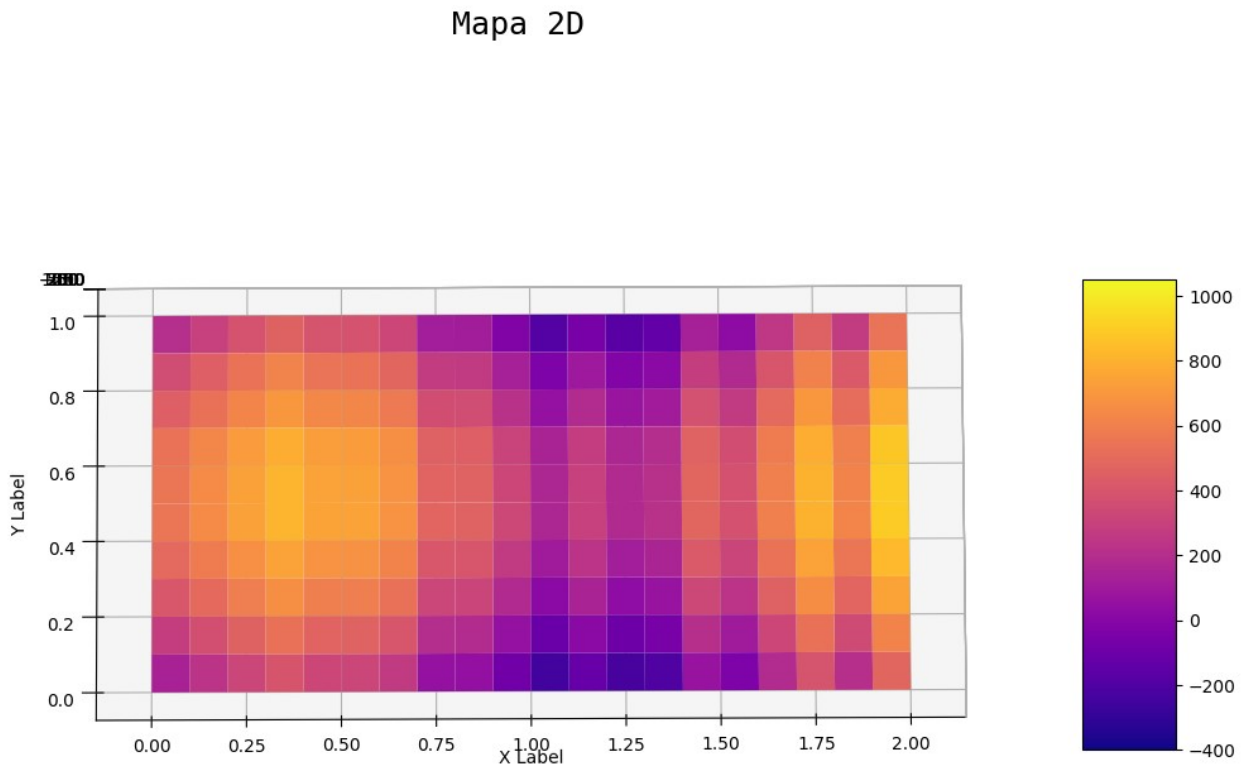
Dawid Ziora gr 3

Projekt Zaliczeniowy z Algorytmów Numerycznych

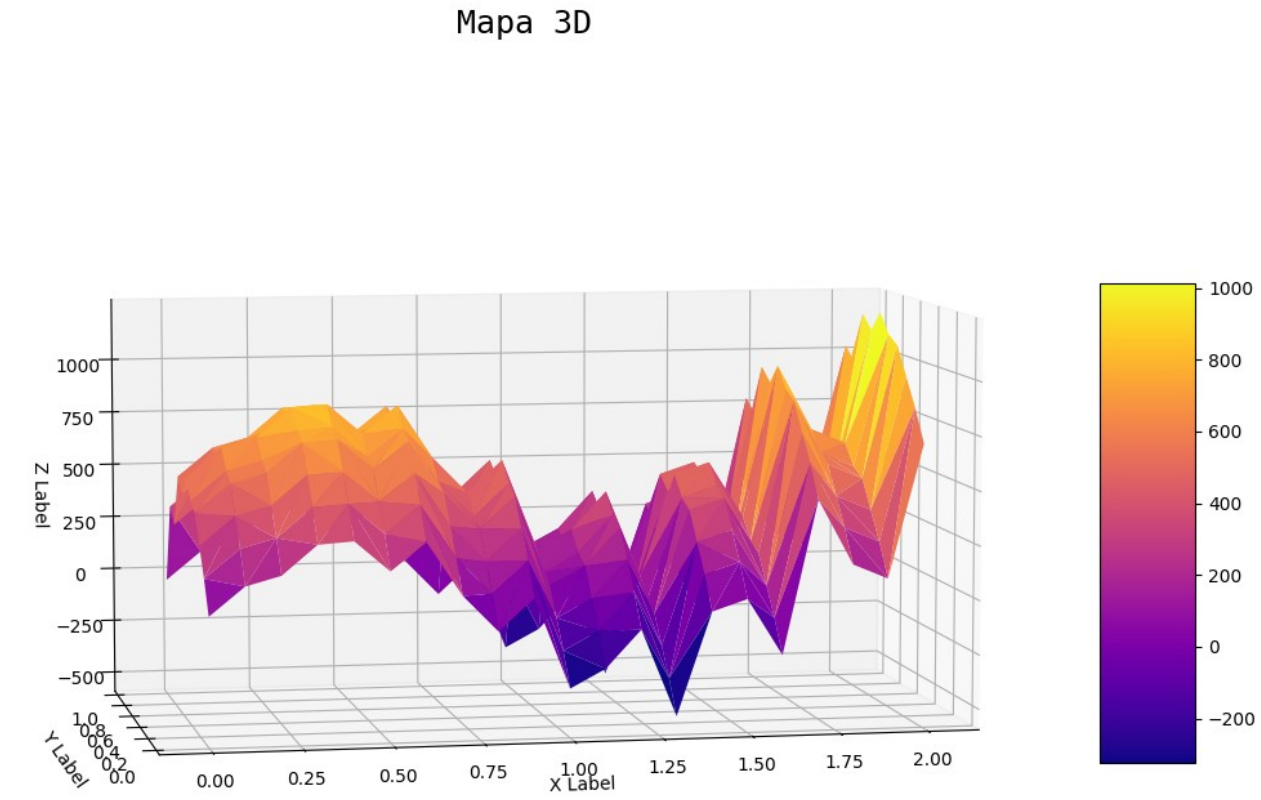
1. Wizualizacja danych w formie mapy 2D.

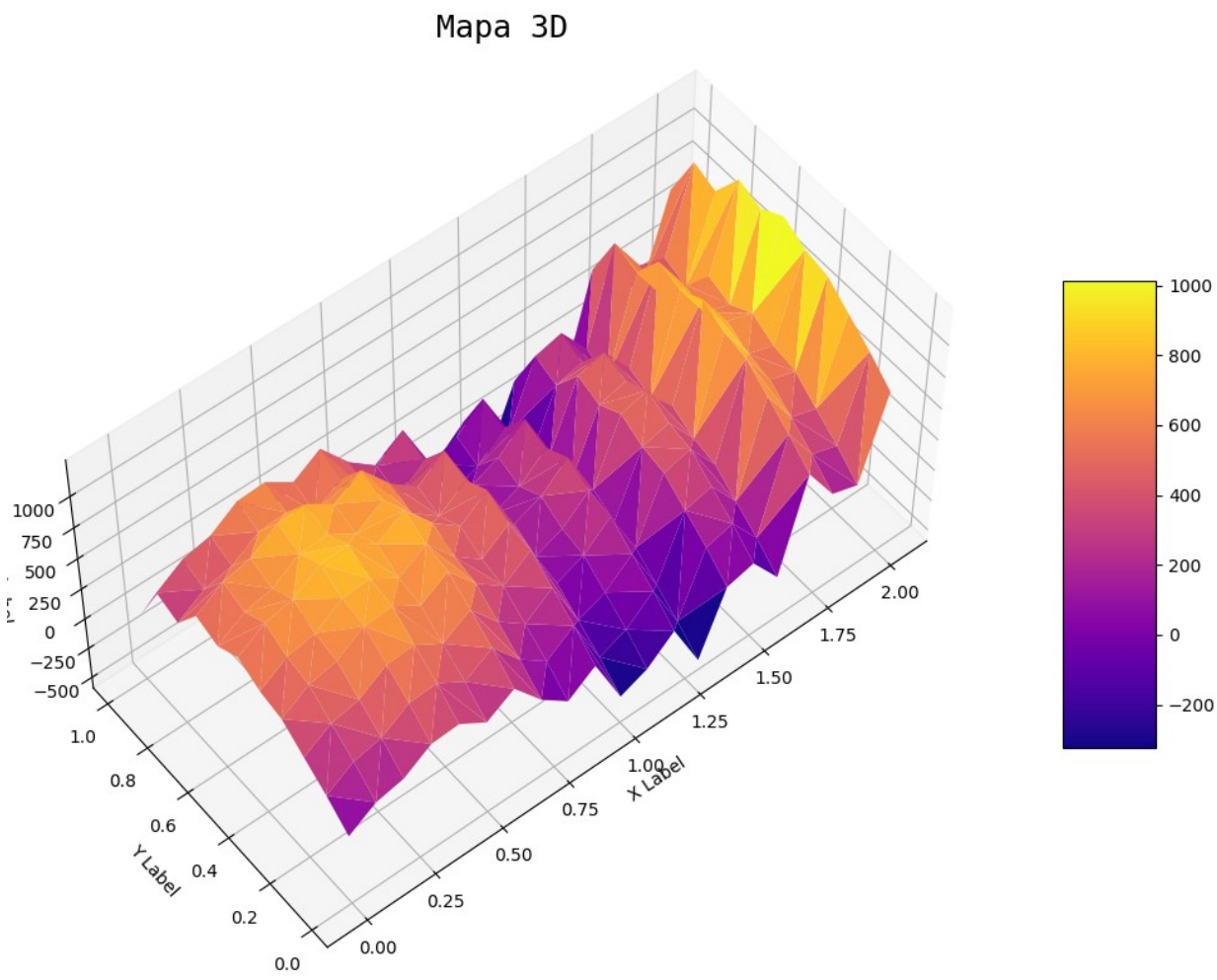


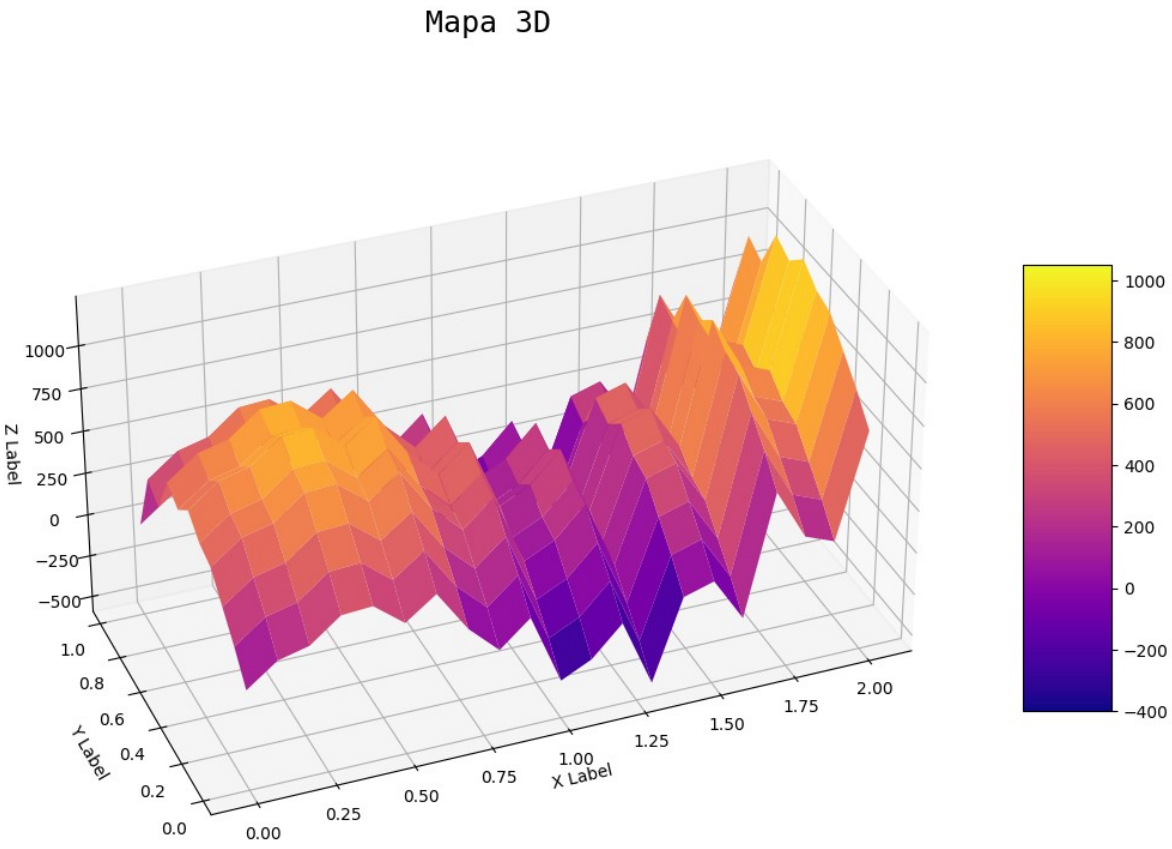
Mapy 2D to graficzne przedstawienie danych na płaszczyźnie dwuwymiarowej. Pozwalają bardziej intuicyjnie i czytelnie zrozumieć dane.
Na powyżej mapie możemy zauważyć funkcje f(X,Y).X należą do zakresu od <0,2> a Y od <0,1>. Kolorami ciepłymi oznaczone są najwyższe punkty tej funkcji , natomiast kolorami zimnymi najniższe. Jak możemy zaobserwować funkcja ta ma najmniejsze wartości mniej więcej dla punktów x=1.2 i wszystkich y dla tego x , czyli y <0,1>
najmniejsza wartość tej funkcji wynosi (-555.752). Największe wartości możemy zobaczyć po prawej stronie tej funkcji w punkcie x=2.0 i w y=0.5 i y=0.6 , największa wartość tej funkcji wynosi (1247.67).
Do utworzenia mapy tej funkcji skorzystałem z biblioteki matplotlib.pyplot.
Aby uzyskać taki efekt wykorzystałem funkcje plt.figure(), która pozwoliła mi na stworzenie nowej figury. Następnie za pomocą komendy add_subplot() dodałem do wcześniej utworzonej figury osie. Następnie utworzyłem powierzchnie wykorzystując komende plot_surface(). Po utworzeniu powierzchni wykorzystałem komendę set_box_aspect() aby zmniejszyć oś z do minimum by wykres był płaski.

funkcja

2.Wizualizacja danych w formie mapy 3D







Mapa 3D to graficzne przedstawienie danych w trójwymiarowej przestrzeni. W przeciwieństwie do map 2D, które wykorzystują dwuwymiarową płaszczyznę, mapy 3D umożliwiają wizualizację danych w 3 wymiarach , dodając trzeci wymiar głębokości.
Na powyższych mapach 3D widzimy przedstawienie funkcji F(X,Y) w trójwymiarze. Pierwsze dwie mapy są utworzone z trójkątów, dzięki czemu są one bardziej dokładne. Skorzystałem do tego z komendy plot_trisurf(). Natomiast trzecia mapa jest stworzony z kwadratów w sposób identyczny jak w punkcje pierwszym z tą różnicą że nie została spłaszczona przy użyciu komendy set_box_aspect().

Dzięki tym mapą możemy zauważyć jak wygląda teren, za który odpowiada funkcja F(X,Y).

funkcja[ii]

3.Wyznaczenie średniej , mediany oraz odchylenia standardowego.

a) Średnia

Wartości średnie dla każdej współrzędnej Y funkcji F(X,Y)

0.0 65.98438095238096
0.1 215.34796190476186
0.2 342.7780176190476
0.3 471.4418285714285
0.4 510.0714666666668
0.5 601.7952428571429
0.6 505.1224904761905
0.7 567.1136490476192
0.8 322.95736666666664
0.9 388.1033476190476
1.0 18.839323333333333

```
def srednia(self,y,z):
    for i in range(11):
        print(round(y[i][i],1),"   ",np.mean(z[i]))
```

Do wyliczenia średnich wartości funkcji użyłem funkcji np.mean() . Służy ona do obliczenia średniej wartości w zestawie danych, co daje nam ogólny pogląd na wartość przeciętną

b)mediana

Środkowy wyraz dla każdej współrzędnej Y funkcji F(x,y)

0.0 66.9558
0.1 216.319
0.2 343.749
0.3 472.413
0.4 511.043
0.5 602.767
0.6 506.094
0.7 568.085
0.8 323.929
0.9 389.075
1.0 19.8108

```
def mediana(self, y, z):
    for i in range(11):
        print(round(y[i][i],1),"   ",np.median(z[i]))
```

Do wyliczenia mediany wykorzystałem funkcje np.median(). Sortuje ona tablice w której znajdują się wyniki F(x,y), a potem wybiera wartość środkową.

c) odchylenie standardowego

Szerokość rozproszenia wartości funkcji F(x,y) wokół jej średniej arytmetycznej.

0.0 277.497031238609
0.1 277.4971199698158
0.2 277.4970522605369
0.3 277.4973304668115
0.4 277.49740117565807
0.5 277.4968667944586
0.6 277.4972184942056
0.7 277.4971635238896

```
def odchylenie(self, y, z):
    for i in range(11):
        print(round(y[i][i],1),"   ",np.std(z[i]))
```

0.8 277.4970436172186
0.9 277.49721037851657
1.0 277.4970629672303

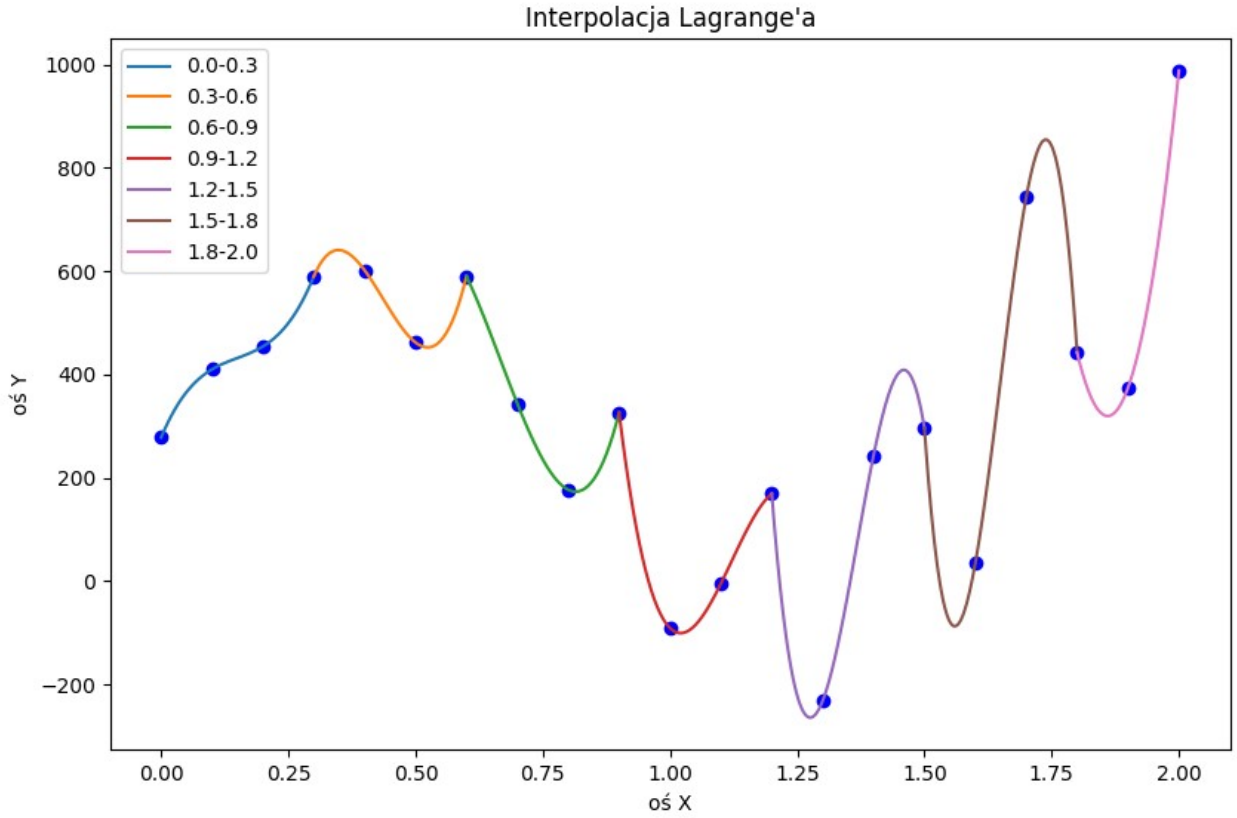
Do wyznaczenia odchylenia standardowego skorzystałem z funkcji np.std().

funkcje[iii]

4.Funkcje interpolacyjne dla wybranej współrzędnej Y

Interpolacja to technika matematyczna używana do wyznaczenia wartości między znanymi danymi punktowymi. Polega na budowaniu funkcji, która przechodzi przez te punkty. Jej celem jest estymacja wartości wewnątrz zakresu, dla którego mamy tylko ograniczoną liczbę pomiarów.

a)Interpolacja Lagrange'a – Jest metodą interpolacji wielomianowej, która pozwala na znalezienie wielomianu. Ze względu na swoją prostotę i intuicyjność metoda ta jest często stosowana na przykład w praktyce inżynierskiej.



Powyższy wykres Interpolacji Lagrange'a przedstawia funkcje dla y=0.2.
Aby otrzymać powyższy wykres funkcji skorzystałem ze wzorów na współczynnik wielomianu interpolacyjnego:

$$a_k = \frac{f_k}{(x_k - x_0)(x_k - x_2),\ldots,(x_k - x_{k-1})(x_k - x_{k+1}),\ldots,(x_k - x_n)} = \frac{f_k}{\prod_{j \neq k}(x_k - x_j)}$$

oraz na Wielomian interpolacyjny Lagrange'a

$$W(x) = \sum_{k=0}^{k=n} a_k (x - x_0)(x - x_2),\ldots,(x - x_{j-1})(x - x_{j+1}),\ldots,(x - x_n) = \sum_{k=0}^{k=n} a_k \prod_{j \neq k}(x - x_j)$$

Zdefiniowałem 4 funkcje
Pierwsza odpowiadała za mianownik dla współczynnika ak, czyli wyliczała wartość :

$$\prod_{j \neq k} (x_k - x_j)$$

```
def mianownik(self, x, k, n):
    m = 1
    for j in range(n):
        if j != k:
            m *= x[k]-x[j]
    return m
```

Druga funkcja odpowiadała za wyliczenie wartości iloczynu , we wzorze na Wielomian. Wykorzystałem do tego bardzo podobny wzór co w funkcji pierwszej z tą różnicą , że x jest stały.

Trzecia funkcja odpowiada za policzenie Wielomianu. Wykorzystałem do tego 2 poprzednie funkcje. Oraz wzoru na Wielomian interpolacyjny lagrange'a Funkcja wyznacza wartość ak i mnoży z wartością z funkcji 2. Zwraca wartość wielomianu.

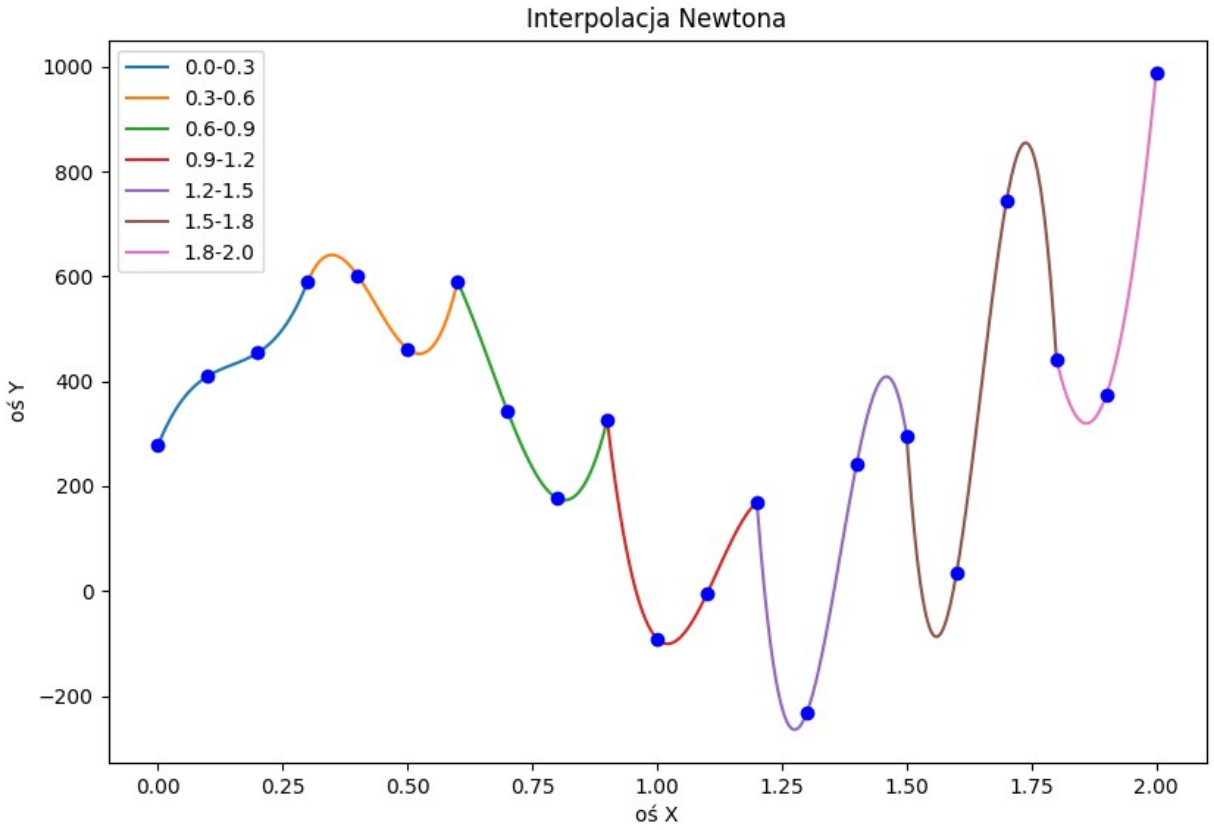
```
def lan(self, x, y, iks, n):
    W = 0
    for j in range(n):
        W += (y[j]/(projekt.mianownik(x, j, n))) * projekt.mianownik1(x, iks, j, n)
    return W
```

Czwarta funkcja dzieliła x i F(x,y) na mniejsze (4 elementowe) podzbiory , następnie dla każdego podzbioru liczyła wartość wielomianu i przedstawiała go na wykresie

```
def funl(self, X, Y):
    plt.plot(X, Y, 'bo')
    X=np.round(X,2)
    x1 = [X[i:i+4] for i in range(0,len(X)-1,3)]
    y1 = [Y[i:i+4] for i in range(0,len(Y)-1,3)]
    for i in range(len(x1)):
        m=len(x1[i])-1
        print (m)
        iks = np.linspace(x1[i][0], x1[i][m], 101)
        plt.plot(iks, projekt.lan(x1[i], y1[i], iks, m+1), label=(f"{x1[i][0]}-{x1[i][m]}"))
    plt.legend()
    plt.title("Interpolacja Lagrange'a")
    plt.xlabel("oś X")
    plt.ylabel("oś Y")
    plt.show()
```

funkcje[iv]

b) Interpolacja Newtona



Powyższy wykres Interpolacij Newtona przedstawia funkcje dla y=0.2.
Interpolacja Newtona jest to jedna z metod przedstawiania Wielomianu. Dla wielomianu n wybiera się n+1 punktów i buduje wielomian postaci :

$$W(x)=\sum_{k=0}^{k=n} a_k\varphi_k(x)=a_0\varphi_0(x)+a_1\varphi_1(x)+a_2\varphi_2(x),\ldots,+a_k\varphi_k(x),\ldots,+a_n\varphi_n(x)$$

Aby utworzyć powyższy wykres zdefiniowałem 3 funkcje.
Pierwsza odpowiadała za wyliczanie współczynników i uzupełnianie macierzy.
Druga funkcja na podstawie wcześniej wyliczonych współczynników wyliczała wartość wielomianu. Trzecia funkcja dzieliła zmienne na mniejsze podzbiory (4 Elementowe) oraz liczyła dla nich wartość wielomianu i przedstawiała na wykresie. Funkcja[v]

```
def funN(self,X, Y):
    X=np.round(X,2)
    x1 = [X[i:i+4] for i in range(0,len(X)-1,3)]
    y1 = [Y[i:i+4] for i in range(0,len(Y)-1,3)]
    for i in range(len(x1)):
        m=len(x1[i])-1
        iks=np.linspace(x1[i][0],x1[i][m],101)
        y=projekt.newton(projekt.roznica(x1[i],y1[i])[0,:],x1[i],iks)
        plt.plot(iks, y,label=(f"{x1[i][0]}-{x1[i][m]}"))
    plt.plot(X, Y, 'bo')
    plt.legend()
    plt.title("Interpolacja Newtona")
    plt.xlabel("oś X")
    plt.ylabel("oś Y")
    plt.show()
```

c) Różnica między interpolacjami

Różnica między tymi interpolacjami (Lagranga - Newtona) wynoszą dla 10 wartości na każdym przedziale.

1.
[0.00000000e+00 0.00000000e+00 0.00000000e+00 -5.68434189e-14
 -1.13686838e-13 -1.13686838e-13 -5.68434189e-14 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
2.
[0.00000000e+00 1.13686838e-13 1.13686838e-13 1.13686838e-13
  0.00000000e+00 0.00000000e+00 0.00000000e+00 -5.68434189e-14
  1.13686838e-13 0.00000000e+00]
3.
[0.00000000e+00 1.13686838e-13 0.00000000e+00 0.00000000e+00
 -5.68434189e-14 -2.84217094e-14 2.84217094e-14 -2.84217094e-14
  0.00000000e+00 0.00000000e+00]
4.
[0.00000000e+00 1.42108547e-14 4.26325641e-14 7.10542736e-14
  5.68434189e-14 2.13162821e-14 -8.88178420e-15 -7.10542736e-14
 -1.42108547e-14 -5.68434189e-14]
5.
[0.00000000e+00 -8.52651283e-14 0.00000000e+00 0.00000000e+00
  4.26325641e-14 9.94759830e-14 1.13686838e-13 1.13686838e-13
  1.70530257e-13 0.00000000e+00]
6.
[0.00000000e+00 1.06581410e-14 -1.42108547e-14 -2.13162821e-14
  0.00000000e+00 -1.13686838e-13 -2.27373675e-13 -3.41060513e-13
  0.00000000e+00 -4.54747351e-13]
7.
[0.00000000e+00 -5.68434189e-14 -5.68434189e-14 5.68434189e-14
  5.68434189e-14 0.00000000e+00 1.13686838e-13 0.00000000e+00
  1.13686838e-13 -1.13686838e-13]

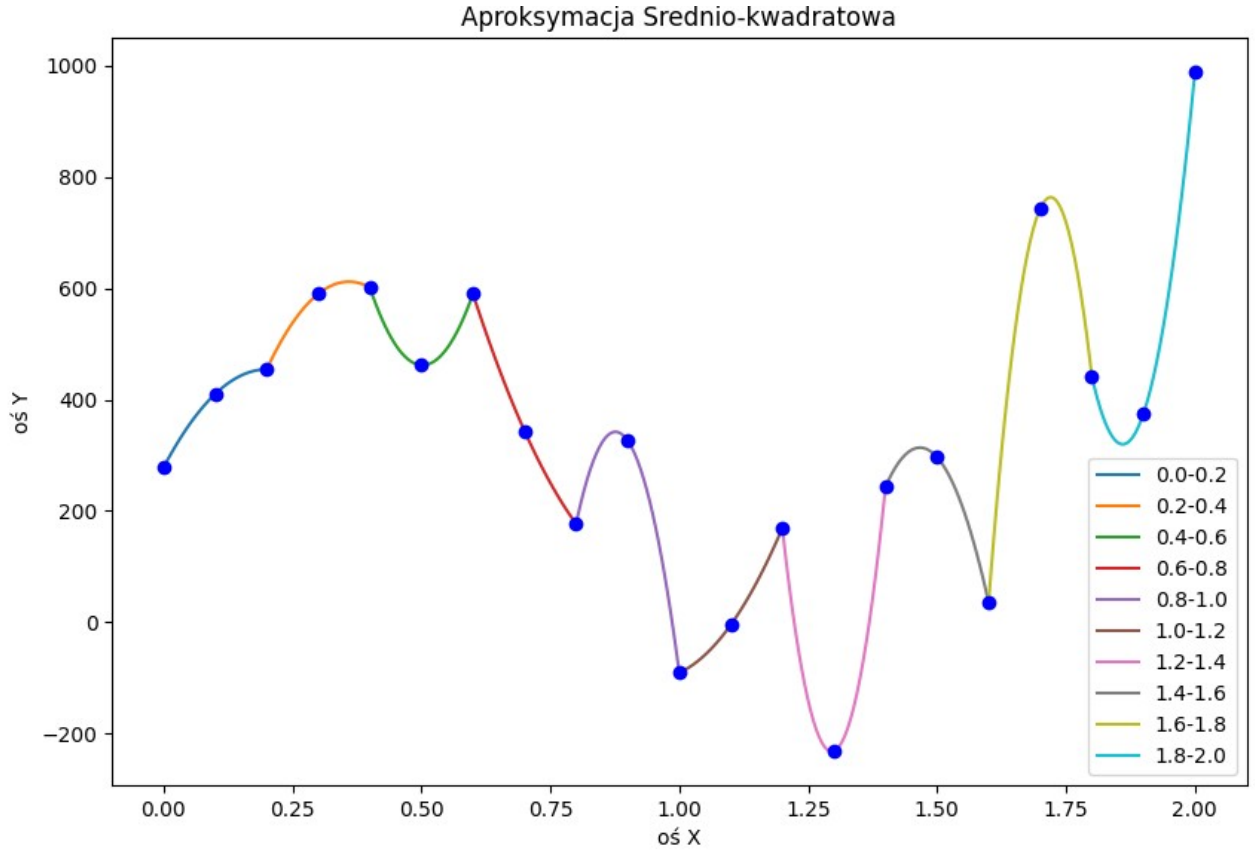
Jak możemy zauważyć wartości te są bliskie zeru , co oznacza że funkcje te zwracają bardzo podobne wyniki.

Funkcja[vi]

5. Funkcje aproksymacyjne dla wybranej współrzędnej Y

Aproksymacja to technika matematyczna polegająca na znalezieniu funkcji, która najlepiej przybliża podane dane.

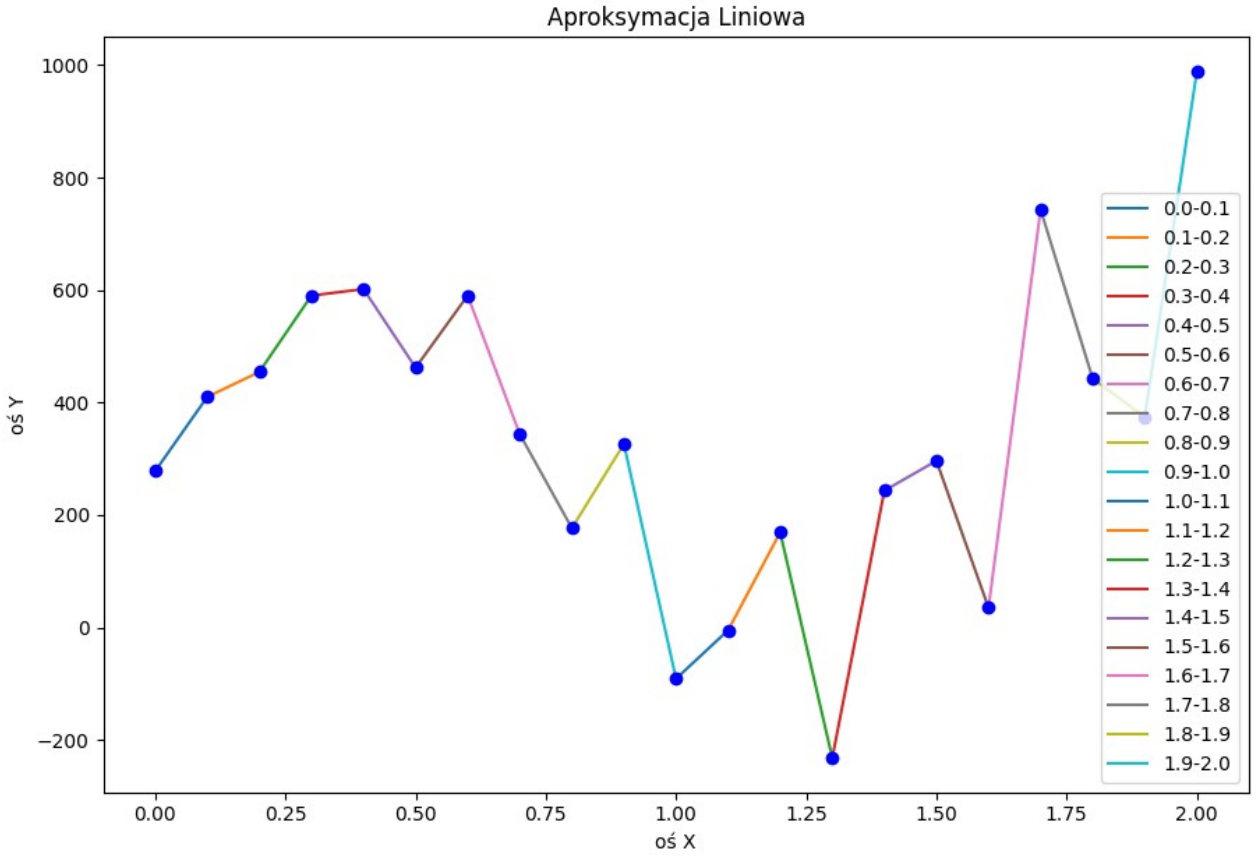
a) Aproksymacja średniokwadratowa – Znana również jako metoda najmniejszych kwadratów. To technika matematyczna służąca do znalezienia najlepszego dopasowania funkcji do zestawu danych, minimalizując sumę kwadratów różnic między wartościami obliczonymi a rzeczywistymi.



Powyższy wykres Aproksymacji średnio-kwadratowej przedstawia funkcje dla y=0.2.
Aby utworzyć powyższy wykres zdefiniowałem 4 funkcje.
Pierwsza funkcja służyła do obliczania układów równań metodą eliminacji Gaussa.
Druga funkcja tworzyła macierz o wymiarach 3x3 oraz uzupełniała ją według schematu
Następnie rozwiązywała macierz za pomocą metody eliminacji Gausaa i zwracała wartości , które obliczyła na postawie macierzy.
Trzecia funkcja była wzorem funkcji czyli f(a0,a1,a2,x) = a0+a1*x+a2*x².
Czwarta funkcja dzieliła na mniejsze podzbiory wartości F(x,y) oraz obliczała ich wartość i przedstawiała na wykresie. Funkcja[vii]

```
def funA(self, X, Y):
    X=np.round(X,2)
    x1 = [X[i:i+3] for i in range(0,len(X)-1,2)]
    y1 = [Y[i:i+3] for i in range(0,len(Y)-1,2)]
    for i in range(len(x1)):
        m=len(x1[i])-1
        K1 = projekt.aproksymacja3(x1[i], y1[i])
        iks = np.linspace(x1[i][0], x1[i][m], 101)
        plt.plot(iks, projekt.f1(K1[0], K1[1], K1[2], iks),label=(f"{x1[i][0]}-{x1[i][m]}"))
    plt.plot(X, Y, 'bo')
    plt.legend()
    plt.title("Aproksymacja Średnio-kwadratowa")
    plt.xlabel("oś X")
    plt.ylabel("oś Y")
    plt.show()
```

b) Aproksymacja liniowa – Znana również jako regresja liniowa, to podstawowa technika aproksymacyjna stosowana w analizie danych. Polega na znalezieniu prostej linii, która najlepiej pasuje do zestawu danych. Aproksymacja ta ma swoje ograniczenia i może być niewłaściwa w przypadku danych o nieliniowych zależnościach.



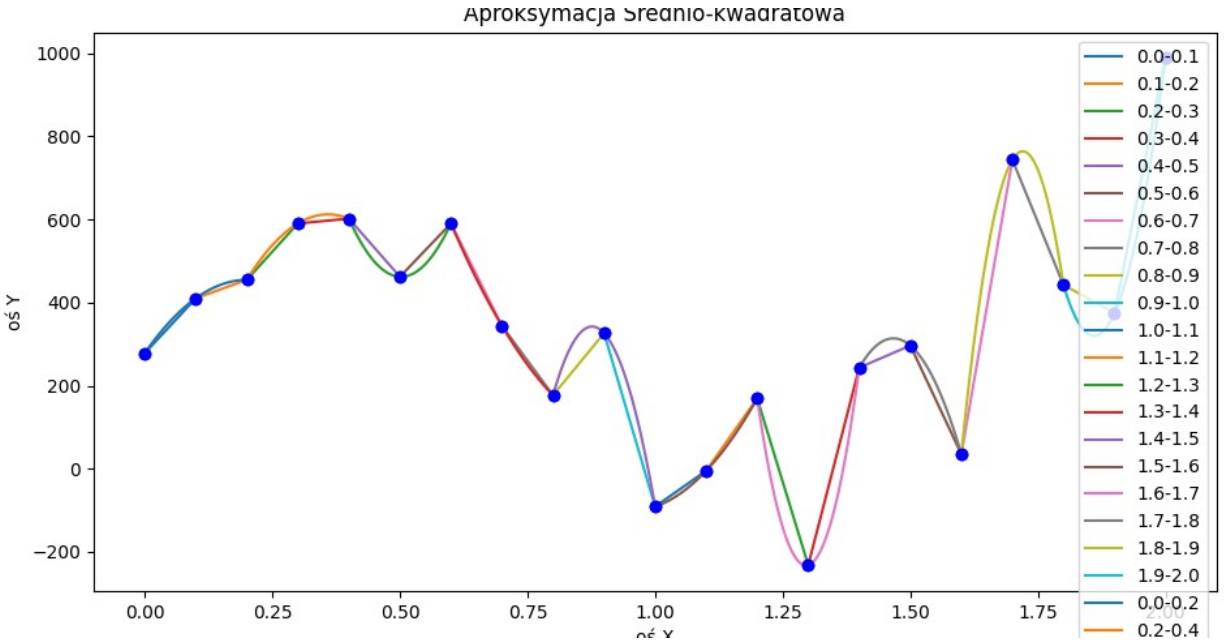
Powyższy wykres Aproksymacji liniowej przedstawia funkcje dla y=0.2.
Aby przedstawić powyższy wykres zdefiniowałem 4 funkcje.
Pierwsza funkcja służyła do obliczania układów równań metodą eliminacji Gaussa.
Druga funkcja tworzyła macierz 2x2 oraz uzupełniała ją według schematu. Następnie rozwiązywała macierz za pomocą metody eliminacji Gaussa i zwracała wartości które obliczyła na podstawie rozwiązywania układów równań.
Trzecia funkcja była wzorem funkcji f(a0,a1,x)=a0+a1*x
Czwarta funkcja dzieliła na mniejsze podzbiory wartości F(x,y) oraz obliczała ich wartości i przedstawiała na wykresie.

Funkcja[viii]

c)Różnica między Aproksymacjami Dla danego punktu
W tym przypadku na przedziale od x=0 do x=0.2

[2.27373675e-13 2.29439486e+00 4.50602239e+00 6.63488258e+00
 8.68097543e+00 1.06443009e+01 1.25248591e+01 1.43226500e+01
 1.60376735e+01 1.76699296e+01 1.92194185e+01 2.06861400e+01
 2.20700941e+01 2.33712809e+01 2.45897004e+01 2.57253526e+01
 2.67782374e+01 2.77483548e+01 2.86357050e+01 2.94402878e+01
 3.01621032e+01 3.08011513e+01 3.13574321e+01 3.18309456e+01
 3.22216917e+01 3.25296704e+01 3.27548819e+01 3.28973260e+01
 3.29570027e+01 3.29339121e+01 3.28280542e+01 3.26394290e+01
 3.23680364e+01 3.20138764e+01 3.15769492e+01 3.10572546e+01
 3.04547926e+01 2.97695634e+01 2.90015667e+01 2.81508028e+01
 2.72172715e+01 2.62009729e+01 2.51019069e+01 2.39200736e+01]

Jak możemy zauważyć błąd między tymi funkcjami jest dosyć znaczący . W niektórych miejscach wartości różnią się od siebie o 33 jednostki co jest dużym błędem jeśli chodzi o takie funkcje.
Z porównania tych funkcji wynika że aproksymacja Średnio-Kwadratowa jest bardziej dokładna niż aproksymacja liniowa.



Funkcja[ix]

6. Pole powierzchni funkcji F(x,y)

Pole powierzchni = 5915.699075680908

Do policzenia pola powierzchni utworzyłem funkcje Pole która za pomocą komendy Delaunay z biblioteki scipy , Dzieli ona przestrzeń na trójkątne elementy w taki sposób, że żadne dwa punkty nie znajdują się wewnątrz okręgu opisanego na trójkącie. Następnie za pomocą komendy simplices wyznaczam indeksy trójkątów. Następnie Wyliczam wartość dla każdego trójkąta. Suma wszystkich wartości trójkątów daje mi pole powierzchni równe 5915.699075680908.

Funkcja[x]

7.Całki z funkcji interpolacyjnej oraz aproksymacyjnej.

Całka jest pojęciem matematycznym, które odwrotnie do pochodnej opisuje akumulacje zmiany wartości funkcji na określonym przedziale.

Do obliczenia całki wykorzystałem funkcje liczącą całkę Simpsona
Całka Simpsona to jedna z technik numerycznego obliczania wartości całki oznaczonej.

Aby obliczyć całkę utworzyłem funkcje CałkaSa , która oblicza całkę na podstawie danych z funkcji interpolacyjne oraz aproksymacyjnej.

```
def calkSa(self, K1,x1):
    cp = 0
    xs = 0
    ys = 0
    n=10
    a=x1[0]
    b=x1[2]

    x=np.zeros(n+1)
    y=np.zeros(n+1)
    h=(b-a)/n
    for i in range(n+1):
        x[i]=a+i*h
        y[i]=(projekt.f1(K1[0], K1[1], K1[2],x[i]))
        h = (b-a)/n
    for i in range(n):
        xs = (x[i]+x[i+1])/2
        ys = (projekt.f1(K1[0], K1[1], K1[2],xs))
        cp += h*((y[i]+y[i+1]+4*ys)/6)
    return cp
```

Całka dla y=0.2 ograniczona od 0.4 do 0.6 dla wartości F(x,y) w punkatch x=0.4 do x=0.6.

Całka Dokładna: 101.330466666667
Całka z Aproksymacji średnio-kwadratowej: 101.33046666666641
Całka Dokładna: 101.330466666666
Całka z interpolacij Lagrange'a: 101.33046666666668

Funkcje[xi]

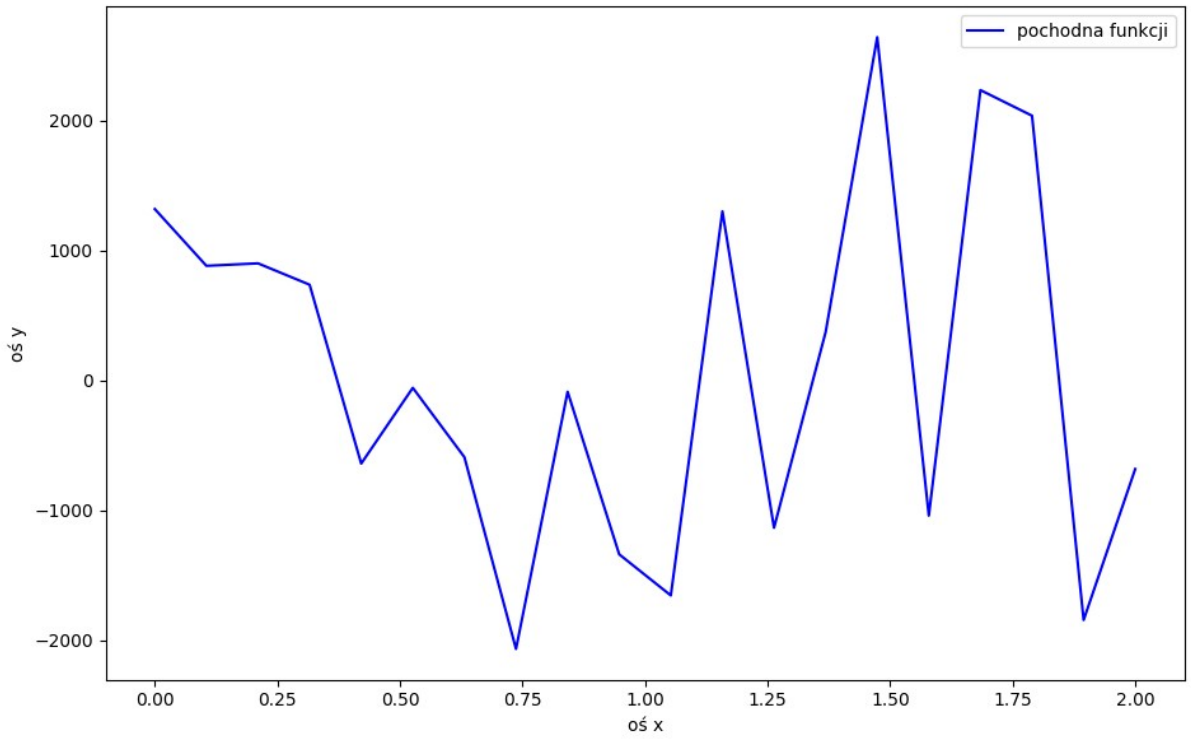
8.Pochodne funkcji.

Pochodna jest pojęciem matematycznym, które opisuje szybkość zmiany funkcji w zależności od zmiennej niezależnej.

Do obliczenia pochodnej funkcji skorzystałem z różniczkowania numerycznego.
Funkcja którą uworzyłem liczyła pochodną przy użyciu wzoru :

Za f(x) podstawiała współrzędne F(x,y) , a za x podstawiała współrzędne x.

$$f^1(x_0) = \frac{f(x_1) - f(x_{-1})}{(x_1 - x_{-1})}$$



Na wykresie powyżej widzimy jak przebiega pochodna dla F(x,y) w punkcie y=0.2

9.Monotoniczność funkcji

Funkcja monotoniczna to funkcja której argument na danym przedziale rośnie albo maleje.

Do obliczenia Monotoniczności funkcji możemy wykorzystać pochodną. Jeżeli pochodna jest niedodatnia to funkcja jest monotoniczna nierosnąca, oraz niemalejąca gdy pochodna jest nieujemna.

Z obliczenia monotoniczności funkcji w punkcie y=0.2 wynika że funkcja nie jest monotoniczna.

Funkcja[xii]

Podsumowując język python pozwala nam w szybki i prosty sposób dokonywać operacji na funkcjach , obliczać ich wartości i przedstawiać dane na wykresach.

Cały kod gotowy do kompilacji :

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.interpolate import lagrange
from scipy.interpolate import interp1d
from scipy.interpolate import interp2d
from scipy.integrate import quad
from scipy.integrate import solve_ivp
from scipy.integrate import dblquad
from scipy.spatial import Delaunay
import sympy as sp


ma = np.loadtxt('136700.dat')
n = 21
m = 11

x = np.linspace(0, 2, n)
y = np.linspace(0, 1, m)
x, y = np.meshgrid(x, y)
z = np.zeros((m, n))


class Projekt:
    def __init__(self):
        self.wartosc = []

    def wyznacz_xyz(self, ma):
        n = ma.shape[0]
        x1, y1, z1 = np.zeros(n), np.zeros(n), np.zeros(n)
        for i in range(n):
            x1[i] = ma[i][0]
            y1[i] = ma[i][1]
            z1[i] = ma[i][2]
        return x1, y1, z1

    def wyznacz(self, z, n, m, ma):
        k = 0
        for i in range(m):
            for j in range(n):
                z[i][j] = ma[k, 2]
                k += 1

    def wykres2D(self, x, y, z):
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        surf = ax.plot_surface(x, y, z, cmap='plasma', edgecolor='none',vmin=-400, vmax=1050)
```

```
    plt.title('Mapa 2D', fontdict={
            'fontname': 'monospace', 'fontsize': 18})
    fig.colorbar(surf, shrink=0.5, aspect=5)
    ax.set_box_aspect([2, 1, 0.0001])
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    plt.show()

def wykres3D(self, x, y, z):

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    surf = ax.plot_surface(x, y, z, cmap='plasma', edgecolor='none',vmin=-400, vmax=1050)
    plt.title('Mapa 3D', fontdict={
            'fontname': 'monospace', 'fontsize': 18})
    fig.colorbar(surf, shrink=0.5, aspect=5)
    ax.set_box_aspect([2, 1, 1])
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')
    plt.show()

def srednia(self,y,z):
    print("Średnia arytmetyczna: ")
    for i in range(11):
        print(round(y[i][i],1),"   ",np.mean(z[i]))


def mediana(self, y, z):
    print("Mediana : ")
    for i in range(11):
        print(round(y[i][i],1),"   ",np.median(z[i]))

def odchylenie(self, y, z):
    print("Odchylenie standardowe: ")
    for i in range(11):
        print(round(y[i][i],1),"   ",np.std(z[i]))

def mianownik(self, x, k, n):
    m = 1
    for j in range(n):
        if j != k:
            m *= x[k]-x[j]
    return m

def mianownik1(self, x, x1, k, n):
    m = 1
    for j in range(n):
        if j != k:
            m *= x1-x[j]
    return m
```

```
def lan(self, x, y, iks, n):
    W = 0
    for j in range(n):
        W += (y[j]/(projekt.mianownik(x, j, n))) * projekt.mianownik1(x, iks, j, n)
    return W

def funl(self, X, Y):
    plt.plot(X, Y, 'bo')
    X=np.round(X,2)
    x1 = [X[i:i+4] for i in range(0,len(X)-1,3)]
    y1 = [Y[i:i+4] for i in range(0,len(Y)-1,3)]
    for i in range(len(x1)):
        m=len(x1[i])-1
        iks = np.linspace(x1[i][0], x1[i][m], 101)
        plt.plot(iks, projekt.lan(x1[i], y1[i], iks, m+1), label=(f"{x1[i][0]}-{x1[i][m]}"))
    plt.legend()
    plt.title("Interpolacja Lagrange'a")
    plt.xlabel("oś X")
    plt.ylabel("oś Y")
    plt.show()
def roznica(self,x, y):

    n = len(y)
    coef = np.zeros([n, n])
    coef[:,0] = y
    for j in range(1, n):
        for i in range(n-j):
            coef[i][j] = \
                (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j]-x[i])

    return coef


def newton(self,coef, x_data, x):

    n = len(x_data) - 1
    p = coef[n]
    for k in range(1, n+1):
        p = coef[n-k] + (x - x_data[n-k])*p
    return p


def funN(self,X, Y):
    X=np.round(X,2)
    x1 = [X[i:i+4] for i in range(0,len(X)-1,3)]
    y1 = [Y[i:i+4] for i in range(0,len(Y)-1,3)]
    for i in range(len(x1)):
        m=len(x1[i])-1
        iks=np.linspace(x1[i][0],x1[i][m],101)
        y=projekt.newton(projekt.roznica(x1[i],y1[i])[0,:],x1[i],iks)
        plt.plot(iks, y,label=(f"{x1[i][0]}-{x1[i][m]}"))
    plt.plot(X, Y, 'bo')
```

```
    plt.legend()
    plt.title("Interpolacja Newtona")
    plt.xlabel("oś X")
    plt.ylabel("oś Y")
    plt.show()

def aproksymacja2(self,x,y):
    M=np.zeros((2,2))
    P=np.zeros (2)
    n=2
    M[0][0]=n
    for i in range (n):
        M[0][1]=M[1][0]+x[i]
        M[1][0]=M[1][0]+x[i]
        M[1][1]=M[1][1]+x[i]**2
    for i in range (n):
        P[0]=P[0]+y[i]
        P[1]=P[1]+x[i]*y[i]
    K=projekt.gauss(M,P)
    return K
def f0(self,a0,a1,x):
    return a0+a1*x

def aproksymacja3(self, x, y):
    M1 = np.zeros((3, 3))
    P1 = np.zeros(3)
    n = 3
    M1[0, 0] = n
    for i in range(n):
        M1[0, 1] = M1[0, 1]+x[i]
        M1[0, 2] = M1[0, 2]+x[i]**2
        M1[1, 0] = M1[1, 0]+x[i]
        M1[1, 1] = M1[1, 1]+x[i]**2
        M1[1, 2] = M1[1, 2]+x[i]**3
        M1[2, 0] = M1[2, 0]+x[i]**2
        M1[2, 1] = M1[2, 1]+x[i]**3
        M1[2, 2] = M1[2, 2]+x[i]**4

    for i in range(n):
        P1[0] += y[i]
        P1[1] += x[i]*y[i]
        P1[2] += (x[i]**2)*y[i]

    K1 = projekt.gauss(M1,P1)
    return K1

def f1(self, a0, a1, a2, x):
    return a0+a1*x+a2*x**2
def fx1(self,a1,a2,x):
    return a1+2*a2*x
def aproksymacja4(self, x, y):
    M1 = np.zeros((4, 4))
```

```
    P1 = np.zeros(4)
    n = 4
    M1[0, 0] = n
    for i in range(n):
        M1[0, 1] = M1[0, 1]+x[i]
        M1[0, 2] = M1[0, 2]+x[i]**2
        M1[0, 3] = M1[0, 3]+x[i]**3
        M1[1, 0] = M1[1, 0]+x[i]
        M1[1, 1] = M1[1, 1]+x[i]**2
        M1[1, 2] = M1[1, 2]+x[i]**3
        M1[1, 3] = M1[1, 3]+x[i]**4
        M1[2, 0] = M1[2, 0]+x[i]**2
        M1[2, 1] = M1[2, 1]+x[i]**3
        M1[2, 2] = M1[2, 2]+x[i]**4
        M1[2, 3] = M1[2, 3]+x[i]**5
        M1[3, 0] = M1[3, 0]+x[i]**3
        M1[3, 1] = M1[3, 1]+x[i]**4
        M1[3, 2] = M1[3, 2]+x[i]**5
        M1[3, 3] = M1[3, 3]+x[i]**6

    for i in range(n):
        P1[0] += y[i]
        P1[1] += x[i]*y[i]
        P1[2] += (x[i]**2)*y[i]
        P1[3] += (x[i]**3)*y[i]

    K1 = projekt.gauss(M1,P1)
    print(K1)
    return K1

def f2(self, a0, a1, a2, a3, x):
    return a0+a1*x+a2*x**2+a3*x**3

def funA(self, X, Y):
    X=np.round(X,2)
    x1 = [X[i:i+3] for i in range(0,len(X)-1,2)]
    y1 = [Y[i:i+3] for i in range(0,len(Y)-1,2)]
    for i in range(len(x1)):
        m=len(x1[i])-1
        K1 = projekt.aproksymacja3(x1[i], y1[i])
        iks = np.linspace(x1[i][0], x1[i][m], 101)
        plt.plot(iks, projekt.f1(K1[0], K1[1], K1[2], iks),label=(f"{x1[i][0]}-{x1[i][m]}"))
    plt.plot(X, Y, 'bo')
    plt.legend()
    plt.title("Aproksymacja Średnio-kwadratowa")
    plt.xlabel("oś X")
    plt.ylabel("oś Y")
    plt.show()
def funAl(self, X, Y):
    X=np.round(X,2)
    x1 = [X[i:i+2] for i in range(0,len(X)-1,1)]
    y1 = [Y[i:i+2] for i in range(0,len(Y)-1,1)]
```

```
    for i in range(len(x1)):
        m=len(x1[i])-1
        K1 = projekt.aproksymacja2(x1[i], y1[i])
        iks = np.linspace(x1[i][0], x1[i][m], 101)
        plt.plot(iks, projekt.f0(K1[0], K1[1],iks),label=(f"{x1[i][0]}-{x1[i][m]}"))
    plt.plot(X, Y, 'bo')
    plt.legend()
    plt.title("Aproksymacja Liniowa")
    plt.xlabel("oś X")
    plt.ylabel("oś Y")
    plt.show()

def pole(sefl,ma):
    tri = Delaunay(ma[:, :2])
    triangle_areas = []
    for i in tri.simplices:
        p0, p1, p2 = ma[i]
        triangle_areas.append(
            0.5 * np.linalg.norm(
                np.cross(p1 - p0, p2 - p0)
            )
        )
    surface_area = np.sum(triangle_areas)
    print("Pole powierzchni: ", surface_area)

def calkSa(self, K1,x1):
    cp = 0
    xs = 0
    ys = 0
    n=10
    a=x1[0]
    b=x1[2]

    x=np.zeros(n+1)
    y=np.zeros(n+1)
    h=(b-a)/n
    for i in range(n+1):
        x[i]=a+i*h
        y[i]=(projekt.f1(K1[0], K1[1], K1[2],x[i]))
        h = (b-a)/n
    for i in range(n):
        xs = (x[i]+x[i+1])/2
        ys = (projekt.f1(K1[0], K1[1], K1[2],xs))
        cp += h*((y[i]+y[i+1]+4*ys)/6)
    return cp
def calkSa1(self, x3,y3):
    cp = 0
    xs = 0
    ys = 0
    n=10
    a=x3[0]
    b=x3[2]
```

```
    x=np.zeros(n+1)
    y=np.zeros(n+1)
    h=(b-a)/n
    for i in range(n+1):
        x[i]=a+i*h
        y[i]=(projekt.lan(x3,y3,x[i],3))
        h = (b-a)/n
    for i in range(n):
        xs = (x[i]+x[i+1])/2
        ys = (projekt.lan(x3,y3,xs,3))
        cp += h*((y[i]+y[i+1]+4*ys)/6)
    return cp

def calkiIiA(self,x,z):
    x1 = [x[2][i:i+3] for i in range(0,len(x[0])-1,2)]
    y1 = [z[2][i:i+3] for i in range(0,len(z[0])-1,2)]

    X=sp.symbols('x')

    K1=projekt.aproksymacja3(x1[2],y1[2])
    a=x1[2][0]
    b=x1[2][2]
    cd = sp.integrate(projekt.f1(K1[0],K1[1],K1[2],X),(X,a,b))
    sa=projekt.calkSa(K1,x1[2])
    print ("Całka Dokładna: ",cd," Całka z Aproksymacji średnio-kwadratowej: ",sa)

    cd1 = sp.integrate(projekt.lan(x1[2],y1[2],X,3),(X,a,b))
    sa1=projekt.calkSa1(x1[2],y1[2])
    print ("Całka Dokładna: ",cd1," Całka z interpolacij Lagrange'a: ",sa1)


def wypz(self,x,K1,):
    z1=np.zeros(20)
    zx1=np.zeros(20)
    print("Wartosc funkcji:",projekt.f1(K1[0],K1[1],K1[2],x))
    z1=(projekt.f1(K1[0],K1[1],K1[2],x))
    zx1=projekt.fx1(K1[1],K1[2],x)
    return z1,zx1

def poch(self,pz,px):
    n=20
    zx=np.zeros(n)
    for i in range (n):
        if(i==0):
            zx[i]=(pz[i+1]-pz[i])/(px[i+1]-px[i])
        elif(i==n-1):
            zx[i]=(pz[i]-pz[i-1])/(px[i]-px[i-1])
        else:
            zx[i]=(pz[i+1]-pz[i-1])/(px[i+1]-px[i-1])
    return zx
def monotonicznosc(self,z):
```

```
    n = len(x)

    rosnie = all((z[i]) <= (z[i+1]) for i in range(n-1))
    maleje = all((z[i]) >= (z[i+1]) for i in range(n-1))

    if rosnie:
        print("Funkcja jest rosnąca dla każdego punktu.")
    elif maleje:
        print("Funkcja jest malejąca dla każdego punktu.")
    else:
        print("Funkcja nie jest monotoniczna dla każdego punktu.")


def poch1(self,x,z):
    iks=np.linspace(0,2,20)
    zx=projekt.poch(z,x)

    plt.plot(iks,zx,'b',label="pochodna funkcji")
    plt.legend()
    plt.xlabel("oś x")
    plt.ylabel("oś y")
    plt.show()
    return zx

def iniciuj(self,X, Y):
    X=np.round(X,2)
    x1 = [X[i:i+4] for i in range(0,len(X)-1,3)]
    y1 = [Y[i:i+4] for i in range(0,len(Y)-1,3)]
    for i in range(len(x1)):
        m=len(x1[i])-1
        x_new=np.linspace(x1[i][0],x1[i][m],10)
        y_new=projekt.newton(projekt.roznica(x1[i],y1[i])[0,:],x1[i],x_new)
        print(projekt.lan(x1[i],y1[i],x_new,m+1)-y_new)

def iniciuj1(self,X, Y):
    X=np.round(X,2)
    x1 = [X[i:i+3] for i in range(0,len(X)-1,2)]
    y1 = [Y[i:i+3] for i in range(0,len(Y)-1,2)]
    m=len(x1[1])-1
    K1 = projekt.aproksymacja3(x1[0], y1[0])
    iks = np.linspace(x1[0][0], x1[0][m], 66)
    W1=projekt.f1(K1[0], K1[1], K1[2], iks)
    X=np.round(X,2)
    x1 = [X[i:i+2] for i in range(0,len(X)-1,1)]
    y1 = [Y[i:i+2] for i in range(0,len(Y)-1,1)]
    m=len(x1[1])-1
    K1 = projekt.aproksymacja2(x1[0], y1[0])
    iks = np.linspace(x1[0][0], x1[0][m], 44)
    W2=projekt.f0(K1[0], K1[1],iks)
    print(W1[0:44],"\n",W2)
    print(W1[0:44]-W2)
```

```
    def gauss(self,A,b):
        n=A.shape[0]
        C=np.zeros((n,n+1))
        C[:,0:n]=A
        C[:,n]=b
        x=np.zeros(n)
        for s in range(0, n-1):
            for i in range(s+1, n):
                # L = A[i,s] / A[s,s]
                for j in range(s+1, n+1):
                    C[i,j] = C[i,j] - (C[i,s] / C[s,s]) * C[s,j]
        x[n-1] = C[n-1,n]/C[n-1,n-1]
        for i in range(n-2,-1,-1):
            suma = 0.0
            for s in range(i+1, n):
                suma = suma + C[i,s] * x[s]
            x[i] = (C[i,n] - suma) / C[i,i]
        return x

projekt = Projekt()
projekt.wyznacz(z, n, m, ma)#Wyznacza Z
projekt.wykres2D(x,y,z)     #Wykres 2D
projekt.wykres3D(x,y,z)     #Wykres 3D
projekt.srednia(y,z)        #Średnia Arytmetyczna
projekt.mediana(y,z)        #Mediana
projekt.odchylenie(y,z)     #Odchylenie Standardowe
projekt.funN(x[2],z[2])     #Funkcja Interpolacja Newtona
projekt.funl(x[2], z[2])    #Funkcja Interpolacja Lagrangea
projekt.funAl(x[2],z[2])    #Funkcja Aproksymacja Liniowa
projekt.funA(x[2], z[2])    #Funkcja Aproksymacja Średnio-Kwadratowa
# projekt.iniciuj(x[2],z[2]) #Funckcja odpowiedzialna za pokazywanie różnicy między funkcjami
Interpolacyjnymi
# projekt.iniciuj1(x[2],z[2]) #Funkcja odpowiedzialna za pokazywanie różnicy między funkcjami
Aproksymacyjnymi
projekt.pole(ma) #Liczy pole
projekt.calkiIiA(x,z)       #Liczy całki
projekt.poch1(x[2],z[2])    #liczy pochodną
projekt.monotonicznosc(projekt.poch(x[2],z[2])) #Sprawdza monotoniczność
```

```
i    def wykres2D(self, x, y, z):
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        surf = ax.plot_surface(x, y, z, cmap='plasma', edgecolor='none',vmin=-400,
vmax=1050)
        plt.title('Mapa 2D', fontdict={
                  'fontname': 'monospace', 'fontsize': 18})
        fig.colorbar(surf, shrink=0.5, aspect=5)
        ax.set_box_aspect([2, 1, 0.0001])
        ax.set_xlabel('X Label')
        ax.set_ylabel('Y Label')
        plt.show()
iidef wykres3D(self, x, y, z):

        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        surf = ax.plot_surface(x, y, z, cmap='plasma', edgecolor='none',vmin=-400,
vmax=1050)
        plt.title('Mapa 3D', fontdict={
                  'fontname': 'monospace', 'fontsize': 18})
        fig.colorbar(surf, shrink=0.5, aspect=5)
        ax.set_box_aspect([2, 1, 1])
        ax.set_xlabel('X Label')
        ax.set_ylabel('Y Label')
        ax.set_zlabel('Z Label')
        plt.show()
iiidef srednia(self,y,z):
      for i in range(11):
         print(round(y[i][i],1),"   ",np.mean(z[i]))


   def mediana(self, y, z):
      for i in range(11):
         print(round(y[i][i],1),"   ",np.median(z[i]))

   def odchylenie(self, y, z):
      for i in range(11):
         print(round(y[i][i],1),"   ",np.std(z[i]))
ivdef mianownik(self, x, k, n):
      m = 1
      for j in range(n):
         if j != k:
            m *= x[k]-x[j]
      return m

   def mianownik1(self, x, x1, k, n):
      m = 1
      for j in range(n):
         if j != k:
            m *= x1-x[j]
      return m

   def lan(self, x, y, iks, n):
```

```
        W = 0
        for j in range(n):
            W += (y[j]/(projekt.mianownik(x, j, n))) * projekt.mianownik1(x, iks, j, n)
        return W

    def funl(self, X, Y):
        plt.plot(X, Y, 'bo')
        X=np.round(X,2)
        x1 = [X[i:i+4] for i in range(0,len(X)-1,3)]
        y1 = [Y[i:i+4] for i in range(0,len(Y)-1,3)]
        for i in range(len(x1)):
            m=len(x1[i])-1
            print (m)
            iks = np.linspace(x1[i][0], x1[i][m], 101)
            plt.plot(iks, projekt.lan(x1[i], y1[i], iks, m+1), label=(f"{x1[i][0]}-{x1[i][m]}"))
        plt.legend()
        plt.title("Interpolacja Lagrange'a")
        plt.xlabel("oś X")
        plt.ylabel("oś Y")
        plt.show()
vdef roznica(self,x, y):

        n = len(y)
        coef = np.zeros([n, n])
        coef[:,0] = y
        for j in range(1, n):
            for i in range(n-j):
                coef[i][j] = \
                    (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j]-x[i])

        return coef


    def newton(self,coef, x_data, x):

        n = len(x_data) - 1
        p = coef[n]
        for k in range(1, n+1):
            p = coef[n-k] + (x - x_data[n-k])*p
        return p


    def funN(self,X, Y):
        X=np.round(X,2)
        x1 = [X[i:i+4] for i in range(0,len(X)-1,3)]
        y1 = [Y[i:i+4] for i in range(0,len(Y)-1,3)]
        for i in range(len(x1)):
            m=len(x1[i])-1
            x_new=np.linspace(x1[i][0],x1[i][m],101)
            y_new=projekt.newton(projekt.roznica(x1[i],y1[i])[0,:],x1[i],x_new)
            plt.plot(x_new, y_new,label=(f"{x1[i][0]}-{x1[i][m]}"))
```

```
        plt.plot(X, Y, 'bo')
        plt.legend()
        plt.title("Interpolacja Newtona")
        plt.xlabel("oś X")
        plt.ylabel("oś Y")
        plt.show()
videf iniciuj(self,X, Y):
        X=np.round(X,2)
        x1 = [X[i:i+4] for i in range(0,len(X)-1,3)]
        y1 = [Y[i:i+4] for i in range(0,len(Y)-1,3)]
        for i in range(len(x1)):
            m=len(x1[i])-1
            x_new=np.linspace(x1[i][0],x1[i][m],10)
            y_new=projekt.newton(projekt.roznica(x1[i],y1[i])[0,:],x1[i],x_new)
            print(projekt.lan(x1[i],y1[i],x_new,m+1)-y_new)
vii   def gauss(self,A,b):
        n=A.shape[0]
        C=np.zeros((n,n+1))
        C[:,0:n]=A
        C[:,n]=b
        x=np.zeros(n)
        for s in range(0, n-1):
            for i in range(s+1, n):
                # L = A[i,s] / A[s,s]
                for j in range(s+1, n+1):
                    C[i,j] = C[i,j] - (C[i,s] / C[s,s]) * C[s,j]
        x[n-1] = C[n-1,n]/C[n-1,n-1]
        for i in range(n-2,-1,-1):
            suma = 0.0
            for s in range(i+1, n):
                suma = suma + C[i,s] * x[s]
            x[i] = (C[i,n] - suma) / C[i,i]
        return x

def aproksymacja3(self, x, y):
        M1 = np.zeros((3, 3))
        P1 = np.zeros(3)
        n = 3
        M1[0, 0] = n
        for i in range(n):
            M1[0, 1] = M1[0, 1]+x[i]
            M1[0, 2] = M1[0, 2]+x[i]**2
            M1[1, 0] = M1[1, 0]+x[i]
            M1[1, 1] = M1[1, 1]+x[i]**2
            M1[1, 2] = M1[1, 2]+x[i]**3
            M1[2, 0] = M1[2, 0]+x[i]**2
            M1[2, 1] = M1[2, 1]+x[i]**3
            M1[2, 2] = M1[2, 2]+x[i]**4

        for i in range(n):
            P1[0] += y[i]
            P1[1] += x[i]*y[i]
            P1[2] += (x[i]**2)*y[i]

        K1 = projekt.gauss(M1, P1)

        return K1

    def f1(self, a0, a1, a2, x):
```

```
        return a0+a1*x+a2*x**2
viii  def gauss(self,A,b):
        n=A.shape[0]
        C=np.zeros((n,n+1))
        C[:,0:n]=A
        C[:,n]=b
        x=np.zeros(n)
        for s in range(0, n-1):
            for i in range(s+1, n):
                # L = A[i,s] / A[s,s]
                for j in range(s+1, n+1):
                    C[i,j] = C[i,j] - (C[i,s] / C[s,s]) * C[s,j]
        x[n-1] = C[n-1,n]/C[n-1,n-1]
        for i in range(n-2,-1,-1):
            suma = 0.0
            for s in range(i+1, n):
                suma = suma + C[i,s] * x[s]
            x[i] = (C[i,n] - suma) / C[i,i]
        return x
def aproksymacja2(self,x,y):
        M=np.zeros((2,2))
        P=np.zeros (2)
        n=2
        M[0][0]=n
        for i in range (n):
            M[0][1]=M[1][0]+x[i]
            M[1][0]=M[1][0]+x[i]
            M[1][1]=M[1][1]+x[i]**2
        for i in range (n):
            P[0]=P[0]+y[i]
            P[1]=P[1]+x[i]*y[i]
        K=projekt.gauss(M,P)
        return K
    def f0(self,a0,a1,x):
        return a0+a1*x
ix        def iniciuj1(self,X, Y):
        X=np.round(X,2)
        x1 = [X[i:i+3] for i in range(0,len(X)-1,2)]
        y1 = [Y[i:i+3] for i in range(0,len(Y)-1,2)]
        m=len(x1[1])-1
        K1 = projekt.aproksymacja3(x1[0], y1[0])
        iks = np.linspace(x1[0][0], x1[0][m], 66)
        W1=projekt.f1(K1[0], K1[1], K1[2], iks)
        X=np.round(X,2)
        x1 = [X[i:i+2] for i in range(0,len(X)-1,1)]
        y1 = [Y[i:i+2] for i in range(0,len(Y)-1,1)]
        m=len(x1[1])-1
        K1 = projekt.aproksymacja2(x1[0], y1[0])
        iks = np.linspace(x1[0][0], x1[0][m], 44)
        W2=projekt.f0(K1[0], K1[1],iks)
        print(W1[0:44],"\n",W2)
        print(W1[0:44]-W2)
x     def pole(sefl,ma):
        tri = Delaunay(ma[:, :2])
        triangle_areas = []
        for simplex in tri.simplices:
            p0, p1, p2 = ma[simplex]
            triangle_areas.append(
                0.5 * np.linalg.norm(
                    np.cross(p1 - p0, p2 - p0)
                )
            )
        surface_area = np.sum(triangle_areas)
```

```
        print("Pole powierzchni: ", surface_area)
xi    def calkSa(self, K1):
        cp = 0
        xs = 0
        ys = 0
        n=10
        a = 0
        b = 2
        x=np.zeros(n+1)
        y=np.zeros(n+1)
        h=(b-a)/n
        for i in range(n+1):
            x[i]=a+i*h
            y[i]=(projekt.f1(K1[0], K1[1], K1[2],x[i]))
            h = (b-a)/n
        for i in range(n):
            xs = (x[i]+x[i+1])/2
            ys = (projekt.f1(K1[0], K1[1], K1[2],xs))
            cp += h*((y[i]+y[i+1]+4*ys)/6)
        return cp
    def calkSa1(self, x3,y3):
        cp = 0
        xs = 0
        ys = 0
        n=10
        a = 0
        b = 2
        x=np.zeros(n+1)
        y=np.zeros(n+1)
        h=(b-a)/n
        for i in range(n+1):
            x[i]=a+i*h
            y[i]=(projekt.lan(x3,y3,x[i],3))
            h = (b-a)/n
        for i in range(n):
            xs = (x[i]+x[i+1])/2
            ys = (projekt.lan(x3,y3,xs,3))
            cp += h*((y[i]+y[i+1]+4*ys)/6)
        return cp

x1 = [x[2][i:i+3] for i in range(0,len(x[0])-1,2)]
y1 = [z[2][i:i+3] for i in range(0,len(z[0])-1,2)]


print (x1,"  ",y1)

X=sp.symbols('x')

K1=projekt.aproksymacja3(x1[2],y1[2])
a=0
b=2

cd = sp.integrate(projekt.f1(K1[0],K1[1],K1[2],X),(X,a,b))
sa=projekt.calkSa(K1)
print ("Całka Dokładna: ",cd," Całka z Aproksymacji średnio-kwadratowej: ",sa)

iks=np.linspace(x1[2][0],x1[2][2],101)

cd1 = sp.integrate(projekt.lan(x1[2],y1[2],X,3),(X,a,b))
sa1=projekt.calkSa1(x1[2],y1[2])
print ("Całka Dokładna: ",cd1," Całka z interpolacij Lagrange'a: ",sa1)
```

```
xii   def monotonicznosc(self,z):
        n = len(x)

        rosnie = all((z[i]) <= (z[i+1]) for i in range(n-1))
        maleje = all((z[i]) >= (z[i+1]) for i in range(n-1))

        if rosnie:
           print("Funkcja jest rosnąca dla każdego punktu.")
        elif maleje:
           print("Funkcja jest malejąca dla każdego punktu.")
        else:
           print("Funkcja nie jest monotoniczna dla każdego punktu.")
```