami
    double alpha = f1_poch(x_start);
    double beta = f1_poch(x_end);
```

Fragment kodu 8: Ustawienie zmiennych początkowych

Nastepnie uzupełniono tablice xx, x_tab, y_tab.

```
    for(int i = -2; i < (n + 4); ++i){
        xx[i+2] = x_start + h * (i-1);
    }
    for(int i = 1; i < (n + 5); ++i){
        x_tab[i-1] = xx[i+1];
    }
    for (int i = 0; i < (n + 4); ++i)
    {
        y_tab[i] = f1(x_tab[i]);
    }
```

Fragment kodu 9: Uzupełnienie tablic

Kolejnym krokiej jest rozwiązanie układu równań. W tym celu zadeklarowano macierz A, oraz wektor b, oraz wypełniono je danymi zgodnie z algorytmem.

```
    //Ustawienie wektora b
    for(int i = 1; i < n; ++i){
        gsl_vector_set(b,i,f1(x_tab[i+1] - beta * h / 3));
    }
    gsl_vector_set(b,0,f1(x_tab[1]) + alpha * h / 3);
    gsl_vector_set(b,n-1,f1(x_tab[1]) - beta * h / 3);

    //ustawienie macierzy A
    for(int i = 1; i<n-1;++i){
        gsl_matrix_set(a,i,i,4);
        gsl_matrix_set(a,i+1,i,1);
        gsl_matrix_set(a,i,i+1,1);
    }

    gsl_matrix_set(a,0,0,4.0);
    gsl_matrix_set(a,n-1,n-1,4.0);
    gsl_matrix_set(a,1,0,1.0);
    gsl_matrix_set(a,0,1,2.0);
    gsl_matrix_set(a,n-1,n-2,2.0);
```

Fragment kodu 10: Przygotowanie do układu równań

Z pomocą GSL rozwiązano układ równań. Rezultat zapisano do wektora cold o rozmiarze n. Następnie dodano wartość na początku oraz na końcu wektora tworząc wektor cnew.

```
for(int i = 0; i < n;i++){
        cnew.push_back(gsl_vector_get(cold,i));
```

```
    }
    tmp = cnew[1] - alpha * h /3;
    cnew.insert(cnew.begin(), tmp);
    tmp = cnew[cnew.size() - 2] + (h / 3) * beta;
    cnew.push_back(tmp);
```

Fragment kodu 11: Wyznaczenie wektora cnew

W ostatnim kroku wyznaczono wartosci funkcji przy pomocy funkcji f_basic.

```
    for(int i = 0; i < 100; i++){
        double x = x_start + (i / 10.0);
        tmp = f_basic(h, xx, cnew,x );
        cout << tmp << endl;
    }
```

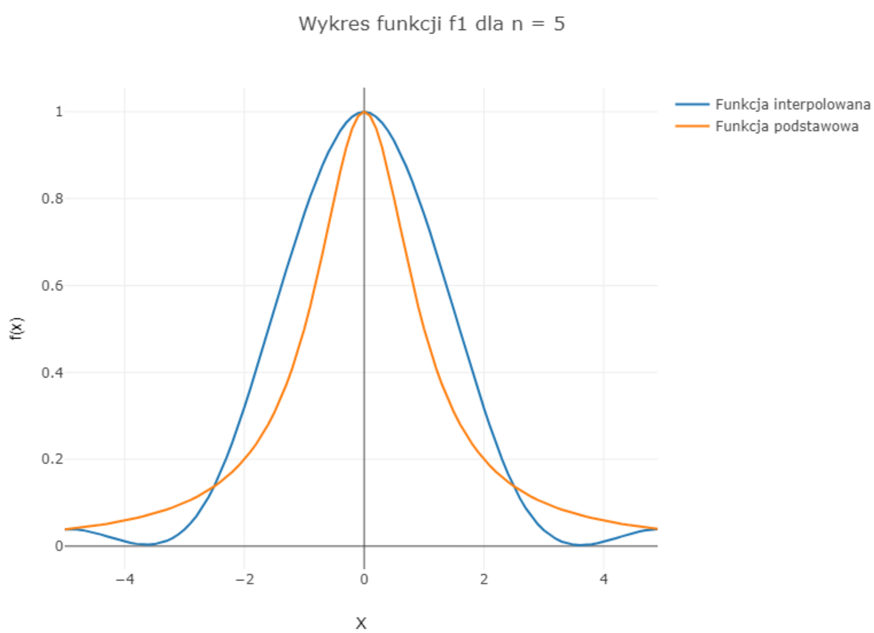
Fragment kodu 12: Wyznaczenie wartosci funkcji

5 Analiza danych

Badanie zostało przeprowadzone dla dwóch różnych funkcji, oraz różnych ilości węzłów.

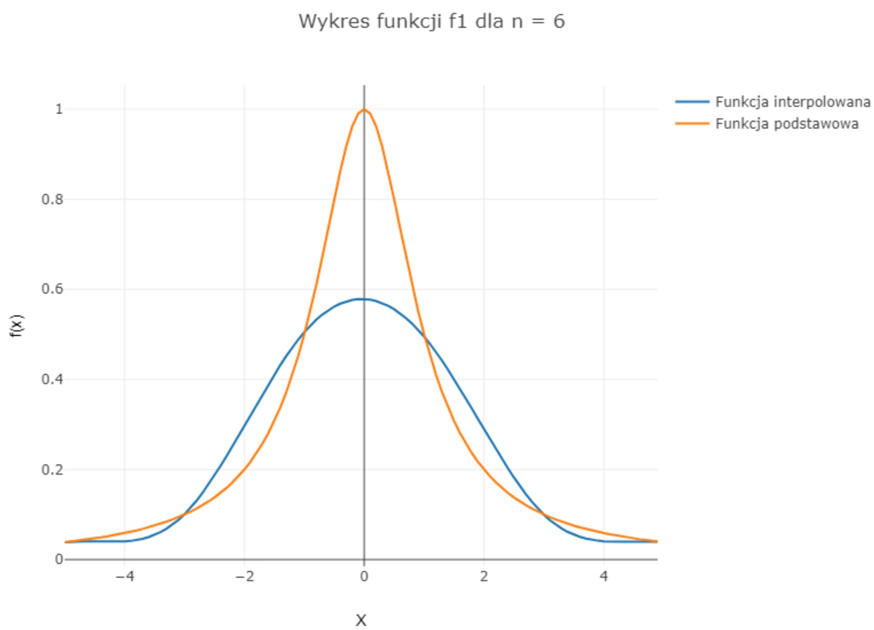
5.1 Analiza rozwiązań funkcji f1

5.1.1 Rozwiązanie dla $n = 5$



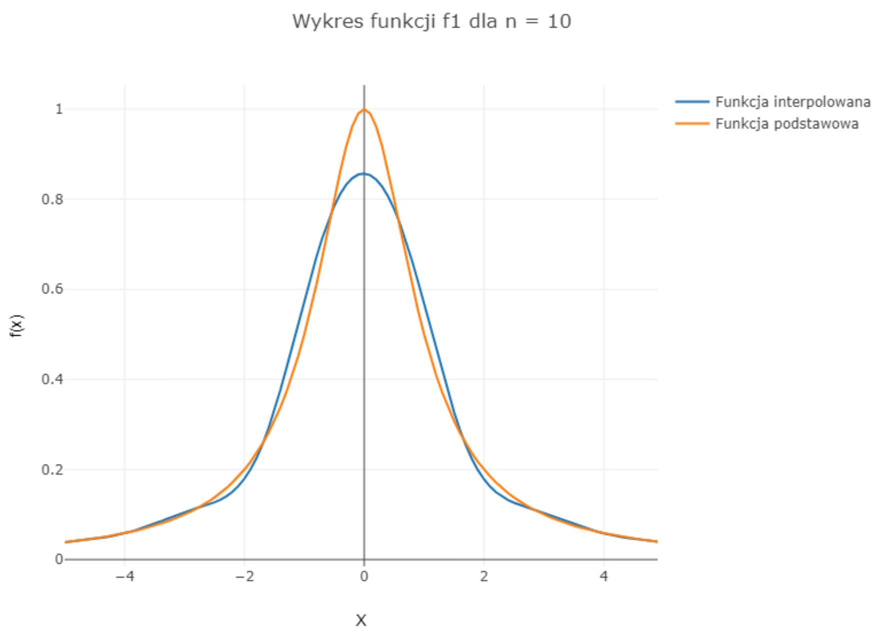
Rysunek 1: Wykres przedstawia funkcję interpolowaną dla liczby węzłów $n = 5$, oraz wykres funkcji oczekiwanej, tj. podstawowej

5.1.2 Rozwiązanie dla $n = 6$



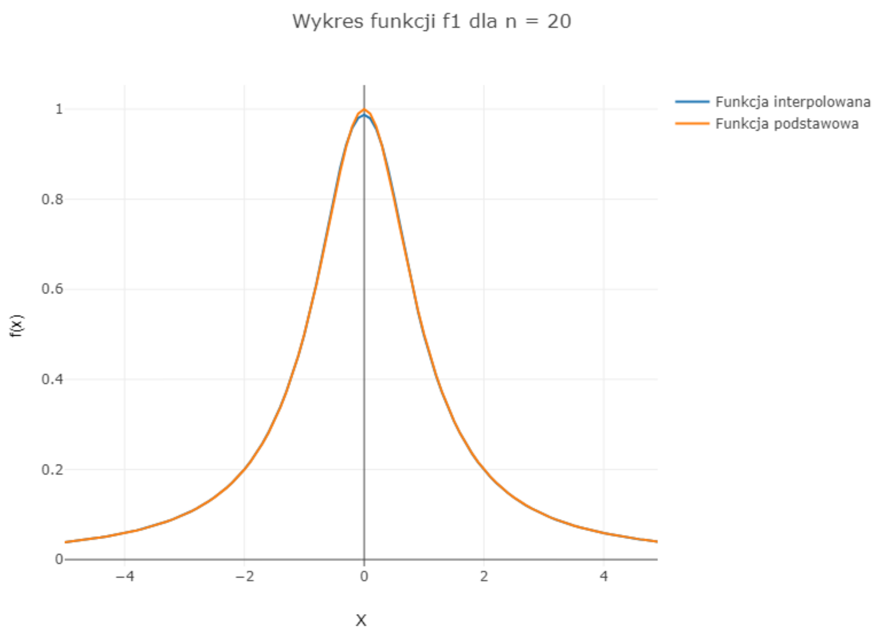
Rysunek 2: Wykres przedstawia funkcję interpolowaną dla liczby węzłów $n = 6$, oraz wykres funkcji oczekiwanej, tj. podstawowej

5.1.3 Rozwiązanie dla $n = 10$



Rysunek 3: Wykres przedstawia funkcję interpolowaną dla liczby węzłów $n = 10$, oraz wykres funkcji oczekiwanej, tj. podstawowej

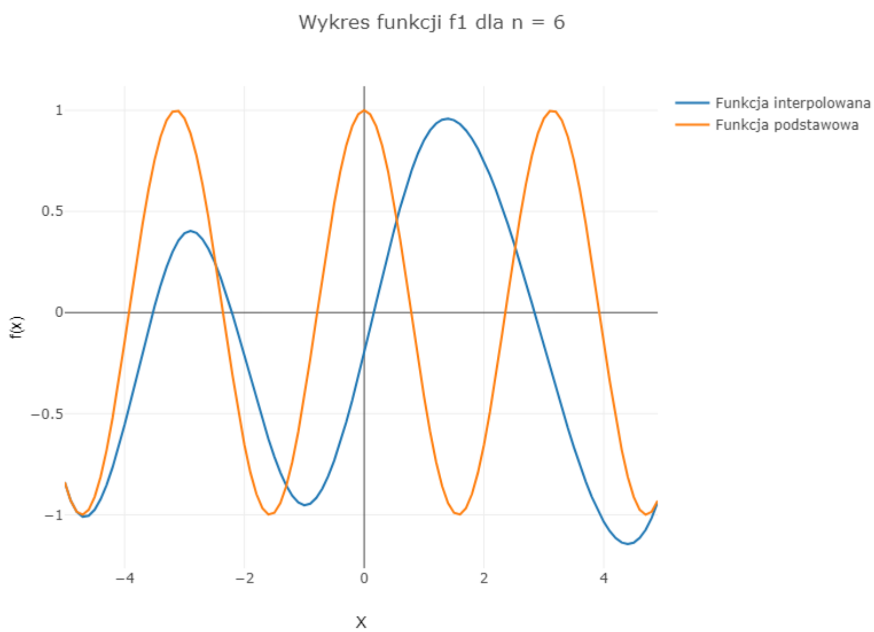
5.1.4 Rozwiązanie dla $n = 20$



Rysunek 4: Wykres przedstawia funkcję interpolowaną dla liczby węzłów $n = 20$, oraz wykres funkcji oczekiwanej, tj. podstawowej

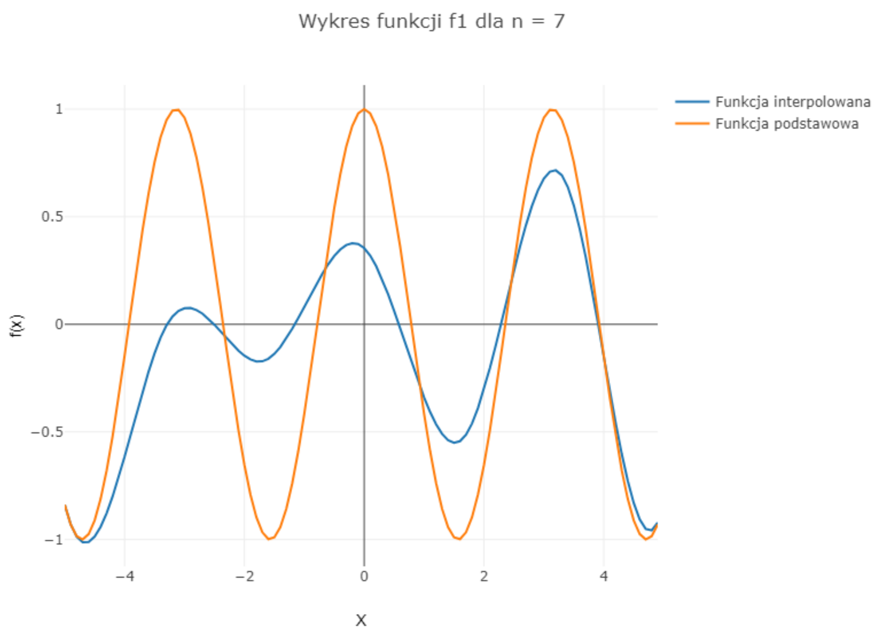
5.2 Analiza rozwiązań funkcji f2

5.2.1 Rozwiązanie dla $n = 6$



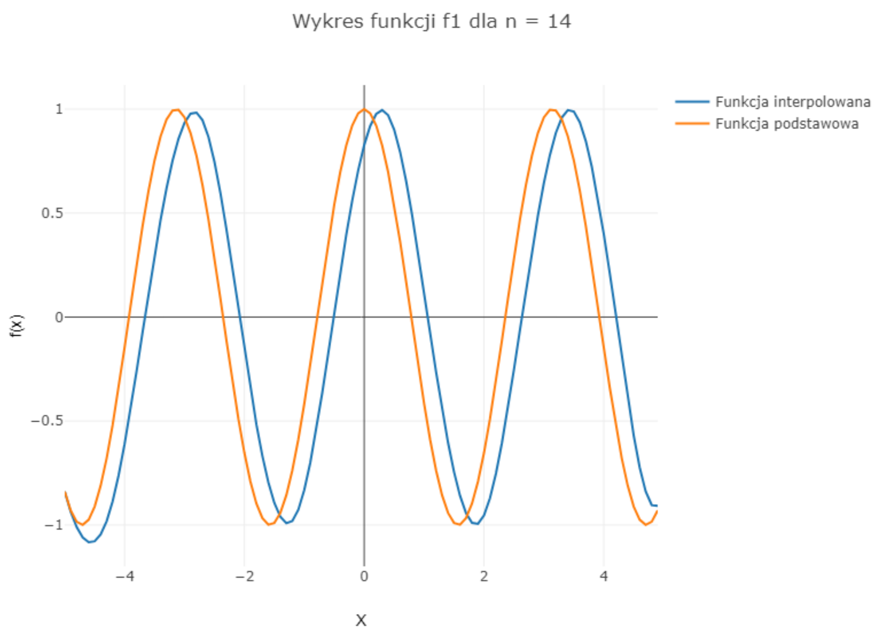
Rysunek 5: Wykres przedstawia funkcję interpolowaną f2 dla liczby węzłów $n = 6$, oraz wykres funkcji oczekiwanej, tj. podstawowej

5.2.2 Rozwiązanie dla $n = 7$



Rysunek 6: Wykres przedstawia funkcję interpolowaną f2 dla liczby węzłów $n = 7$, oraz wykres funkcji oczekiwanej, tj. podstawowej

5.2.3 Rozwiązanie dla $n = 14$



Rysunek 7: Wykres przedstawia funkcję interpolowaną f2 dla liczby węzłów $n = 14$, oraz wykres funkcji oczekiwanej, tj. podstawowej

6 Wnioski

Skuteczność Interpolacja funkcjami sklejanymi okazała się skuteczną metodą do aproksymacji danych, szczególnie w kontekście potrzeby tworzenia ciągłych i gładkich krzywych interpolacyjnych przez określone przedziały. Wszystkie przeprowadzone eksperymenty wykazały, że metoda jest w stanie skutecznie wygładzić dane wejściowe oraz zapewnić odpowiedni stopień aproksymacji funkcji.

Wpływ ilości węzłów Analiza rozwiązań dla różnych ilości węzłów (n = 5, 6, 10, 20 dla funkcji f1 oraz n = 6, 7, 14 dla funkcji f2) wykazała, że zwiększenie liczby węzłów prowadzi do wyższej dokładności interpolacji. W pierwszej funkcji zauważamy, że wraz z wzrostem ilości węzłów, funkcja w każdym kroku przybliża się do rozwiązania. Ciekawym jest to, że przy ilości węzłów n = 5, wierzchołki obu funkcji pokrywały się, lecz przy n = 6, mocno odbiegały od siebie, mimo zbliżenia całości funkcji. W drugiej funkcji przy ilości węzłów n = 6, zauważamy, że otrzymana funkcja zawiera zaledwie dwa okresy w danych przedziale, mimo że cos(2x) zawiera ich aż trzy. Zwiększenie ilości węzłów zaledwie o jeden, rozwiązało ten problem. Podwojenie węzłów spowodowało prawie całkowite pokrycie funkcji. Prowadzi to do wniosku, że ilość węzłów ma ogromne znaczenie w kontekście dokładności metody.