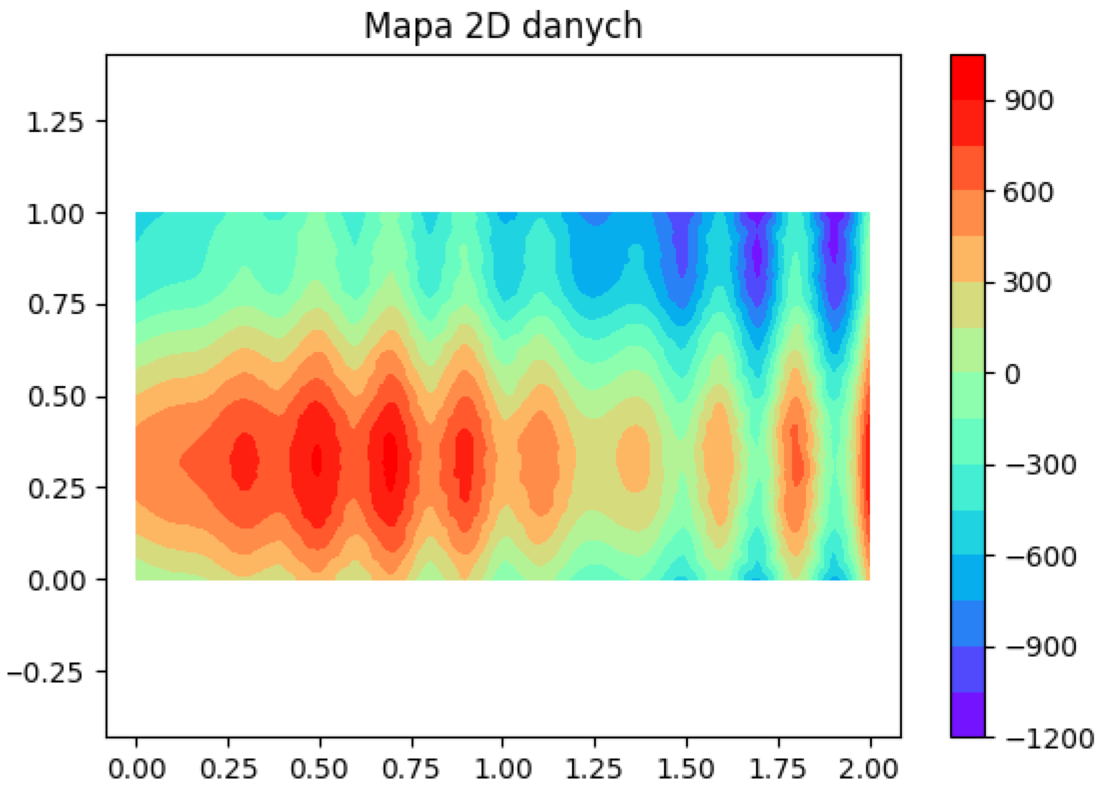
Algorytmy Numeryczne projekt Mateusz Kapituła
Numer albumu: 136559
Grupa laboratoryjna 7

Kod źródłowy projektu został początkowo opracowany w środowisku GNU Octave. Ze względu na ograniczoną znajomość tego języka programowania po zakończeniu laboratoriów, reszta implementacji została uzupełniona przy użyciu języka Python.

Wizualizacja danych w formie mapy 2D



Wykres 2D przedstawia dane z pliku 136559.dat.Przedstawia on lokalizacje punktów F(x) dzięki współrzędnym x,y. Mapa ta pozwala na zilustrowanie sobie wartość tych punktów przez zmianę koloru na cieplejszy, bądź zimniejszy. Zakresy:

1. Dla x od 0 do 2,
2. Dla y od 0 do 1,
3. Dla f(x) od -1179 do 915

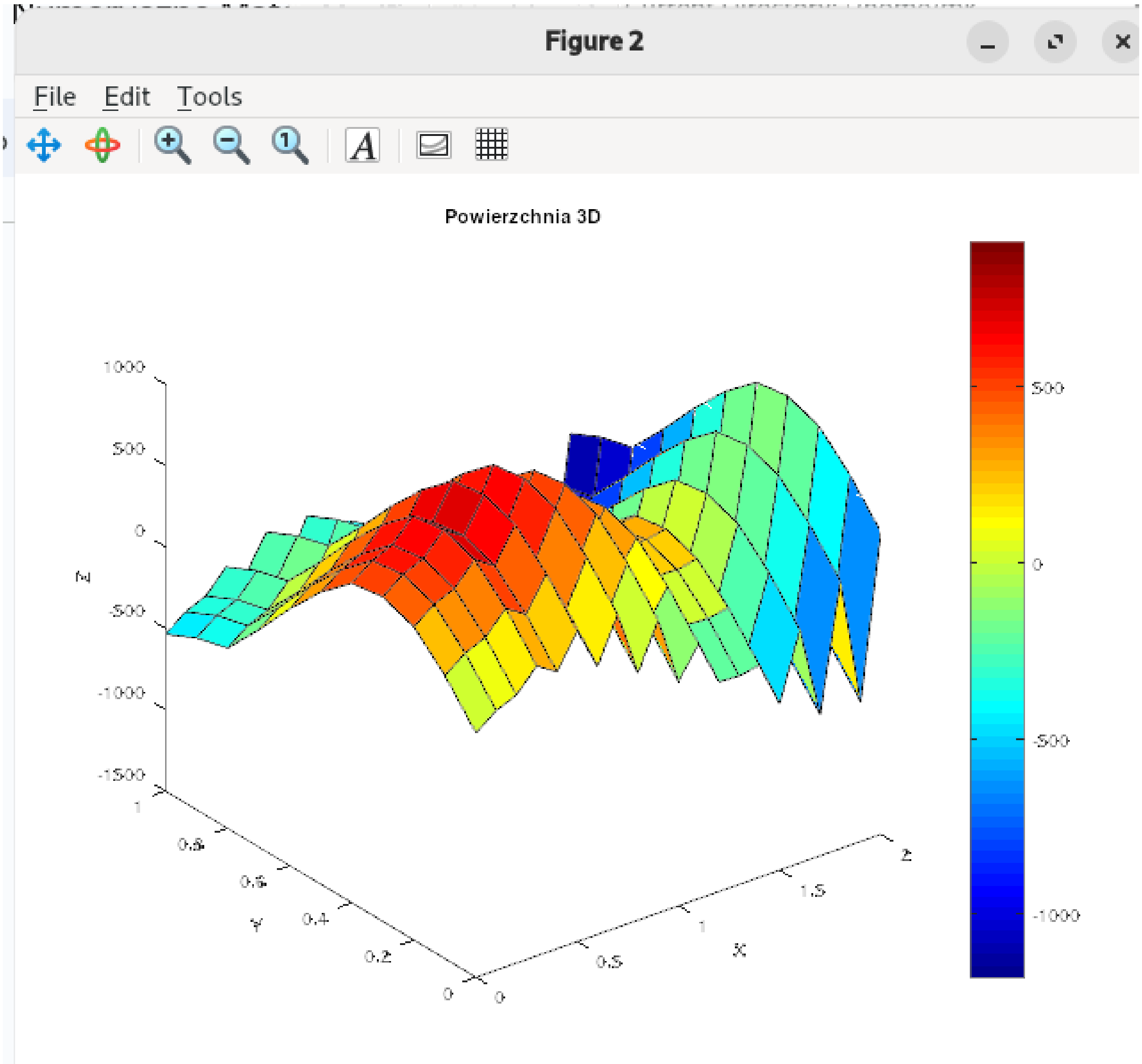
Wnioski:

- Wizualizacja danych w formie 2d pozwala na powierzchowne przeanalizowanie podanych danych.
- Mapa 2d pozwala zauważyć trendy jak na przykład zbiorowisko wysokich wartości , które są blisko siebie.
- Jest łatwa do napisania ale użyteczne informacje, które przedstawia są ograniczone.

Kod źródłowy :

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import griddata
liczby = np.loadtxt('136559.dat')
liczbyX = np.zeros(shape = liczby.shape[0])
liczbyY = np.zeros(shape = liczby.shape[0])
liczbyF = np.zeros(shape = liczby.shape[0])
for i in range(liczby.shape[0]):
    liczbyX[i] = liczby[i][0]
for i in range(liczby.shape[0]):
    liczbyY[i] = liczby[i][1]
for i in range(liczby.shape[0]):
    liczbyF[i] = liczby[i][2]
xi = np.linspace(liczbyX.min(), liczbyX.max(), 1000)
yi = np.linspace(liczbyY.min(), liczbyY.max(), 1000)
zi = griddata((liczbyX, liczbyY), liczbyF, (xi[None, :], yi[:, None]), method = 'cubic')
zmin = liczbyF.min()
zmax = liczbyF.max()
CS = plt.contourf(xi, yi, zi, 15, cmap = plt.cm.rainbow, vmax = zmax, vmin = zmin)
plt.axis('equal')
plt.title('Mapa 2D danych')
plt.colorbar()
plt.show()
```

Wizualizacja danych w formie powierzchni 3D



Kod źródłowy:

plot_map2d.m

```
data = load("136559.dat");
x = data(:, 1);
y = data(:, 2);
z = data(:, 3);

min_x = 0;
max_x = 2;
min_y = 0;
max_y = 1;
min_z = -1179;
max_z = 915;

zakres = (x >= min_x) & (x <= max_x) & (y >= min_y) & (y <= max_y) & (z >= min_z) & (z <= max_z);

x = x(zakres);
y = y(zakres);
z = z(zakres);

function plot_map3d(x, y, z)
    figure;
    [X, Y] = meshgrid(unique(x), unique(y));
    Z_grid = griddata(x, y, z, X, Y, 'linear');
    surf(X, Y, Z_grid);
    colorbar;
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    title('Powierzchnia 3D');
    colormap(jet);
end

plot_map3d(x, y, z);
```

Funkcja plot_map3d() tworzy okno graficzne i rysuje na nim powierzchnię 3D.

Zakresy:

1. Dla x od 0 do 2,
2. Dla y od 0 do 1,
3. Dla f(x) od -1179 do 915

Funkcja ta działa w następujący sposób:

- Najpierw tworzy macierze X i Y, które zawierają wszystkie możliwe wartości współrzędnych osi X i Y.
- Następnie, wykorzystując funkcję griddata(), oblicza wartości Z dla wszystkich punktów w macierzach X i Y.
- Na koniec, wykorzystując funkcję surf(), rysuje powierzchnię 3D na podstawie wyliczonych wartości Z.

Prezentowanie danych w formie 3D może być bardzo pomocne w ich zrozumieniu , ponieważ ułatwia zauważenie trendów oraz relacji pomiędzy danymi.W tym przykładzie możemy zauważyć , że ta figura ma epicentrum ciepłego koloru i im dalej tym kolor staje się zimniejszy a co za tym idzie wartości mniejsze, można również bardzo łatwo dostrzec nierówność tej powierzchni.

Wnioski:

- Prezentowanie danych w formie 3D może być bardzo pomocne w ich zrozumieniu.
- Powierzchnie 3D mogą przedstawiać różnego rodzaju dane.
- Prezentacja danych w formie 3D może ułatwić identyfikację wzorców i trendów w danych.

- Prezentacja danych w formie 3D może pomóc w zrozumieniu relacji między różnymi zmiennymi.

Średnia, mediana i odchylenie standardowe z podziałem na współrzędne y

```
data = load("136559.dat");
x = data(:, 1);
y = data(:, 2);
z = data(:, 3);

unique_y = unique(y);
wynik_mediana = zeros(length(unique_y), 1);
wynik_srednia = zeros(length(unique_y), 1);
wynik_odchylenie = zeros(length(unique_y), 1);

function sorted_array = sortowanie(array)
    n = length(array);
    for i = 1:n
        for j = i+1:n
            if array(i) > array(j)
                temp = array(i);
                array(i) = array(j);
                array(j) = temp;
            end
        end
    end
    sorted_array = array;
end


for i = 1:length(unique_y)
    idx = y == unique_y(i);
    z_subset = z(idx);

    srednia = mean(z_subset);

    sorted_z = sortowanie(z_subset);
    n = length(sorted_z);
    if mod(n, 2) == 0
        mid = n / 2;
        mediana = (sorted_z(mid) + sorted_z(mid + 1)) / 2;
    else
        mid = floor(n / 2) + 1;
        mediana = sorted_z(mid);
    end

    tmp = sum((z_subset - srednia).^2);
    odchylenie = sqrt(tmp / (length(z_subset) - 1));

    wynik_srednia(i) = srednia;
    wynik_mediana(i) = mediana;
    wynik_odchylenie(i) = odchylenie;
end

fprintf('srednie wartosci:\tmediana:\todchylenie standardowe:\n');
for i = 1:length(unique_y)
    fprintf('Dla Y %.6f: %.6f\t%.6f\t%.6f\n', unique_y(i), wynik_srednia(i), wynik_mediana(i), wynik_odchylenie(i));
end
```

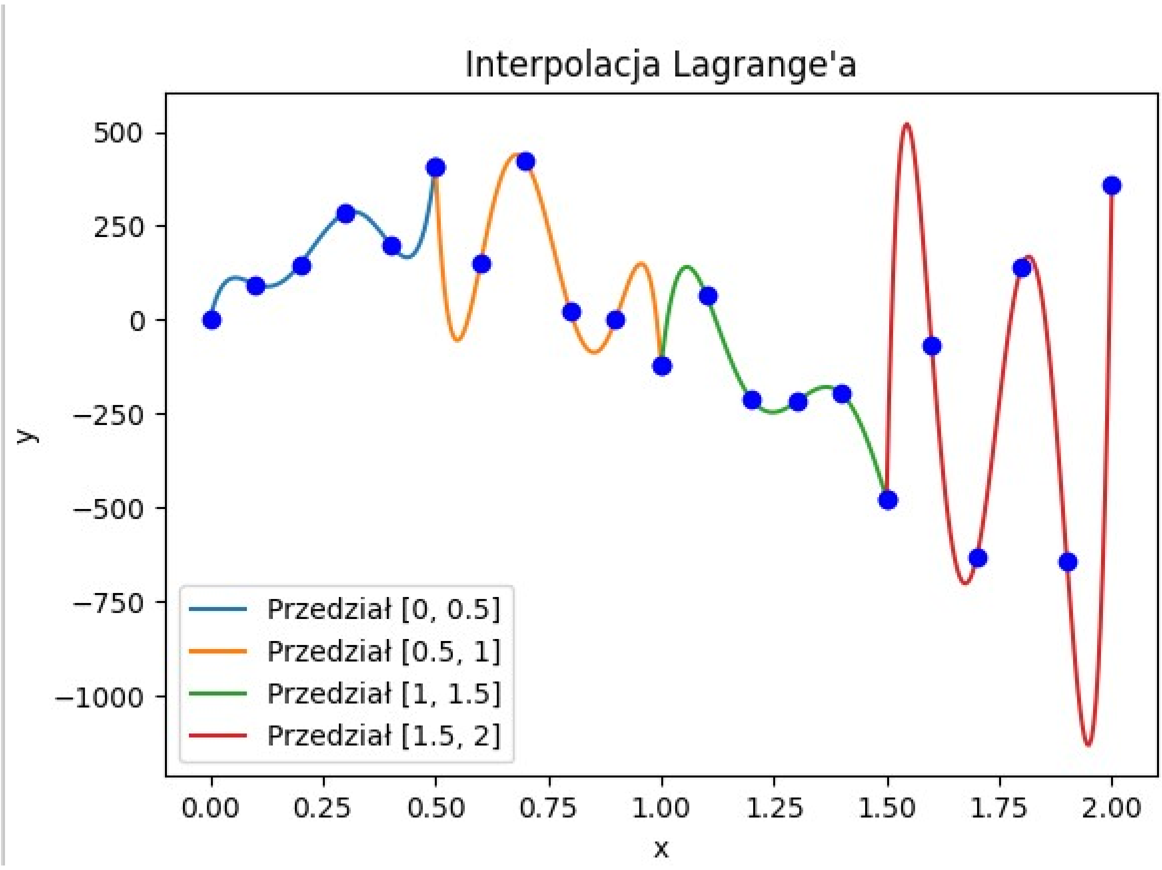
Kod ten oblicza statystyki opisowe dla danych z pliku 136559.dat.

```
srednie wartosci:       mediana:        odchylenie standardowe:
Dla Y 0.000000: 1.174867        65.550200       312.116145
Dla Y 0.100000: 240.168024      304.543000      312.116202
Dla Y 0.200000: 425.165667      489.541000      312.116086
Dla Y 0.300000: 494.814514      559.190000      312.116112
Dla Y 0.400000: 464.373410      528.749000      312.116206
Dla Y 0.500000: 294.317519      358.693000      312.116099
Dla Y 0.600000: 77.288203       141.664000      312.116161
Dla Y 0.700000: -166.755019     -102.380000     312.116212
Dla Y 0.800000: -393.030505     -328.655000     312.116217
Dla Y 0.900000: -444.941471     -380.566000     312.115973
Dla Y 1.000000: -536.204143     -471.829000     312.116420
>> c
```

Wnioski:

- Średnie wartości dla różnych wartości Y wahały się od ujemnych do dodatnich liczb. Oznacza to, że wartościZ nie wykazują jednolitego trendu wzrostowego ani spadkowego wraz z rosnącym Y.
- Odchylenie standardowe jest bardzo podobne dla wszystkich obliczonych przypadków. Oznacza to, że dane są rozproszone wokół średnich wartości w zbliżonym stopniu.
- Mediana jest bardziej stabilna niż średnia. Różnice między medianami dla różnychY sugerują zmienność rozkładu danych.

Funkcja interpolacyjna metodą Lagrange'a



Dla przyspieszenia obliczeń , zwiększenia czytelności i uniknięcia ogromnych skoków na wykresie podzieliłem funkcje na 4 przedziały , które są wyróżnione kolorami.

Wnioski:

- Z moich obserwacji wraz ze zwiększaniem się x wykres staje się coraz bardziej agresywny w skokach między punktami.
- Podział funkcji na przedziały jest bardzo użyteczny przy dużej ilości danych

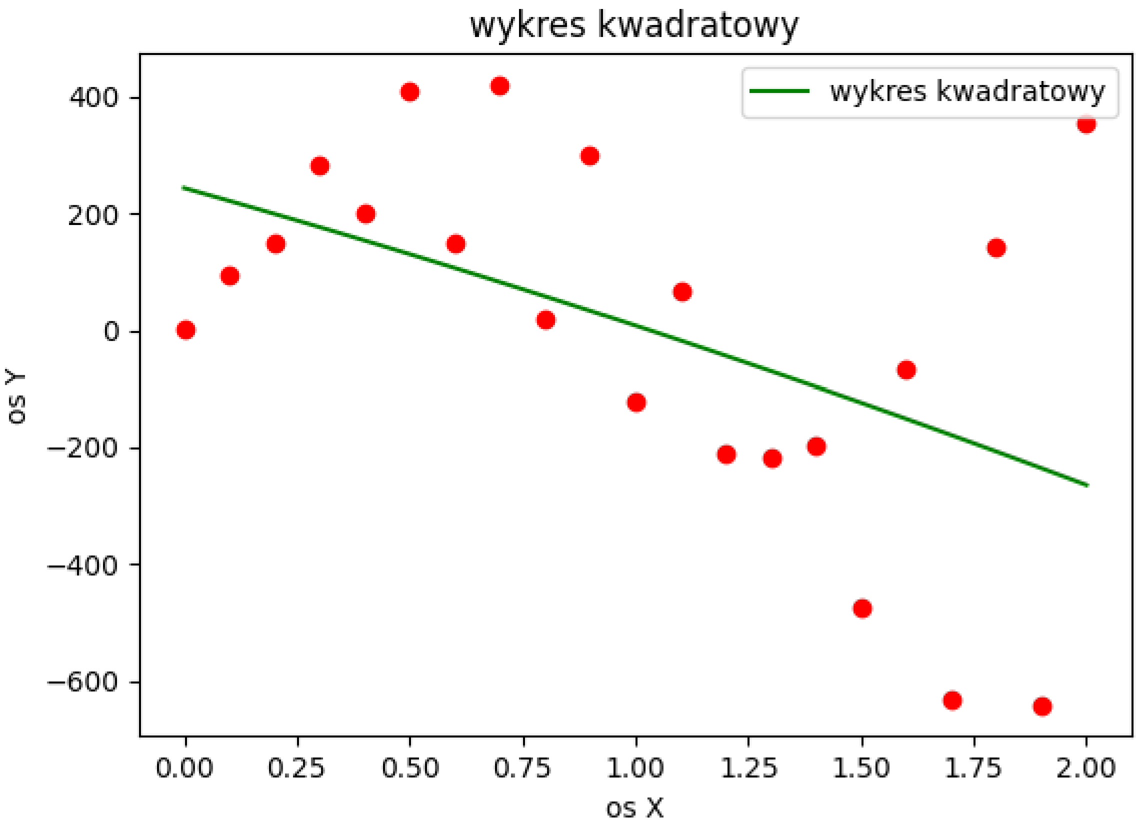
Kod źródłowy:

```python
import numpy as np
import matplotlib.pyplot as plt

def wczytaj_dane():
    liczby = np.loadtxt('136559.dat')
    liczbyX = np.zeros(shape=21)
    liczbyY = np.zeros(shape=21)

    for i in range(21):
        liczbyX[i] = liczby[i][0]

    for i in range(21):
        liczbyY[i] = liczby[i][2]

    return liczbyX, liczbyY

def mianownik(i, x):
    wynik = 1
    for j in range(x.shape[0]):
        if i != j:
            wynik *= x[i] - x[j]
    return wynik

def interp_lagA(x, y):
    n = x.shape[0]
    a = np.zeros(n)

    for i in range(n):
        a[i] = y[i] / mianownik(i, x)

    return a

def funkcja(a, x, xi):
    wynik = 0
    n = x.shape[0]

    for i in range(x.shape[0]):
        mian = 1
        for j in range(x.shape[0]):
            if i != j:
                mian *= (xi - x[j])
        wynik += a[i] * mian

    return wynik
```

```python
def rysuj_lag(xp, xk, x, y, label):
    A = interp_lagA(x, y)
    lP = 100
    xt = np.linspace(xp, xk, lP)
    yt = np.zeros(xt.shape[0])

    for i in range(xt.shape[0]):
        yt[i] = funkcja(A, x, xt[i])

    plt.plot(xt, yt, label=label)
    plt.plot(x, y, 'bo')

x, y = wczytaj_dane()
x1 = np.array([x[0], x[1], x[2], x[3], x[4], x[5]])
y1 = np.array([y[0], y[1], y[2], y[3], y[4], y[5]])
x2 = np.array([x[5], x[6], x[7], x[8], x[9], x[10]])
y2 = np.array([y[5], y[6], y[7], y[8], x[9], y[10]])
x3 = np.array([x[10], x[11], x[12], x[13], x[14], x[15]])
y3 = np.array([y[10], y[11], y[12], y[13], y[14], y[15]])
x4 = np.array([x[15], x[16], x[17], x[18], x[19], x[20]])
y4 = np.array([y[15], y[16], y[17], y[18], y[19], y[20]])

rysuj_lag(0, 0.5, x1, y1, 'Przedział [0, 0.5]')
rysuj_lag(0.5, 1, x2, y2, 'Przedział [0.5, 1]')
rysuj_lag(1, 1.5, x3, y3, 'Przedział [1, 1.5]')
rysuj_lag(1.5, 2, x4, y4, 'Przedział [1.5, 2]')

plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Interpolacja Lagrange\'a')
plt.show()
```

Funkcje Aproksymacyjne

Aproksymacja kwadratowa

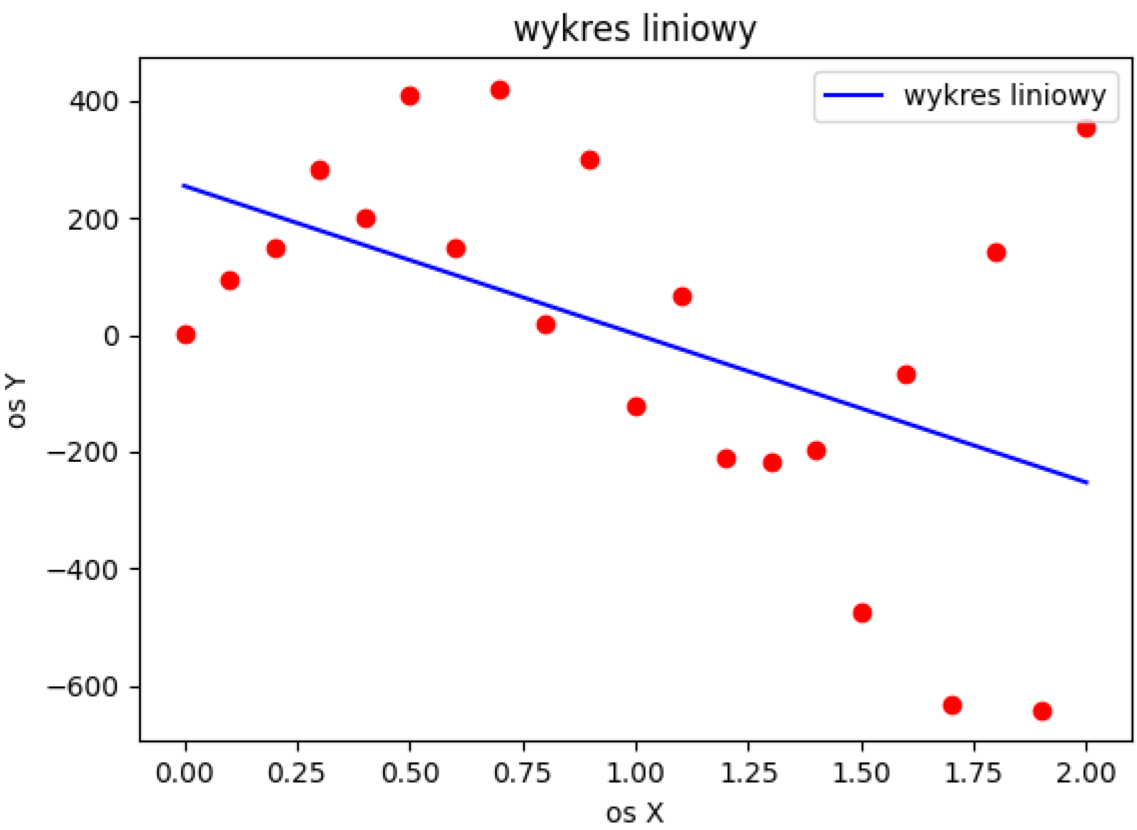


Aproksymacja polega na przedstawieniu funkcji na wykresie w celu przybliżenia punktów danych. Na zamieszczonym wykresie można zauważyć, że różnica między rzeczywistymi punktami a wynikami aproksymacji kwadratowej jest dość istotna. Wykres wygładza te różnice do takiego stopnia, że punkty wydają się tworzyć jedną linię.

Wnioski:

- **Aproksymacja kwadratowa jest przydatna w przypadkach, gdy chcemy znaleźć prostą funkcję matematyczną, która najlepiej odwzorowuje nasze dane.**
- **Możemy użyć aproksymacji kwadratowej do optymalizacji funkcji, aby znaleźć maksimum lub minimum danej funkcji.**
- **Aproksymacja kwadratowa może pomóc w identyfikacji ogólnego kształtu i trendów danych**

Aproksymacja liniowa



Analizując przedstawiony wykres, zauważamy, że aproksymacja liniowa skutkuje większymi różnicami między rzeczywistymi punktami a wynikami aproksymacji w porównaniu do aproksymacji kwadratowej. Linia aproksymacyjna wydaje się gorzej odwzorowywać ogólny kształt rozkładu danych zwłaszcza na końcu wykresu.

Wnioski:

- **Aproksymacja liniowa jest kolejną metodą odwzorowania danych.**
- **Wybór stopnia aproksymacji ma istotny wpływ na dokładność odwzorowania danych.**
- **W niektórych przypadkach, szczególnie gdy dane mają bardziej złożony kształt, wyższy stopień wielomianu może prowadzić do bardziej precyzyjnych wyników.**

Kod źródłowy:

```python
import numpy as np
import matplotlib.pyplot as plt

dane = np.loadtxt('136559.dat')
wartosci_x = dane[:21, 0]
wartosci_y = dane[:21, 2]

macierz_liniowa = np.array([[len(wartosci_x), np.sum(wartosci_x)],
                            [np.sum(wartosci_x), np.sum(wartosci_x ** 2)]])
wektor_liniowy = np.array([np.sum(wartosci_y), np.sum(wartosci_x * wartosci_y)])

macierz_kwadratowa = np.array([[len(wartosci_x), np.sum(wartosci_x), np.sum(wartosci_x ** 2)],
                               [np.sum(wartosci_x), np.sum(wartosci_x ** 2), np.sum(wartosci_x ** 3)],
                               [np.sum(wartosci_x ** 2), np.sum(wartosci_x ** 3), np.sum(wartosci_x ** 4)]])
wektor_kwadratowy = np.array([np.sum(wartosci_y), np.sum(wartosci_x * wartosci_y), np.sum(wartosci_x ** 2 * wartosci_y)])

wspolczynniki_liniowe = np.linalg.solve(macierz_liniowa, wektor_liniowy)
wspolczynniki_kwadratowe = np.linalg.solve(macierz_kwadratowa, wektor_kwadratowy)

def funkcja_liniowa(x, a1, a0):
    return a1 * x + a0

def funkcja_kwadratowa(x, a2, a1, a0):
    return a2 * x ** 2 + a1 * x + a0

x_liniowy = np.linspace(0, 2, 25)
plt.plot(x_liniowy, funkcja_liniowa(x_liniowy, wspolczynniki_liniowe[1], wspolczynniki_liniowe[0]), 'b-')
plt.plot(wartosci_x, wartosci_y, 'ro')
plt.xlabel('Oś X')
plt.ylabel('Oś Y')
plt.legend(['Wykres liniowy'])
plt.title('Wykres liniowy')
plt.show()

x_kwadratowy = np.linspace(0, 2, 25)
plt.plot(x_kwadratowy, funkcja_kwadratowa(x_kwadratowy, wspolczynniki_kwadratowe[2], wspolczynniki_kwadratowe[1], wspolczynniki_kwadratowe[0]), 'g-')
plt.plot(wartosci_x, wartosci_y, 'ro')
plt.xlabel('Oś X')
plt.ylabel('Oś Y')
plt.legend(['Wykres kwadratowy'])
plt.title('Wykres kwadratowy')
plt.show()
```

Pole powierzchni funkcji

```
Pole powierzchni funkcji : 7701.475294238946
```

Kod źródłowy:

```
function pole = liczPole(x1, x2, x3, y1, y2, y3, z1, z2, z3)
    a = [x(x1), y(y1), z(z1)];
    b = [x(x2), y(y2), z(z2)];
    c = [x(x3), y(y3), z(z3)];
    ab = b - a;
    ac = c - a;
    wyz1 = ab(2) * ac(3) - ab(3) * ac(2);
    wyz2 = -(ab(1) * ac(3) - ab(3) * ac(1));
    wyz3 = ab(1) * ac(2) - ab(2) * ac(1);
    abXac = [wyz1, wyz2, wyz3];
    pole = 0.5 * sqrt(wyz1^2 + wyz2^2 + wyz3^2);
end

data = load("136559.dat");
x = data(:, 1);
y = data(:, 2);
z = data(:, 3);

suma = 0;
for z = 1:20
    for i = z:21:210
        suma += liczPole(i, i + 1, i + 21, i, i + 1, i + 21, i, i + 1, i + 21);
        suma += liczPole(i + 1, i + 1, i + 21, i + 21, i + 1, i + 21, i + 22, i + 1, i + 21);
    end
end

fprintf('Pole powierzchni funkcji: %.4f\n', suma);
```

Pole powierzchni zostało obliczone na podstawie pola trójkąta. Każdy wierzchołek trójkąta jest opisany trzema współrzędnymi, które tworzą wektor. Pole powierzchni trójkąta obliczono na podstawie tych współrzędnych. Następnie, aby obliczyć pole powierzchni całej figury, podzielono ją na trójkąty i zsumowano pola powierzchni poszczególnych trójkątów.

Wnioski:

- Kod można zoptymalizować pod kątem wydajności.
- Kod można rozszerzyć, aby umożliwiał obliczanie pola powierzchni innych figur. Można to zrobić na przykład, dodając obsługę innych typów figur, takich jak prostokąty, okręgi lub koła.

Całka z funkcji interpolacyjnej i aproksymacyjnych

```
interpolacja : -56.08302857863909
aproksymacja funkcja liniowa: 2.349733333333287
aproksymacja kwadratowa: 3.5352165157898305
```

Wnioski:

- **Można zauważyć , że oba wyniki aproksymacji są dodatnie i do siebie Podobne. Wynik aproksymacji kwadratowej jest większy i bardziej dokładny od liniowej , ponieważ moje dane zawierają nieliniowe elementy i zakrzywienia.**
- **Wynik interpolacji na minusie oznacza , że otrzymano wartości poniżej zera w obszarze, gdzie interpolacja ma miejsce.**

Kod źródłowy:

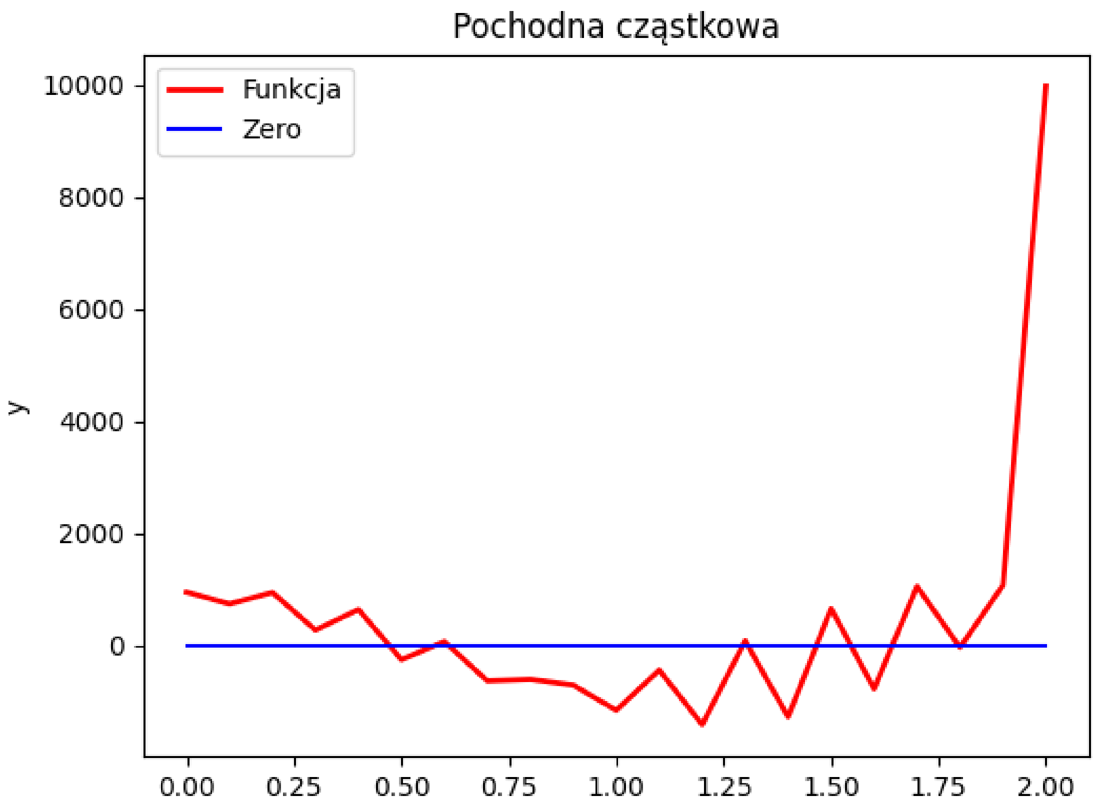
```
data = load("136559.dat");
x = data(:, 1);
y = data(:, 2);
z = data(:, 3);

function suma = poleLagrange(x, y)
    suma = 0;
    for i = 1:(length(x) - 1)
        wysokosc = abs(x(i) - x(i + 1));
        podstawaA = y(i);
        podstawaB = y(i + 1);
        pole = 0.5 * (podstawaA + podstawaB) * wysokosc;
        suma += pole;
    end
end

function m = mianownik(i, x)
    m = 1;
    for j = 1:length(x)
        if i != j
            m *= x(i) - x(j);
        end
    end
end

function a = interp_lagA(x, y)
    n = length(x);
    a = zeros(1, n);
    for i = 1:n
        a(i) = y(i) / mianownik(i, x);
    end
end

function w = funkcja(a, x, xi)
    w = 0;
    n = length(x);
    for i = 1:length(x)
        m = 1;
        for j = 1:length(x)
            if i != j
                m *= (xi - x(j));
            end
        end
```

```
function [xt, yt] = rysuj_lag(xp, xk, x, y)
    A = interp_lagA(x, y);
    lP = 100;
    xt = linspace(xp, xk, lP);
    yt = zeros(1, length(xt));
    for i = 1:length(xt)
        yt(i) = funkcja(A, x, xt(i));
    end
end

x1 = [x(1), x(2), x(3), x(4), x(5), x(6)];
y1 = [y(1), y(2), y(3), y(4), y(5), y(6)];
x2 = [x(6), x(7), x(8), x(9), x(10), x(11)];
y2 = [y(6), y(7), y(8), y(9), x(10), y(11)];
x3 = [x(11), x(12), x(13), x(14), x(15), x(16)];
y3 = [y(11), y(12), y(13), y(14), y(15), y(16)];
x4 = [x(16), x(17), x(18), x(19), x(20), x(21)];
y4 = [y(16), y(17), y(18), y(19), y(20), y(21)];

[xt1, yt1] = rysuj_lag(0, 0.5, x1, y1);
[xt2, yt2] = rysuj_lag(0.5, 1, x2, y2);
[xt3, yt3] = rysuj_lag(1, 1.5, x3, y3);
[xt4, yt4] = rysuj_lag(1.5, 2, x4, y4);

pole1 = poleLagrange(xt1, yt1);
pole2 = poleLagrange(xt2, yt2);
pole3 = poleLagrange(xt3, yt3);
pole4 = poleLagrange(xt4, yt4);
pole = pole1 + pole2 + pole3 + pole4;

printf('interpolacja: %f\n', pole);

M = zeros(2, 2);
P = zeros(1, 2);
W = zeros(1, 2);
A = zeros(1, 2);

M2 = zeros(3, 3);
P2 = zeros(1, 3);
W2 = zeros(1, 3);
A2 = zeros(1, 3);
```

```
for i = 1:length(x)
    M2(1, 1) = length(x);
    M2(1, 2) += x(i);
    M2(1, 3) += x(i) * x(i);
    M2(2, 1) += x(i);
    M2(2, 2) += x(i) * x(i);
    M2(2, 3) += x(i) * x(i) * x(i);
    M2(3, 1) += x(i) * x(i);
    M2(3, 2) += x(i) * x(i) * x(i);
    M2(3, 3) += x(i) * x(i) * x(i) * x(i);
end

for i = 1:length(y)
    W2(1) += y(i);
    W2(2) += x(i) * y(i);
    W2(3) += x(i) * x(i) * y(i);
end

A2 = M2 \ W2;

for i = 1:length(x)
    M(1, 1) = length(x);
    M(1, 2) += x(i);
    M(2, 1) += x(i);
    M(2, 2) += x(i) * x(i);
end

for i = 1:length(y)
    W(1) += y(i);
    W(2) += x(i) * y(i);
end

A = M \ W;

function result = f(x, a1, a0)
    result = a1 * x + a0;
end

function result = f2(x, a2, a1, a0)
    result = a2 * x * x + a1 * x + a0;
end
```

```
xN = linspace(0, 2, 25);
poleLiniowa = poleLagrange(xN, f(xN, A(2), A(1)));
poleKwadratowa = poleLagrange(xN, f2(xN, A2(3), A2(2), A2(1)));

printf('aproksymacja funkcja liniowa: ', poleLiniowa);
printf('aproksymacja kwadratowa: ', poleKwadratowa);
```

Pochodne cząstkowe



W wyniku zastosowania metody polegającej na operacjach na sąsiadach punktu, udało się obliczyć pochodne cząstkowe.

Wnioski:

- Te pochodne pozwalają na analizę zachodzących zmian w funkcji w reakcji na niewielkie modyfikacje jednego z jej argumentów. Dzięki temu jesteśmy w stanie określić, jak szybko i w jakim kierunku funkcja zmienia się w konkretnym punkcie.

Kod źródłowy:

```
data = load("136559.dat");
x = data(:, 1);
y = data(:, 2);
z = data(:, 3);

wynik = zeros(size(x));
wynik(1) = (z(2) - z(1)) / (x(2) - x(1));
wynik(end) = (z(end) - z(end-1)) / (x(end) - x(end-1));

for i = 2:(length(x)-1)
    wynik(i) = (z(i + 1) - z(i - 1)) / (x(i + 1) - x(i - 1));
end

disp(wynik);

plot(x, wynik, 'r-', 'linewidth', 2);
hold on;
plot(x, zeros(size(x)), 'b-');
xlabel('x');
ylabel('y');
title('Pochodna cząstkowa');
legend('Funkcja', 'Zero');
grid on;
hold off;
```

Wyniki:

Command Window

```
   942.806
   735.155
   936.512
   265.775
   633.955
  -261.825
    59.210
  -640.093
  -611.925
  -715.627
 -1166.004
  -447.190
 -1421.291
    74.105
 -1285.275
   651.160
  -782.690
  1047.040
   -46.230
  1069.640
  -463.926
    11.765
   735.155
   936.510
   265.775
   633.955
  -261.825
    59.210
  -640.095
  -611.920
  -715.625
 -1166.010
  -447.188
 -1421.287
    74.105
 -1285.279
   651.158
  -782.690
  1047.040
   -46.230
  1069.645
  -435.508
    40.185
   735.155
   936.510
   265.775
   633.960
  -261.825
```

Command Window

```
  -261.825
    59.210
  -640.095
  -611.925
  -715.630
 -1166.005
  -447.185
 -1421.290
    74.105
 -1285.275
   651.160
  -782.690
  1047.035
   -46.230
  1069.645
  -374.798
   100.894
   735.155
   936.510
   265.775
   633.960
  -261.830
    59.205
  -640.090
  -611.920
  -715.630
 -1166.005
  -447.185
 -1421.290
    74.105
 -1285.276
   651.160
  -782.694
  1047.035
   -46.225
  1069.645
  -322.119
   153.573
   735.150
   936.510
   265.780
   633.960
  -261.830
    59.205
  -640.090
  -611.920
  -715.630
 -1166.005
  -447.185
```

Command Window

```
-1285.277
651.160
-782.693
1047.035
-46.230
1069.645
-248.638
227.055
735.155
936.510
265.775
633.960
-261.830
59.205
-640.090
-611.920
-715.630
-1166.005
-447.186
-1421.290
74.105
-1285.276
651.160
-782.695
1047.035
-46.225
1069.645
-223.914
251.777
735.153
936.510
265.775
633.955
-261.825
59.210
-640.092
-611.920
-715.629
-1166.005
-447.185
-1421.290
74.105
-1285.280
651.157
-782.690
1047.038
-46.230
1069.645
-209.697
```

Command Window

```
-1285.277
651.160
-782.693
1047.035
-46.230
1069.645
-248.638
227.055
735.155
936.510
265.775
633.960
-261.830
59.205
-640.090
-611.920
-715.630
-1166.005
-447.186
-1421.290
74.105
-1285.276
651.160
-782.695
1047.035
-46.225
1069.645
-223.914
251.777
735.153
936.510
265.775
633.955
-261.825
59.210
-640.092
-611.920
-715.629
-1166.005
-447.185
-1421.290
74.105
-1285.280
651.157
-782.690
1047.038
-46.230
1069.645
-209.697
```

Command Window

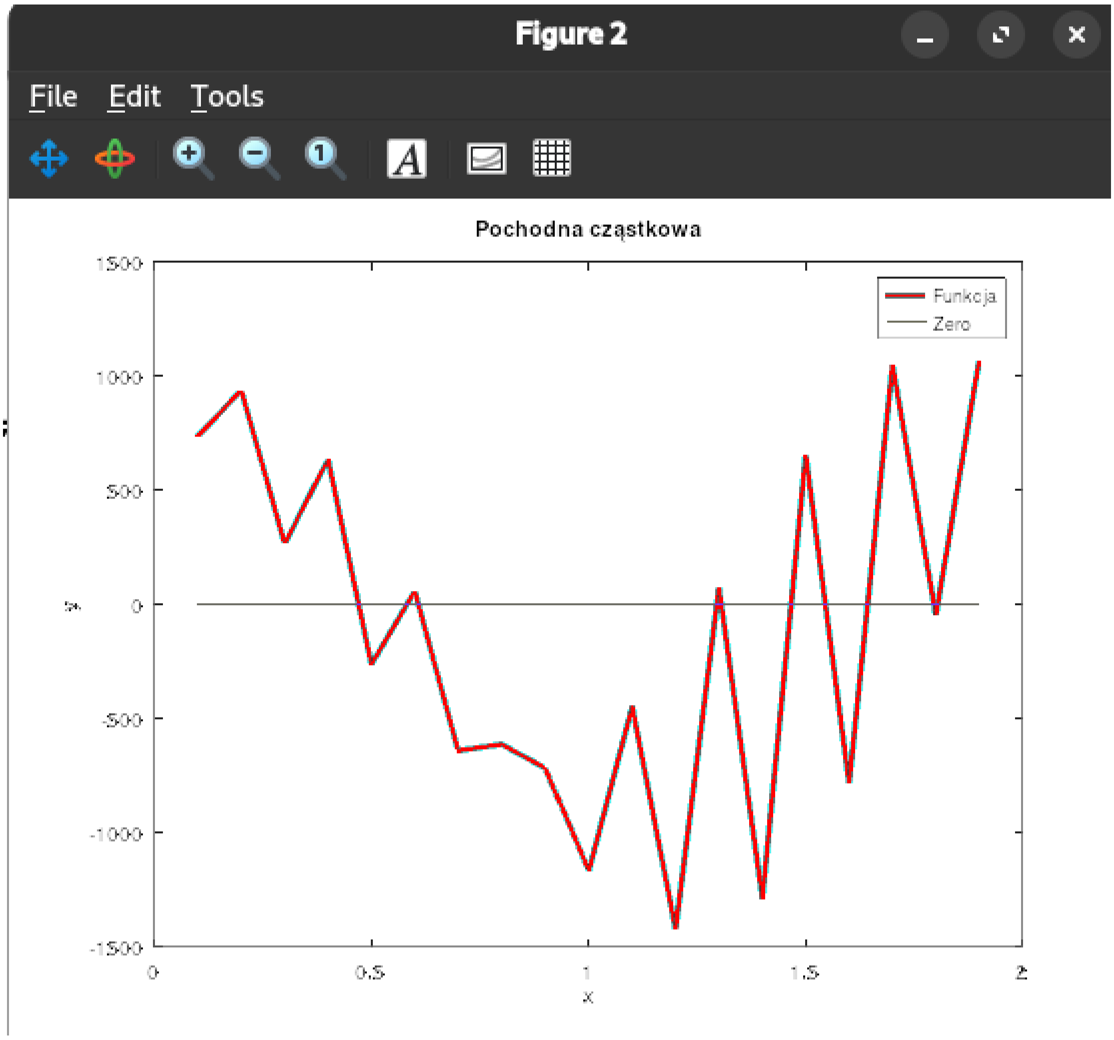
```
-1285.277
  651.160
 -782.693
 1047.035
  -46.230
 1069.645
 -248.638
  227.055
  735.155
  936.510
  265.775
  633.960
 -261.830
   59.205
 -640.090
 -611.920
 -715.630
-1166.005
 -447.186
-1421.290
   74.105
-1285.276
  651.160
 -782.695
 1047.035
  -46.225
 1069.645
 -223.914
  251.777
  735.153
  936.510
  265.775
  633.955
 -261.825
   59.210
 -640.092
 -611.920
 -715.629
-1166.005
 -447.185
-1421.290
   74.105
-1285.280
  651.157
 -782.690
 1047.038
  -46.230
 1069.645
 -209.697
```

Command Window

```
 -209.697
  265.995
  735.155
  936.511
  265.776
  633.955
 -261.827
   59.210
 -640.094
 -611.920
 -715.625
-1166.010
 -447.190
-1421.285
   74.105
-1285.280
  651.160
 -782.690
 1047.039
  -46.230
 1069.647
 -219.048
  256.644
  735.155
  936.510
  265.775
  633.958
 -261.830
   59.209
 -640.090
 -611.921
 -715.630
-1166.006
 -447.185
-1421.290
   74.105
-1285.275
  651.160
 -782.710
 1047.035
  -46.200
 1069.646
 -310.818
  164.873
  735.150
  936.510
  265.780
  633.958
 -261.830
```

-261.830
59.208
-640.090
-611.921
-715.630
1166.005
-447.185
1421.290
74.105
1285.275
651.160
-782.705
1047.035
-46.200
1069.646
-290.106
185.584
735.155
936.510
265.775
633.955
-261.825
59.210
-640.095
-611.920
-715.625
1166.010
-447.190
1421.285
74.105
1285.270
651.160
-782.700
1047.040
-46.250
1069.640
9980.990

Monotoniczność



Wnioski:

Z tego wykresu jak i również wyniku z konsoli podanego niżej widać że funkcja bardzo często zmienia swoją monotoniczność.

Kod źródłowy:

```
data = load("136559.dat");
datax = data(:, 1);
datay = data(:, 3);

wynik = zeros(21, 1);
wynik(1) = (datay(2) - datay(1)) / (datax(2) - datax(1));
wynik(21) = (datay(21) - datay(20)) / (datax(21) - datax(20));
for i = 2:20
    wynik(i) = (datay(i + 1) - datay(i - 1)) / (datax(i + 1) - datax(i - 1));
end

for i = 1:21
    if wynik(i) > 0
        disp(["Funkcja w okolicy punktu X" num2str(datax(i)) " jest rosnąca"]);
    else
        disp(["Funkcja w okolicy punktu X" num2str(datax(i)) " jest malejąca"]);
    end
end
figure;
plot(datax(2:20), wynik(2:20), 'r-', 'linewidth', 2);
hold on;
plot(datax(2:20), zeros(19, 1), 'b-');
xlabel('x');
ylabel('y');
title('Pochodna cząstkowa');
legend('Funkcja', 'Zero');
grid on;
hold off;
```

Wyniki z konsoli:

```
Funkcja w okolicy punktu X0 jest rosnąca
Funkcja w okolicy punktu X0.1 jest rosnąca
Funkcja w okolicy punktu X0.2 jest rosnąca
Funkcja w okolicy punktu X0.3 jest rosnąca
Funkcja w okolicy punktu X0.4 jest rosnąca
Funkcja w okolicy punktu X0.5 jest malejąca
Funkcja w okolicy punktu X0.6 jest rosnąca
Funkcja w okolicy punktu X0.7 jest malejąca
Funkcja w okolicy punktu X0.8 jest malejąca
Funkcja w okolicy punktu X0.9 jest malejąca
Funkcja w okolicy punktu X1 jest malejąca
Funkcja w okolicy punktu X1.1 jest malejąca
Funkcja w okolicy punktu X1.2 jest malejąca
Funkcja w okolicy punktu X1.3 jest rosnąca
Funkcja w okolicy punktu X1.4 jest malejąca
Funkcja w okolicy punktu X1.5 jest rosnąca
Funkcja w okolicy punktu X1.6 jest malejąca
Funkcja w okolicy punktu X1.7 jest rosnąca
Funkcja w okolicy punktu X1.8 jest malejąca
Funkcja w okolicy punktu X1.9 jest rosnąca
Funkcja w okolicy punktu X2 jest rosnąca
>>
```