WYDZIAŁ INŻYNIERII MECHANICZNEJ I INFORMATYKI

Projekt zaliczeniowy
Metody Numeryczne

Autor: **Rafał Mastalerz**

Rok akademicki 2021/2022

Informatyka, semestr II
Tryb stacjonarny
Nr indeksu: 135018

Spis treści

Rozdział 1

WSTĘP

Zadanie polega na wykonaniu wielostopniowej analizy punktów w przestrzeni 3D na płaszczyźnie kartezjańskiej. Podzielone zostało na kilka modułów, z którego każdy pozwala na uzyskanie kolejnych informacji na temat danych poddanych analizie. W pracy zostały zastosowane odpowiednio dobrane metody numeryczne w celu optymalnej oceny wyników przykładowych pomiarów z pliku **135018.dat**

W projekcie wykorzystano między innymi interpolację Czebyszewa, aproksymację metodą najmniejszych kwadratów, całkowanie numeryczne metodą trapezów. Do obliczenia pola uzyskanej figury skorzystano ze wzoru Herona na pole trójkąta.

Zarówno do wizualizacji, jak i do obliczeń numerycznych, wykorzystano oprogramowanie **Octave** na licencji GNU General Public License (GPL). Skrypty tworzone oraz uruchamiane były w środowisku **Linux LUbuntu**. Niniejszy dokument wygenerowany został za pomocą oprogramowania LaTeX.

Rozdział 2

WIZUALIZACJA DANYCH

2.1 Wizualizacja danych w formie mapy 2D

Pierwsze zadanie dotyczy wizualizacji zadanej chmury punktów na mapie 2D z określeniem kolorów dla osi Z.
W celu wykonania mapy na podstawie pliku wejściowego **135018.dat** utworzone zostały następujące macierze:

X(1x20)	**Y(1x20)**	**Z(20x20)**
Współrzędne osi X	Współrzędne osi Y	Współrzędne osi Z

Kod źródłowy 2.1: main.m (fragment 1)

```
A = dlmread('135018.dat', ' '); % Pobieranie danych z pliku

% Tworzenie macierzy wspolrzednych X i Y
for i=1:20
  X(i)=Y(i)=A(i,2);
endfor

% Tworzenie macierzy wspolrzednych Z
k=1;
h=1;
for i=1:rows(A)
  Z(k,h)=A(i,3);
  k+=1;
  if (k==21)
    k=1;
    h+=1;
  endif
endfor

clear A i k h; % Czyszczenie niepotrzebnej macierzy A oraz zmiennych i,
    k, h
```

Powyższy kod źródłowy jest odpowiedzialny za wczytanie danych z pliku **135018.dat** do macierzy A, która następnie jest dzielona na podmacierze.

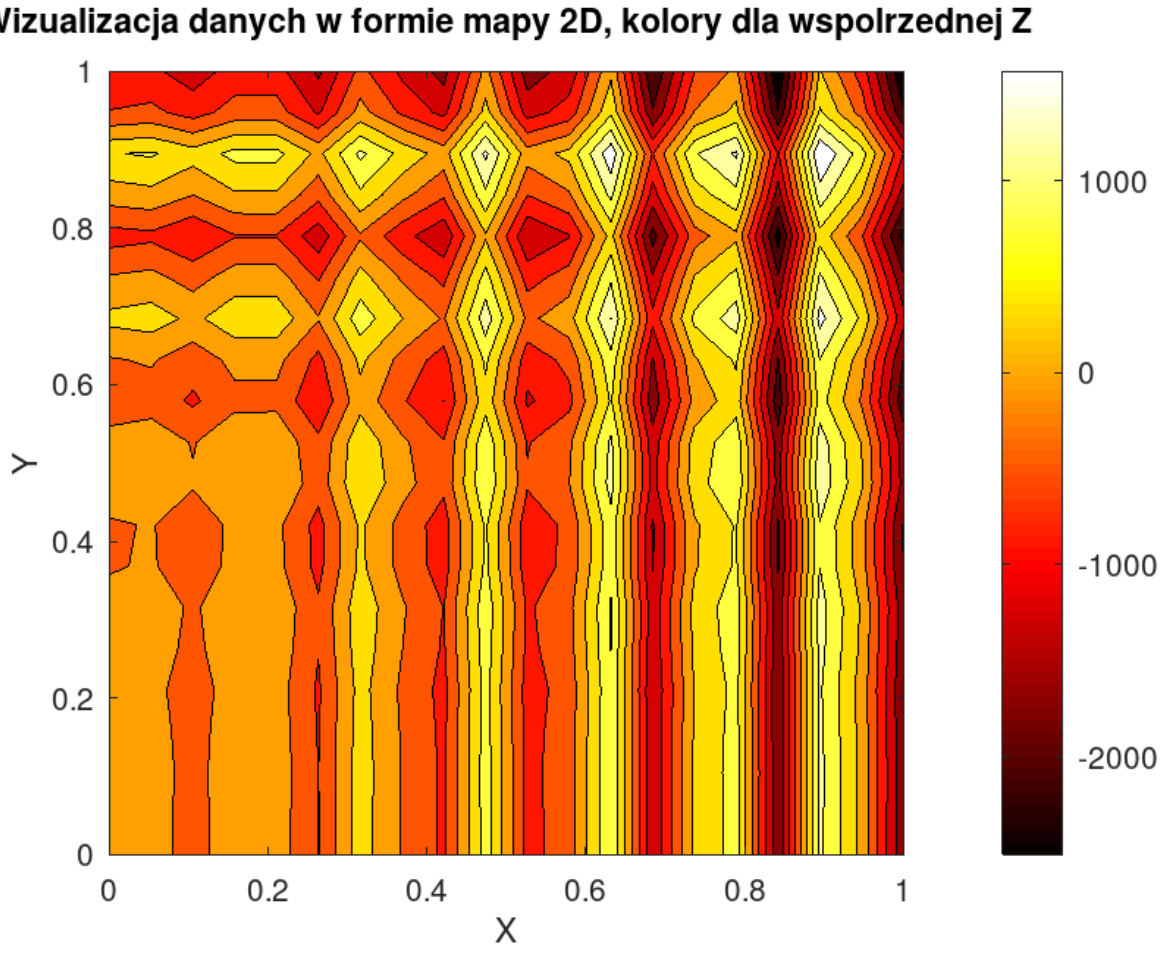
Macierze X i Y są w zasadzie tymi samymi tablicami, gdyż współrzędne są symetryczne względem siebie. Rozmiar macierzy to 1x20.

Macierz Z ma strukturę 20x20, każda kolumna odpowiada za poszczególny wiersz X.

Do utworzenia mapy 2D wykorzystany został następujący kod:

Kod źródłowy 2.2: main.m (fragment 2)

```
figure(1); % Numer wykresu
contourf(X,Y,Z); % Linijka odpowiada za rysowanie mapy
title("Wizualizacja danych w formie mapy 2D, kolory dla wspolrzednej Z"
    );
xlabel("X"); % Etykieta osi X
ylabel("Y"); % Etykieta osi Y
colorbar; % Wyswietlanie paska kolorow (legenda)
colormap('hot'); % Wybrany schemat kolorow
```



Z uzyskanej mapy możliwe jest odczytanie wartości Z na podstawie kolorów widocznych w legendzie po prawej stronie rysunku. Na biało oznaczone są wartości maksymalne, a na czarno wartości minimalne.

Na podstawie takiego wykresu trudno jednak zorientować się w dokładnym rozkładzie punktów, ze względu na niską precyzję obrazka. Dużo lepszy ogląd sytuacji uzyskamy stosując wykres 3D.

2.2 Wizualizacja danych w formie mapy 3D

Mapa 3D jest znacznie dokładniejszą metodą prezentacji współrzędnych trójwymiarowych. Możliwość wykonania obrotu wygenerowanego wielokąta pozwala znacznie łatwiej zobrazować sobie analizowaną powierzchnię.

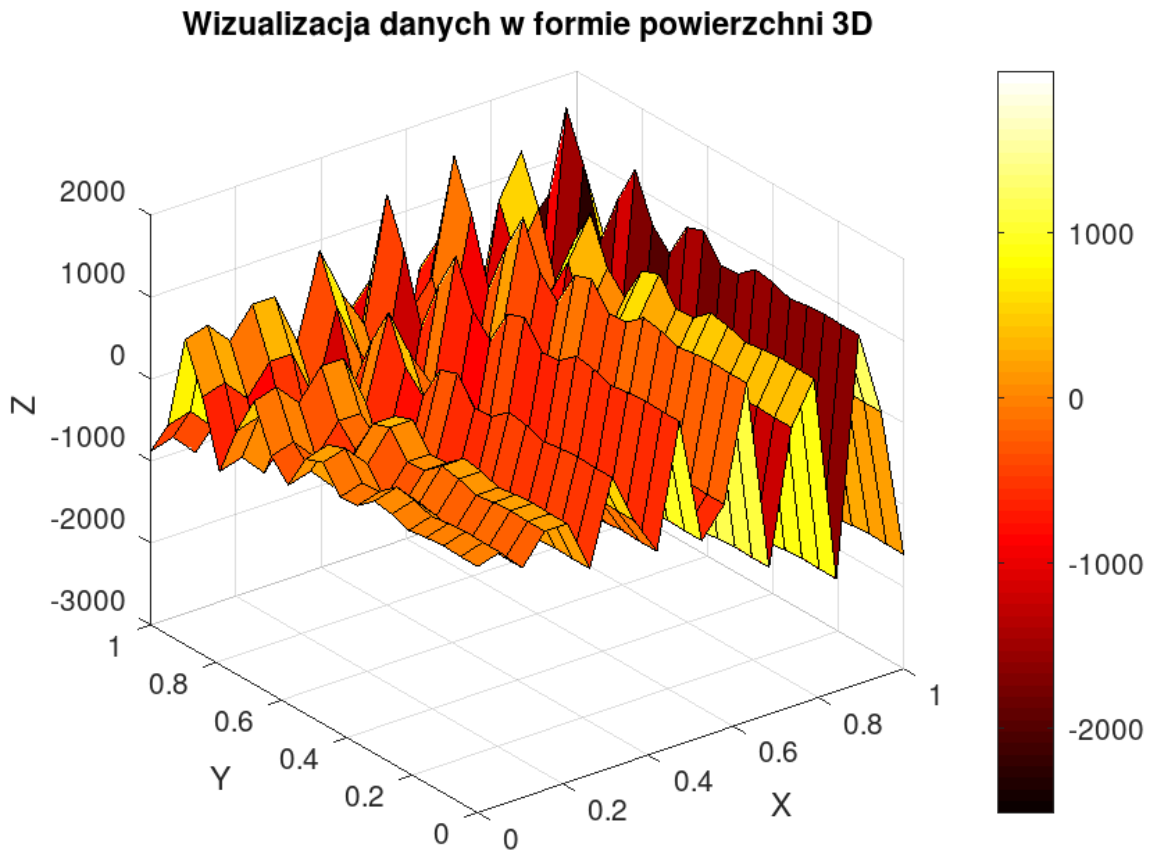
Obracanie mapy 3D pozwala również w znacznie łatwiejszy sposób zorientować się w monotoniczności punktów oraz w przebiegu hipotetycznych pomiarów.

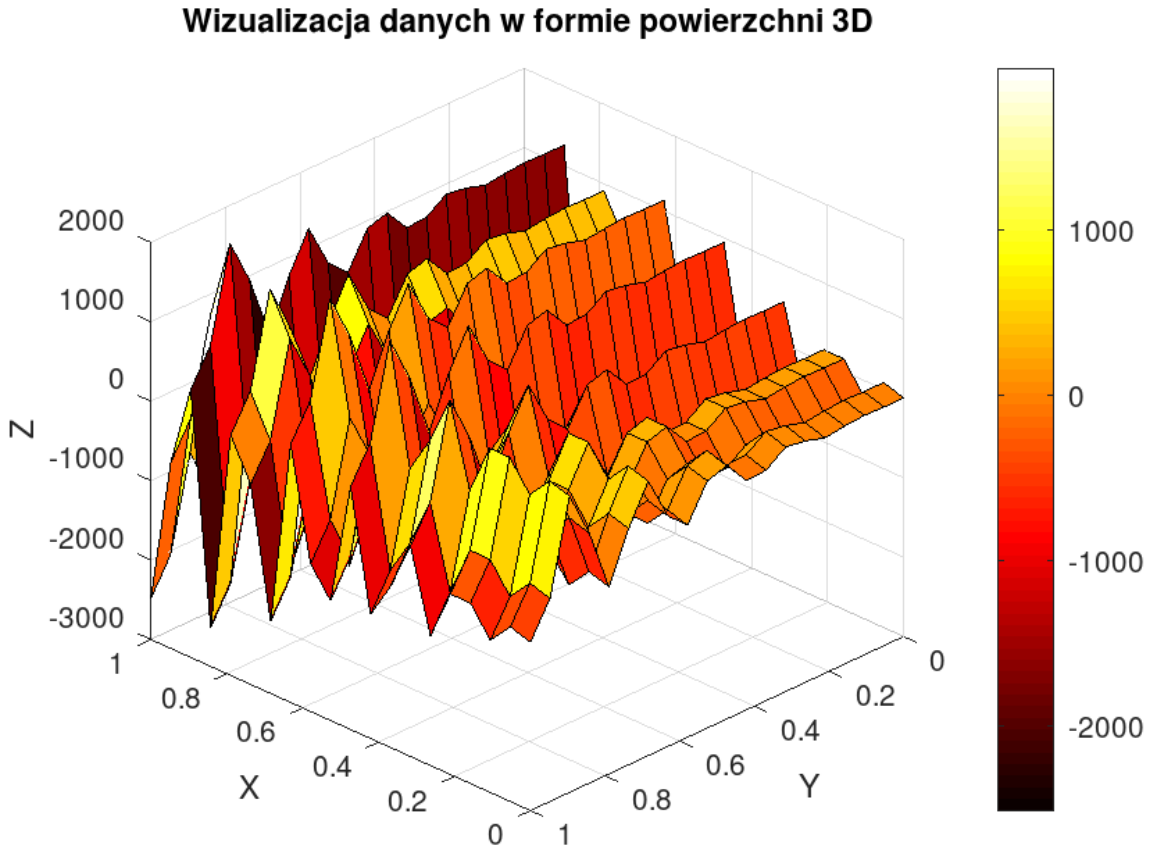
Oprogramowanie Octave umożliwia wygenerowanie wielu rodzajów wizualizacji.

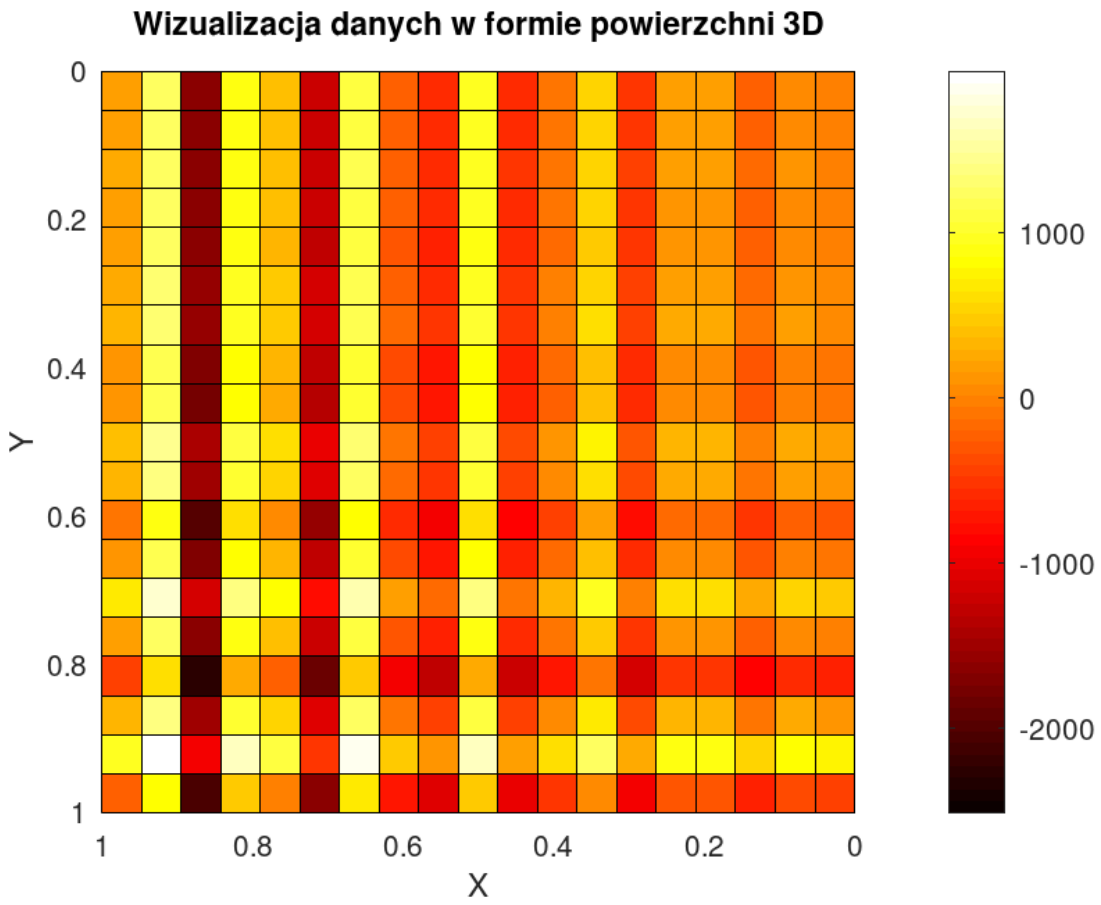
Za wygenerowanie powierzchni 3D odpowiedzialny jest następujący fragment kodu (do obliczeń ciągle wykorzystywane są wcześniej wspomniane macierze X,Y i Z):

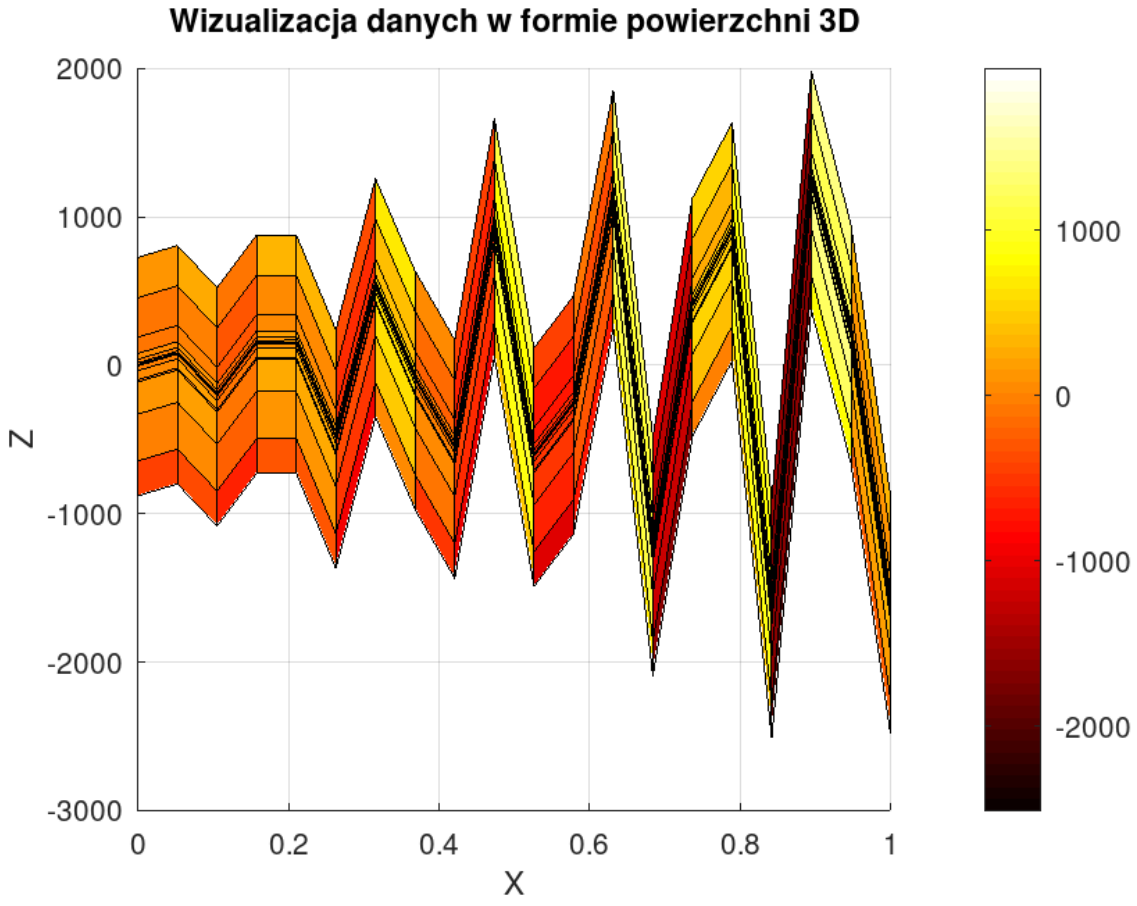
Kod źródłowy 2.3: main.m (fragment 3)

```
figure(2); % Numer wykresu
surf(X,Y,Z); % Funkcja rysujaca wykres powierzchni oparty na
   prostokatach
title("Wizualizacja danych w formie powierzchni 3D");
colorbar; % Wyswietlanie paska kolorow (legenda)
xlabel("X"); % Etykieta osi X
ylabel("Y"); % Etykieta osi Y
zlabel("Z"); % Etykieta osi Z
colormap('hot'); % Schemat kolorow
```


Wizualizacja danych w formie powierzchni 3D
2000
1000
0
-1000
-2000
-3000
Z
1
0.8
0.6
0.4
0.2
0
Y
0
0.2
0.4
0.6
0.8
1
X
1000
0
-1000
-2000


Wizualizacja danych w formie powierzchni 3D
2000
1000
0
-1000
-2000
-3000
Z
1
0.8
0.6
0.4
0.2
0
X
0
0.2
0.4
0.6
0.8
1
Y
1000
0
-1000
-2000


Wizualizacja danych w formie powierzchni 3D
0
0.2
0.4
0.6
0.8
1
Y
1
0.8
0.6
0.4
0.2
0
X
1000
0
-1000
-2000


Wizualizacja danych w formie powierzchni 3D
2000
1000
0
-1000
-2000
-3000
Z
0
0.2
0.4
0.6
0.8
1
X
1000
0
-1000
-2000

Rozdział 3

WYZNACZANIE WSPÓŁRZĘDNYCH DO ANALIZY

Wybór współrzędnych do obliczeń jest niezwykle ważną czynnością w sytuacji, kiedy nie mamy możliwości lub czasu na analizę całościową danych. Kluczowe w tym przypadku jest określenie kryterium, według którego dokonane zostanie większość obliczeń takich jak interpolacja, aproksymacja lub całki.

W przypadku niniejszej analizy kryterium wyboru jest wiersz z największym odchyleniem standardowym. W tym celu wyznaczone zostały dodatkowe parametry: średnia, mediana i wspomniane wcześniej odchylenie standardowe.

3.1 Średnia arytmetyczna

Średnia arytmetyczna (w języku potocznym określana po prostu jako średnia) to suma liczb podzielona przez ich liczbę. Wzór ogólny parametru wygląda następująco:

$$M = \frac{a_1 + a_2 + \cdots + a_n}{n} \tag{3.1}$$

Średnia arytmetyczna stanowi szczególny przypadek średniej potęgowej rzędu 1.

Średnia arytmetyczna jest podatna na obserwacje odstające i skośność rozkładu. Inne średnie, takie jak na przykład mediana, mogą dawać bardziej wiarygodne wyniki.

W środowisku **Octave** dostępna jest funkcja *mean*, która jest zdefiniowana jako: [1]

Kod źródłowy 3.1: Funkcja wbudowana mean

```
mean (x) = SUM_i x(i) / N
```

W analizie użyto natomiast matematycznej metody obliczania tego wskaźnika:

Kod źródłowy 3.2: srednia_f.m

```
function [srednia]=srednia_f(Z,wys=0)
  srednia=zeros(rows(Z),1);
    for j=1:columns(Z)
      suma=0;
      for i=1:rows(Z)
        suma+=Z(i,j);
      endfor
    srednia(j)=suma/rows(Z);
      if(wys==1)
        printf("Wiersz X nr: %d -  rednia  arytmetyczna: %f\n",j,
            srednia(j));
      endif
    endfor
     if(wys==1)
        printf("\n");
      endif
endfunction
```

Natomiast wywołanie funkcji wygląda następująco:

Kod źródłowy 3.3: main.m (fragment 4)

```
srednia=srednia_f(Z,1);
```

Funkcja zwraca macierz *srednia*(20x1), w której każdy wiersz odpowiada jednej kolumnie macierzy Z:

```
Wiersz X nr: 1 - Średnia arytmetyczna: -45.311468
Wiersz X nr: 2 - Średnia arytmetyczna: 34.587000
Wiersz X nr: 3 - Średnia arytmetyczna: -248.404665
Wiersz X nr: 4 - Średnia arytmetyczna: 106.332880
Wiersz X nr: 5 - Średnia arytmetyczna: 106.915875
Wiersz X nr: 6 - Średnia arytmetyczna: -534.536630
Wiersz X nr: 7 - Średnia arytmetyczna: 485.083105
Wiersz X nr: 8 - Średnia arytmetyczna: -137.952150
Wiersz X nr: 9 - Średnia arytmetyczna: -601.124500
Wiersz X nr: 10 - Średnia arytmetyczna: 892.553420
Wiersz X nr: 11 - Średnia arytmetyczna: -655.811500
Wiersz X nr: 12 - Średnia arytmetyczna: -303.841425
Wiersz X nr: 13 - Średnia arytmetyczna: 1083.438250
Wiersz X nr: 14 - Średnia arytmetyczna: -1258.689300
Wiersz X nr: 15 - Średnia arytmetyczna: 355.063820
Wiersz X nr: 16 - Średnia arytmetyczna: 863.184665
Wiersz X nr: 17 - Średnia arytmetyczna: -1676.313850
Wiersz X nr: 18 - Średnia arytmetyczna: 1209.465850
Wiersz X nr: 19 - Średnia arytmetyczna: 173.921750
Wiersz X nr: 20 - Średnia arytmetyczna: -1653.443800
```

Rysunek 3.1: Wynik działania skryptu srednia_f.m

3.2 Mediana

Mediana, zwana *drugim kwantylem* lub *wartością środkową*, to wartość cechy w uporządkowanym szeregu, poniżej i powyżej której znajduje się jednakowa liczba obserwacji wyników. Mediana stanowi kwantyl rzędu 1/2, czyli jest drugim kwartylem. Można powiedzieć również, że jest trzecim kwantylem szóstego rzędu, piątym decylem itp.

Mediana ma interpretację jako optymalne przewidywanie wartości za pomocą jednej liczby, gdy przyjętą funkcją błędu przewidywania jest moduł różnicy (odchylenia).

Aby obliczyć medianę ze zbioru n obserwacji, sortujemy wyniki w kolejności rosnącej i numerujemy od 1 do n. W następnym kroku, jeśli n jest nieparzyste, medianą jest wartość obserwacji w środku (czyli obserwacji numer $\frac{n+1}{2}$).

Jeśli natomiast n jest parzyste, wynikiem jest średnia arytmetyczna między dwiema środkowymi obserwacjami, czyli obserwacją numer n i obserwacją numer $\frac{n}{2}+1$.

Poniżej implementacja w Octave:

Kod źródłowy 3.4: mediana_f.m

```
function med = mediana_f(Z,wys=0)
    med=zeros(columns(Z),1);
    Zm=sort(Z);
    for i=1:columns(Z)
      if(mod(columns(Z),2)==0)
        med(i,1)=(Zm((columns(Z)/2),i)+Zm(((columns(Z)/2)+1),i))/2;
      else
        med(i,1)=Zm((columns(Z)/2),i);
      endif
      if(wys==1)
        printf("Wiersz X nr: %d - mediana: %f\n",i, med(i));
      endif
    endfor
      if(wys==1)
        printf("\n");
      endif
endfunction
```

Wywołanie funkcji wygląda następująco:

Kod źródłowy 3.5: main.m (fragment 5)

```
mediana=mediana_f(Z,1);
```

```
Wiersz X nr: 1 - mediana: -0.018520
Wiersz X nr: 2 - mediana: 79.879950
Wiersz X nr: 3 - mediana: -203.111500
Wiersz X nr: 4 - mediana: 151.625500
Wiersz X nr: 5 - mediana: 152.208500
Wiersz X nr: 6 - mediana: -489.244000
Wiersz X nr: 7 - mediana: 530.376500
Wiersz X nr: 8 - mediana: -92.659100
Wiersz X nr: 9 - mediana: -555.831500
Wiersz X nr: 10 - mediana: 937.846500
Wiersz X nr: 11 - mediana: -610.518500
Wiersz X nr: 12 - mediana: -258.548500
Wiersz X nr: 13 - mediana: 1128.730000
Wiersz X nr: 14 - mediana: -1213.395000
Wiersz X nr: 15 - mediana: 400.356500
Wiersz X nr: 16 - mediana: 908.477500
Wiersz X nr: 17 - mediana: -1631.020000
Wiersz X nr: 18 - mediana: 1254.760000
Wiersz X nr: 19 - mediana: 219.214500
Wiersz X nr: 20 - mediana: -1608.150000
```

Rysunek 3.2: Wynik działania skryptu mediana_f.m

3.3 Odchylenie Standardowe

Jest to klasyczna miara zmienności. Odchylenie standardowe jest pierwiastkiem kwadratowym z wariancji. Kierując się intuicją, odchylenie standardowe stanowi informację o tym, jak szeroko wartości danej wielkości są rozrzucone wokół jej średniej. Im mniejsza jest wartość odchylenia, tym obserwacje są bardziej skupione wokół średniej.

Funkcja obliczająca wartość odchylenia standardowego to:

Kod źródłowy 3.6: odchylenie_f.m

```
function S = odchylenie_f(Z,wys=0)
    SR=srednia_f(Z);
    [w,k] = size(Z);
    S=zeros(k,2);
  for i=1:k
    suma=0;
    for j=1:w
     suma=suma+((Z(j,i)-SR(i,1))^2);
    endfor
  S(i,1) = (suma/w);
  S(i,2) = sqrt(suma/w);
      if(wys==1)
        printf("Wiersz X nr: %d - odchylenie standardowe: %f\n",i, S(i,
           2));
      endif
  endfor
       if(wys==1)
        printf("\n");
      endif
endfunction
```

Funkcja wykorzystuje wcześniej wymienioną funkcję z kodu źródłowego 3.2 (srednia_f.m).

Wywołanie funkcji jest następujące:

Kod źródłowy 3.7: main.m (fragment 6)

```
odchylenie=odchylenie_f(Z,1);
[a,b]=max(odchylenie(:,2));
printf("Maksymalne odchylenie standardowe to %f i zachodzi w %d linijce
   \n",a,b);
```

```
Wiersz X nr: 1 - odchylenie standardowe: 339.166871
Wiersz X nr: 2 - odchylenie standardowe: 339.166821
Wiersz X nr: 3 - odchylenie standardowe: 339.166733
Wiersz X nr: 4 - odchylenie standardowe: 339.166905
Wiersz X nr: 5 - odchylenie standardowe: 339.166906
Wiersz X nr: 6 - odchylenie standardowe: 339.166175
Wiersz X nr: 7 - odchylenie standardowe: 339.166669
Wiersz X nr: 8 - odchylenie standardowe: 339.166891
Wiersz X nr: 9 - odchylenie standardowe: 339.166535
Wiersz X nr: 10 - odchylenie standardowe: 339.166930
Wiersz X nr: 11 - odchylenie standardowe: 339.167574
Wiersz X nr: 12 - odchylenie standardowe: 339.167211
Wiersz X nr: 13 - odchylenie standardowe: 339.167339
Wiersz X nr: 14 - odchylenie standardowe: 339.166666
Wiersz X nr: 15 - odchylenie standardowe: 339.166579
Wiersz X nr: 16 - odchylenie standardowe: 339.166536
Wiersz X nr: 17 - odchylenie standardowe: 339.166509
Wiersz X nr: 18 - odchylenie standardowe: 339.166374
Wiersz X nr: 19 - odchylenie standardowe: 339.166859
Wiersz X nr: 20 - odchylenie standardowe: 339.166622
```

Rysunek 3.3: Wynik działania skryptu odchylenie_f.m

Powyższe instrukcje pozwalają na wytypowanie wiersza X, nad którym prowadzę obliczenia. Z obliczeń wynika, iż tym wierszem jest:

```
Maksymalne odchylenie standardowe to 339.167574 i zachodzi w 11 linijce
```

Rysunek 3.4: Wybrany wiersz do dalszych obliczeń

Aby wyodrębnić tę linię, został wykorzystany kod:

Kod źródłowy 3.8: main.m (fragment 7)

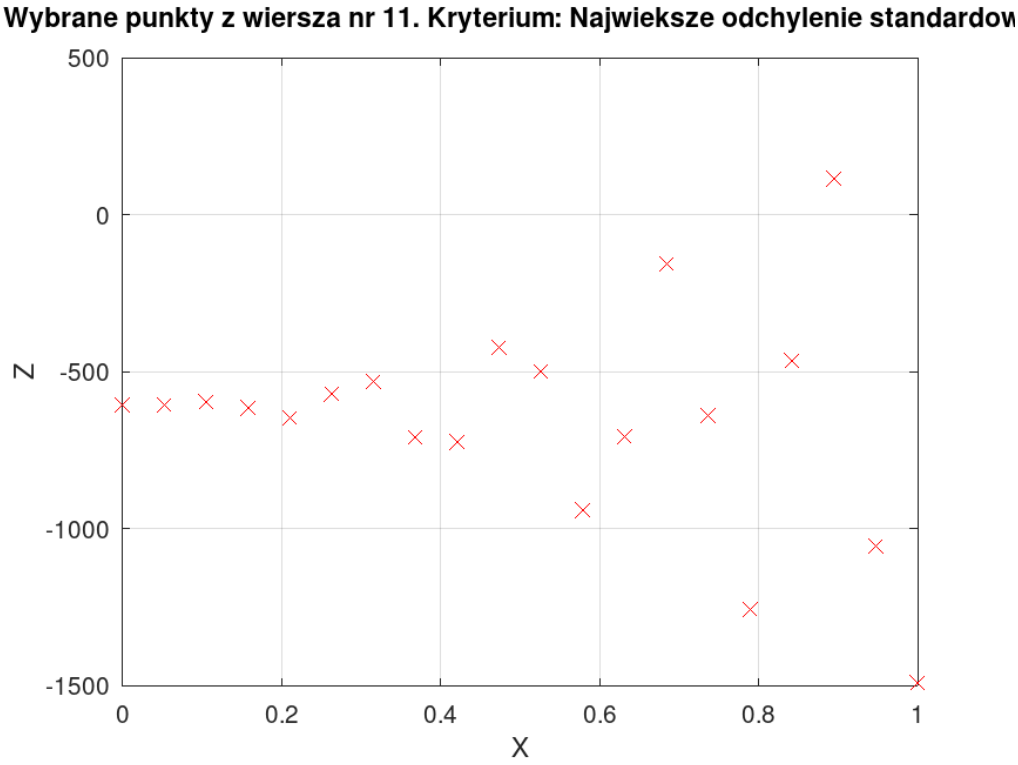
```
1 Z11=Z(:,b); % 11 linijka z osi Z
```

Kod ten spowodował utworzenie macierzy Z11(20x1) o wartościach:

```
Z11 =

  -6.060000000000000e+02
  -6.056400000000000e+02
  -5.952900000000000e+02
  -6.150370000000000e+02
  -6.475450000000000e+02
  -5.708890000000000e+02
  -5.315130000000000e+02
  -7.087290000000000e+02
  -7.230000000000000e+02
  -4.221730000000000e+02
  -4.991950000000000e+02
  -9.402070000000000e+02
  -7.064130000000000e+02
  -1.560470000000000e+02
  -6.383910000000000e+02
  -1.255610000000000e+03
  -4.639780000000000e+02
   1.153370000000000e+02
  -1.055310000000000e+03
  -1.490600000000000e+03
```

Rysunek 3.5: Macierz Z11

Macierz ta jest następnie używana w dalszych etapach projektu. Poniżej zaznaczone zostały analizowane punkty.



Rozdział 4

ANALIZA SZCZEGÓŁOWA X NUMER 11

4.1 Informacje wstępne

Do analizy punktów zostaną wykorzystane wybrane algorytmy z dziedziny metod numerycznych przewidziane do tego celu. Kluczowymi elementami do osiągnięcia efektów będą **interpolacja** oraz **aproksymacja**.

Wymienione metody pozwalają na utworzenie wzorów wielomianów przybliżających jak najwierniej oryginalną funkcję, stojącą za punktami dyskretnymi pobranymi w trakcie wykonywanych pomiarów.

4.2 Interpolacja Czebyszewa

W analizie numerycznej węzły Czebyszewa są pewnymi rzeczywistymi liczbami algebraicznymi, a więc pierwiastkami wielomianów Czebyszewa pierwszego rodzaju. Często są używane jako węzły w interpolacji wielomianowej, gdyż uzyskiwany dzięki tej metodzie wielomian interpolacyjny minimalizuje efekt Rungego (duże oscylacje wielomianu interpolacyjnego przy krańcach przedziału). Jest to jedna z przyczyn, dla których wybrałem tę metodę. Miejsca zerowe wielomianów Czebyszewa zagęszczają się ku krańcom przedziału, pozwala to

lepiej związać wielomian zapobiegając naturalnym dla wielomianów wysokiego rzędu oscylacjom.

W niniejszej interpolacji wartości argumentów powinny być znormalizowane do przedziału $[-1,1]$. Normalizacji dokonuje się na podstawie wzoru [3]:

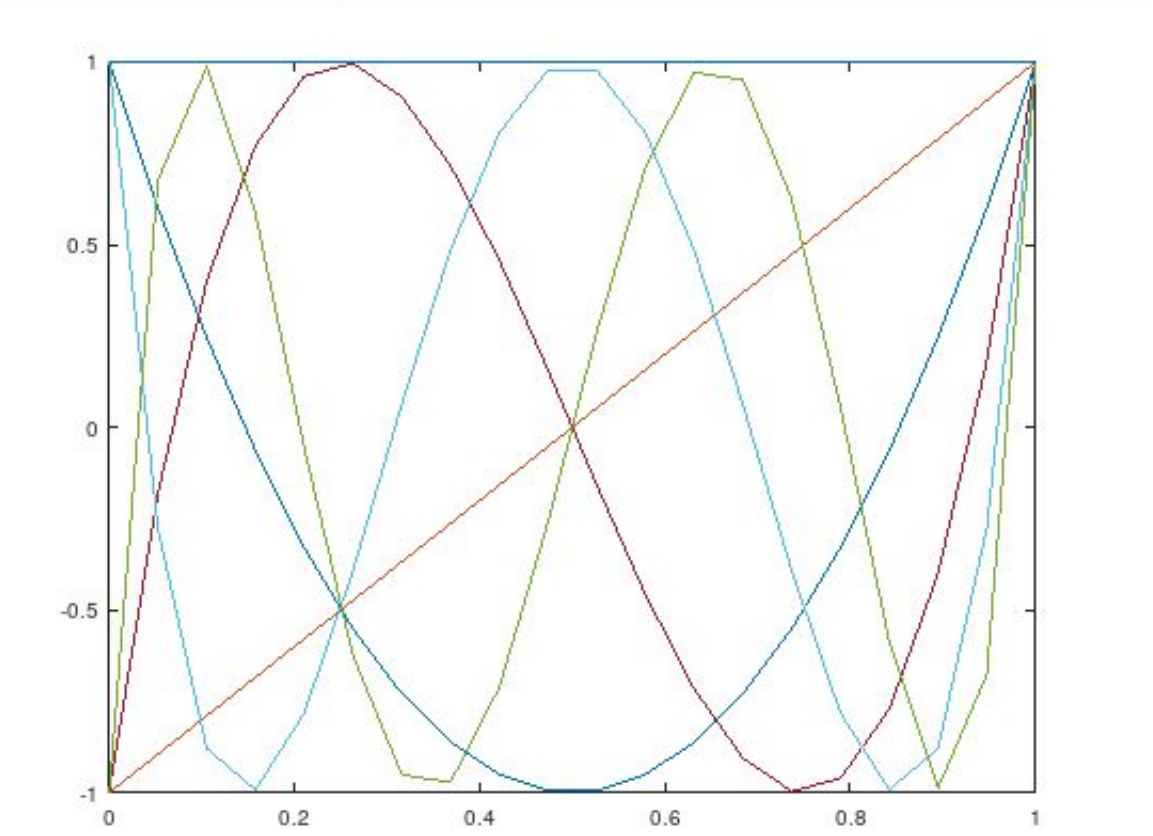
$$x = x(\xi), x = \frac{a+b}{2} + \frac{b-a}{2}\xi \tag{4.1}$$

Funkcje bazowe wielomianu Czebyszewa wyznaczane są rekurencyjnie $T_0(\xi), T_1(\xi), \ldots, T_k(\xi), \ldots, T_n(\xi)$

Wzór na wyznaczenie kolejnych wielomianów jest następujący:

$$\begin{aligned} T_{k+1}(\xi) &= 2\xi T_k(\xi) - T_{k-1}(\xi) \\ T_0(\xi) &= 1 \quad T_1(\xi) = \xi \end{aligned} \tag{4.2}$$

Poniżej zamieszczony został wykres dla $n = 5$



Współczynniki dla wzoru wielomianu interpolacyjnego wyznaczane są z ogólnego równania interpolacji wielomianowej:

$$\begin{bmatrix} T_0(\xi_0) & T_1(\xi_0) & \cdots & T_k(\xi_0) & \cdots & T_n(\xi_n) \\ T_0(\xi_1) & T_1(\xi_1) & \cdots & T_k(\xi_1) & \cdots & T_n(\xi_n) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ T_0(\xi_k) & T_1(\xi_k) & \cdots & T_k(\xi_k) & \cdots & T_n(\xi_k) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ T_0(\xi_n) & T_1(\xi_n) & \cdots & T_k(\xi_n) & \cdots & T_n(\xi_n) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_k \\ \vdots \\ f_n \end{bmatrix} \tag{4.3}$$

Kod *Octave* generujący rekurencyjnie funkcje bazowe wielomianu Czebyszewa zamieszczony został poniżej:

Kod źródłowy 4.1: T.m

```
function y = T(n,x)
% n - liczba wielomianu
% x - macierz wejsciowa xi

m = length(x);
if n == 0
    y = ones(1,m);
else
    if n == 1;
        y = x;
    else
        y = 2 * x .* T(n-1,x)-T(n-2,x);
    endif
endif
endfunction
```

Funkcja realizująca interpolację Czebyszewa wykorzystuje powyższe instrukcje do obliczeń. Jej kod źródłowy wygląda następująco:

Kod źródłowy 4.2: czebyszew_interpolacja_f.m

```
function czebyszew_interpolacja_f(X,Y,Z11)
% X - macierz dla osi X
% Y - macierz dla osi Z
% Z11 - macierz dla punktow Z na wykresie (nie uczestniczy w
    obliczeniach

Y=trans(Z11); % Transponowanie macierzy Y
Xi=zeros(size(X)); % Zerowanie macierzy Xi

for i=1:columns(X)
  Xi(i)=(X(i)-0.5)/0.5; % Normalizacja wspolrzednych X
endfor

```

```
13 W=zeros(columns(X)); % Macierz W wspolczynnikow
14
15 for i=1:columns(X)
16   for j=1:columns(X)
17     W(i,j)=T(j-1,Xi(i)); % Wypelnianie macierzy wspolczynnikow
           odpowiednimi danymi
18   endfor
19 endfor
20
21 CZEBY=gaussinv(W)*trans(Y); % Rozwiazywanie rownania za pomoca funkcji
     gaussinv oraz trans
22
23 printf("Wielomian interpolacyjny Czebyszewa: \n"); % Linijki wypisujace
     na ekran wielomian interpolacyjny
24 printf("%f + " ,CZEBY(1,1));
25 for i = 1:columns(X)-2
26   printf("%f * T(%d,x) + ", CZEBY(i+1,1),i);
27 endfor
28 printf("%f * T(%d,x)\n",CZEBY(columns(X),1),columns(X)-1);
29
30 wielomian = CZEBY(1,1);
31 for i=1:columns(X)-1
32    wielomian+=CZEBY(i+1,1) * T(i,Xi);
33    cze(i)=CZEBY(i+1,1);
34    wsp=T(i,Xi);
35 endfor
36 % Czesc graficzna
37 plot(Xi,wielomian);
38 title("Interpolacja Czebyszewa")
39 hold on;
40 grid on;
41 xlabel("Y znormalizowane");
42 ylabel("Z11");
43 plot(Xi,Z11,"og");
44
45 endfunction
```

Jak można zauważyć, funkcja wykorzystuje transponowanie oraz odwracanie macierzy w postaci funkcji *trans* oraz *gaussinv*. Poniżej zamieszczone zostały kody źródłowe wymienionych metod:

Kod źródłowy 4.3: trans.m

```
1 function T = trans(C)
2     for i=1:rows(C)
3      for j=1:columns(C)
4        T(j,i)=C(i,j);
5      end
6    end
7 endfunction
```

trans jest elementarną funkcją transponującą macierz i nie wymaga głębszych wyjaśnień.

Funkcja *gaussinv* służy do odwracania macierzy metodą eliminacji

Gaussa. Funkcja ta została wykorzystana ze względu na jej znacznie wyższą szybkość działania w porównaniu na przykład do metody Crammera. Jej dużą zaletą jest fakt, że nie jest konieczne wyznaczanie wielu minorów jak w przypadku wcześniej wymienionego sposobu.

Kod źródłowy 4.4: gaussinv.m

```
function b = gaussinv(a)

  [r,c] = size(a);

  if r ~= c
      disp("Blad odwracania macierzy, dozwolone tylko macierze
         kwadratowe!");
      b = [];
      return
  endif

  b = eye(r);

  for j = 1 : r
      for i = j : r
          if a(i,j) ~= 0
              for k = 1 : r
                  s = a(j,k);
                  a(j,k) = a(i,k);
                  a(i,k) = s;
                  s = b(j,k);
                  b(j,k) = b(i,k);
                  b(i,k) = s;
              endfor
              t = 1/a(j,j);
              for k = 1 : r
                  a(j,k) = t * a(j,k);
                  b(j,k) = t * b(j,k);
              endfor
              for L = 1 : r
                  if L ~= j
                      t = -a(L,j);
                      for k = 1 : r
                          a(L,k) = a(L,k) + t * a(j,k);
                          b(L,k) = b(L,k) + t * b(j,k);
                      endfor
                  endif
              endfor
          endif
          break
      endfor

      if a(i,j) == 0
          disp("Uwaga: Macierz osobliwa!")
          b = "Blad!";
          return
      endif
end
endfunction
```

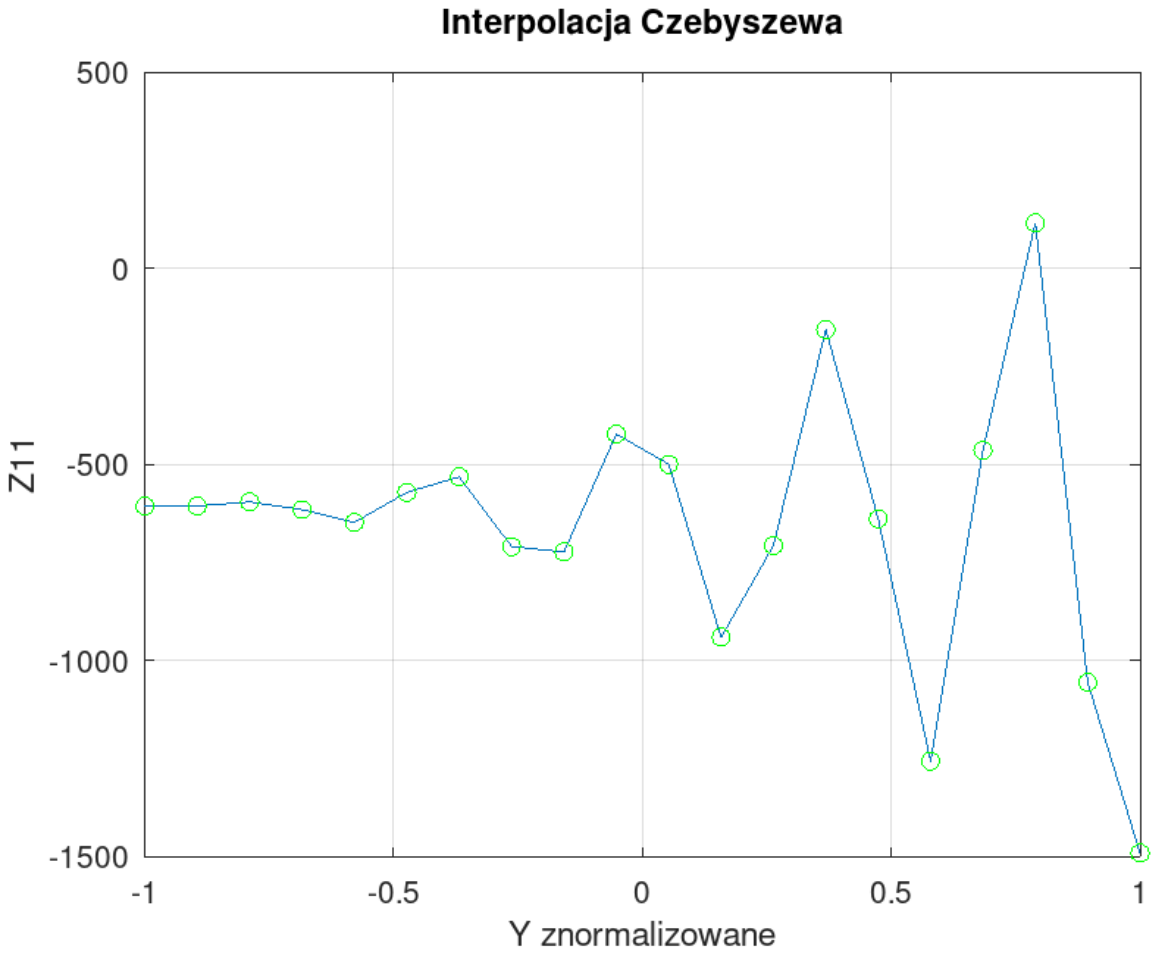
Funkcja przyjmuje za argument macierz przeznaczoną do odwrócenia (w tym przypadku *a*) i zwraca macierz *b* już odwróconą. Funkcja jest również zabezpieczona przed macierzą osobliwą (której wyznacznik == 0). W przypadku wykrycia takiej macierzy, zostanie zwrócony błąd.

Wywołanie funkcji interpolującej wielomianem Czebyszewa:

Kod źródłowy 4.5: main.m (fragment 8)

```
1 czebyszew_interpolacja_f(Y,Z11,Z11);
```

Wynik działania funkcji interpolacyjnej Czebyszewa dla punktów z wiersza 11 jest następujący:



Zielone kółka oznaczają interpolowane punkty dyskretne. Niebieska linia jest wykresem funkcji interpolacyjnej.

Wielomian (rekurencyjny) wygenerowany przez wskazaną funkcję:

```
Wielomian interpolacyjny Czebyszewa:
-385.248401 + 1264.712918 * T(1,x) + 391.805022 * T(2,x) + 962.990052 * T(3,x) + 207.472274 * T(4
,x) + 436.817607 * T(5,x) + 5.486572 * T(6,x) + -85.105376 * T(7,x) + -103.362661 * T(8,x) + -448
.016539 * T(9,x) + -163.581109 * T(10,x) + -725.923176 * T(11,x) + -361.818278 * T(12,x) + -946.0
60691 * T(13,x) + -473.292253 * T(14,x) + -572.605004 * T(15,x) + -68.637964 * T(16,x) + -232.008
415 * T(17,x) + -97.123200 * T(18,x) + -97.101373 * T(19,x)
```

Rysunek 4.1: Wielomian interpolacyjny Czebyszewa dla wiersza 11

T(y,x) wywołuje wcześniej przedstawioną metodę z pliku T.m. Wzór ten zostanie później wykorzystany do dalszych obliczeń.

4.3 Aproksymacja metodą najmniejszych kwadratów stopnia 2

W praktykach inżynierskich wykonywane są często eksperymenty polegające na pomiarach par wielkości, które, jak przypuszczamy, są ze sobą powiązane jakąś zależnością funkcyjną $y = f(x)$, są to na przykład: wydłużenie sprężyny w zależności od wiszącego na niej ciężaru, stężenie reagenta w zależności od czasu, zmiana temperatury w jednostce czasu itp. Dobrym posunięciem jest znalezienie takiej krzywej, która możliwie najlepiej przybliża te punkty odczytów. Znajdowanie takich krzywych jest istotą w **teorii aproksymacji** [5].

Aproksymacja w gruncie rzeczy służy do rozwiązywania dwóch typów problemów:

1. Gdy mamy podane punkty dyskretne np. bezpośrednio pobrane z wykonywanych pomiarów;
2. Gdy znana jest funkcja rozkładu, jednak wystąpiła potrzeba jej uproszczenia lub znalezienia pewnej zależności między danymi.

Do analizy problemu z niniejszego projektu wykorzystana została aproksymacja metodą najmniejszych kwadratów stopnia 1.

Kod odpowiedzialny za realizację zadania jest następujący:

Kod źródłowy 4.6: aproksymacja_kwadraty_f.m

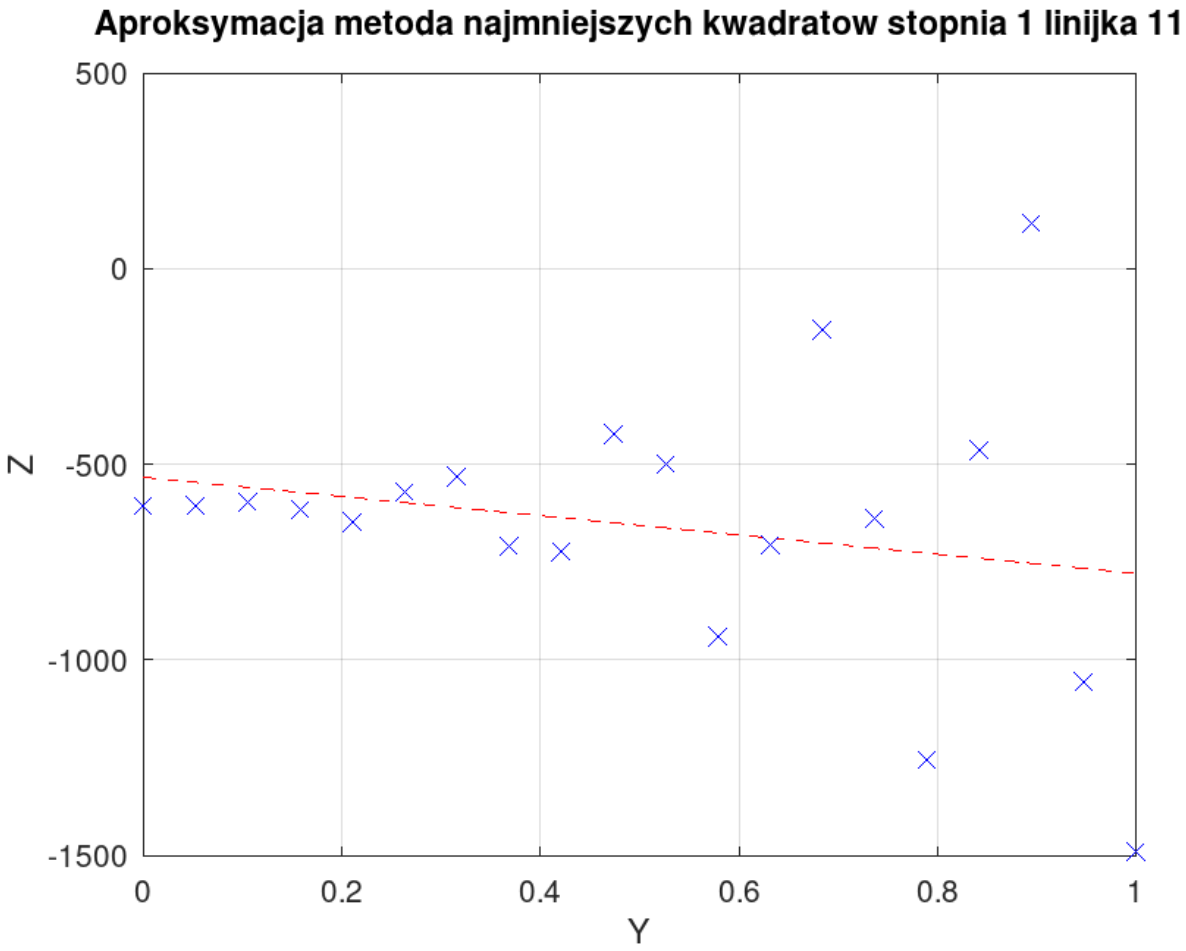
```
function A_ap=aproksymacja_kwadraty_f(x_ap,y_ap,n,linijka,zaok=100)
  % x_ap - macierz wejsciowa Y
  % y_ap - macierz wejsciowa Z
  % n - stopien wielomianu aproksymacyjnego
  % linijka - numer analizowanej linijki
  % zaok - domyslnie 100, zmienna mowiaca o liczbie punktow
     generowanych dla plynnego wykresu

  if length(x_ap)==length(y_ap)
  %ustalamy stopien wielomianu aproksymacyjnego
  stopien=n;
  n=length(x_ap);

  X_ap=zeros(stopien+1,stopien+1);
  A_ap=zeros(stopien+1,1);
  Y_ap=zeros(stopien+1,1);

  %obliczanie macierzy X
  for i=1:stopien+1
    for j=1:stopien+1
      for k=1:n
        %algorytm ustalajacy wartosci macierzy X
        X_ap(i,j)+=x_ap(1,k)^(i+(j-2));
      end
    end
  end

  %obliczanie macierzy Y
  for i=1:stopien+1
    for k=1:n
      Y_ap(i,1)+=y_ap(1,k)*x_ap(1,k)^(i-1);
    end
  end

  A_ap=il_macierzy(gaussinv(X_ap),Y_ap);

```

```
36    %obliczanie wartosci funkcji na podstawie obliczonych wartosci
          wspolczynnikow wielomianu
37    yi_ap=zeros(1,n);
38
39    for i=1:n
40      for j=1:stopien+1
41        yi_ap(1,i)+=A_ap(j,1)*x_ap(i)^(j-1);
42      endfor
43    endfor
44
45    disp("Wielomian aproksymacyjny: ");
46    j_ap=stopien+1;
47    for i=1:stopien
48      j_ap--;
49      printf("%f * x^%d + ",A_ap(j_ap+1),j_ap)
50    endfor
51    printf("%f\n",A_ap(j_ap))
52
53    %"plynny wykres"
54      step=(max(x_ap)-min(x_ap))/zaok;
55      Ry_ap=zeros(1,zaok+1);
56      Rx_ap=zeros(1,zaok+1);
57      %obliczanie wartosci funkcji Ry na podstawie ustalonych
            wspolczynnikow wielomianu
58      for i=1:zaok+1
59        Rx_ap(1,i)=min(x_ap)+(i-1)*step;
60        for j=1:stopien+1
61          Ry_ap(1,i)=Ry_ap(1,i)+A_ap(j)*Rx_ap(1,i)^(j-1);
62        endfor
63    endfor
64
65      tytul_rys=sprintf("Aproksymacja metoda najmniejszych kwadratow
            stopnia %d linijka %d", stopien, linijka);
66      plot(Rx_ap, Ry_ap,'--r');
67      hold on;
68      plot(x_ap, y_ap,'xb');
69      title(tytul_rys);
70
71      grid on;
72      xlabel("Y");
73      ylabel("Z");
74
75    else
76      disp("Liczba wspolrzednych x nie jest rowna liczbie wspolrzednych y
            ");
77    end
78  endfunction
```

W wyniku działania funkcji rysowany jest gładki wykres. W przypadku niniejszej analizy wykorzystany był stopień pierwszy wielomianu. Do wywołania funkcji służy polecenie:

Kod źródłowy 4.7: main.m (Fragment 9)

```
1  aproksymacja_kwadraty_f(Y,trans(Z11),1,b,1000);
```



W toku obliczeń uzyskany został następujący wielomian aproksymacyjny $ax+b$:

```
Wielomian aproksymacyjny:
-245.909147 * x^1 + -532.856932
```

Rysunek 4.2: Wielomian aproksymacyjny 1 stopnia dla wiersza 11

4.4 Całkowanie numeryczne

Matematycznie rzecz biorąc całkowanie to operacja odwrotna do różniczkowania. Obliczając całkę w rzeczywistości obliczamy pole powierzchni znajdujące się pod funkcją, z której ją obliczyliśmy.

Komputerowe wyliczenia całek różnią się od analitycznych (dokładnych) ze względu na to, że komputer może wykonywać ogromną

liczbę nieskomplikowanych obliczeń. W tym celu najłatwiej jest podzielić przedział $[a,b]$ na wiele małych podprzedziałów (im ich więcej, tym całka jest dokładniejsza).

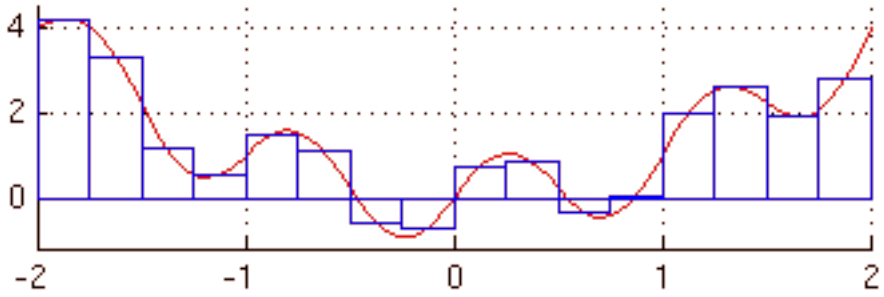
Istnieje wiele metod całkowania na przykład metoda prostokątów lub Simpsona. W niniejszej analizie wykorzystana została jednak metoda trapezów. Polega ona na podziale obliczanego przedziału na pół w taki sposób, aby powstał trapez.

W najpopularniejszych metodach do dzielenia przedziałów wykorzystuje się zmienną:

$$p = \frac{b-a}{n} \tag{4.4}$$

Porównanie algorytmów całkowania: [2]

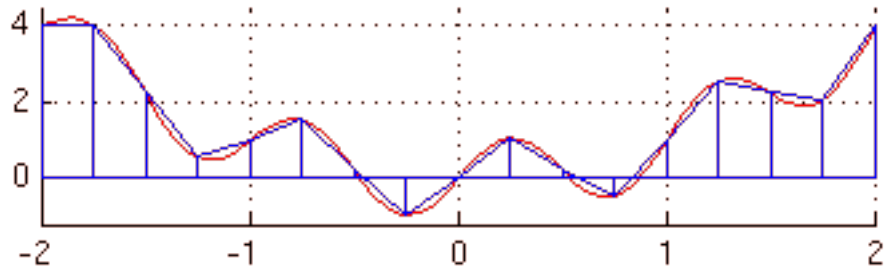
- Metoda kwadratów:



Stosujemy wzór:

$$\int_a^b f(x)dx = p\sum_{i=0}^{n-1} f(x_i) \tag{4.5}$$

- Metoda trapezów: Dokładniejsza niż metoda prostokątów.



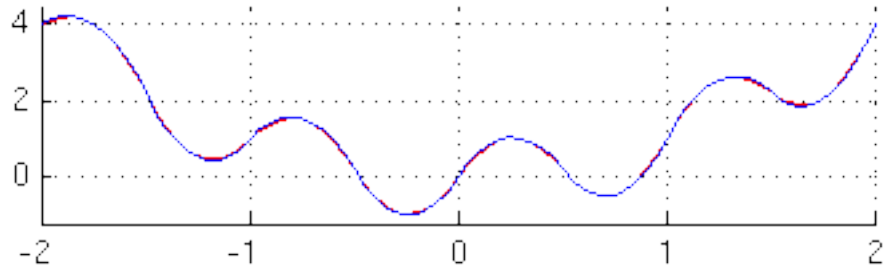
Stosujemy wzór:

$$\int_a^b f(x)dx = p[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)] \tag{4.6}$$

lub:

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} [\frac{f(x_i)+f(x_{i+1})}{2} p] \tag{4.7}$$

- Metoda Simpsona



Jest to jedna z dokładniejszych metod całkowania numerycznego. Metoda wymaga podzielenia przedziału całkowania na parzystą liczbę $2n$ podprzedziałów.

Do całkowania wykorzystane zostało oprogramowanie **Octave**, w którym została zawarta funkcja wbudowana *quad*, licząca numerycznie pole z podanej funkcji i zakresu. Została ona wykorzystana do sprawdzenia poprawności otrzymanych wyników całkowania za pomocą metody trapezów. Kod źródłowy dla całki, wykorzystany do obliczeń całek w projekcie, jest następujący: Funkcja zastosowana do obliczeń wygląda następująco:

Kod źródłowy 4.8: calka_f.m

```
function calka = calka_f(a,b,n,fz)
% Calkowanie metoda trapezow
% a, b - przedzial
% n - liczba podzialow
% fz - funkcja anonimowa

% Funkcja zwraca zmienna calka

  suma=0;
  h=(b-a)/n;
  x=zeros(n,1);
  y=zeros(n,1);
  calka=0;
  for i=1:n
    x(i,1)=a+i*h;
    y(i,1)=fz(x(i,1));
  endfor

  suma=((y(1,1)+y(n,1))/2);

  for i=1:n-1
     suma+=fz(x(i,1));
  endfor
  calka=h*suma;
endfunction
```

4.4.1 Całka z funkcji interpolacyjnej

W obliczeniach wykorzystana zostanie metoda trapezów z interpolacji Czebyszewa. Dla przypomnienia wzór rekurencyjny uzyskanego wielomianu:

```
Wielomian interpolacyjny Czebyszewa:
-385.248401 + 1264.712918 * T(1,x) + 391.805022 * T(2,x) + 962.990052 * T(3,x) + 207.472274 * T(4
,x) + 436.817607 * T(5,x) + 5.486572 * T(6,x) + -85.105376 * T(7,x) + -103.362661 * T(8,x) + -448
.016539 * T(9,x) + -163.581109 * T(10,x) + -725.923176 * T(11,x) + -361.818278 * T(12,x) + -946.0
60691 * T(13,x) + -473.292253 * T(14,x) + -572.605004 * T(15,x) + -68.637964 * T(16,x) + -232.008
415 * T(17,x) + -97.123200 * T(18,x) + -97.101373 * T(19,x)
```

Rysunek 4.3: Wielomian interpolacyjny Czebyszewa dla wiersza 11

Wywołanie funkcji:

Kod źródłowy 4.9: main.m (fragment 10)

```
%Przypisanie wielomianu interpolacyjnego
fint=@(x) -385.248401 + 1264.712918 * T(1,x) + 391.805022 * T(2,x) +
   962.990052 * T(3,x) + 207.472274 * T(4,x) + 436.817607 * T(5,x) +
   5.486572 * T(6,x) + -85.105376 * T(7,x) + -103.362661 * T(8,x) + -
   448.016539 * T(9,x) + -163.581109 * T(10,x) + -725.923176 * T(11,x)
   + -361.818278 * T(12,x) + -946.060691 * T(13,x) + -473.292253 * T(14
   ,x) +-572.605004 * T(15,x) + -68.637964 * T(16,x) + -232.008415 * T(
   17,x) + -97.123200 * T(18,x) + -97.101373 * T(19,x);

  tic(); printf("\nCalka z funkcji interpolacyjnej dla 100 podzialow: %
     f \n",calka_f(Y(1,1), Y(1,20), 100,fint)); toc
  tic(); printf("\nCalka z funkcji interpolacyjnej dla 250 podzialow: %
     f \n",calka_f(Y(1,1), Y(1,20), 250,fint)); toc
  tic(); printf("\nCalka z funkcji interpolacyjnej dla 400 podzialow: %
     f \n",calka_f(Y(1,1), Y(1,20), 400,fint)); toc
  tic(); printf("\nCalka z funkcji interpolacyjnej dla 1000 podzialow:
     %f \n",calka_f(Y(1,1), Y(1,20), 1000,fint)); toc
  tic(); printf("\nSprawdzenie funkcja wbudowana Octave: %f\n", quad(
     fint,Y(1,1),Y(1,20))); toc
```

Wynikiem działania funkcji jest pole powierzchni pod wykresem. Zwracany jest również czas wykonywanych obliczeń.

```
Calka z funkcji interpolacyjnej dla 100 podzialow: -295.483001
Elapsed time is 137.959 seconds.

Calka z funkcji interpolacyjnej dla 250 podzialow: -290.500825
Elapsed time is 351.651 seconds.

Calka z funkcji interpolacyjnej dla 400 podzialow: -289.920885
Elapsed time is 537.306 seconds.

Calka z funkcji interpolacyjnej dla 1000 podzialow: -289.608569
Elapsed time is 1271.62 seconds.

Sprawdzenie funkcja wbudowana Octave: -289.549092
Elapsed time is 13.3029 seconds.
```

Rysunek 4.4: Wyniki całkowania wielomianu interpolacyjnego

Jak wynika z obliczeń, zwiększenie liczby podziałów w przypadku całkowania metodą trapezów, skutkuje zwiększeniem precyzji na wyjściu. Oczywistością jest, że przy liczbie tysiąca podziałów wynik jest najbardziej zbliżony do całki obliczonej za pomocą specjalnie zoptymalizowanej funkcji *quad*. Istotne jest również porównanie czasu

pracy funkcji. Metoda trapezów przy tej liczbie podziałów zajęła w moim środowisku obliczeniowym 1253 sekund, co jest dużym czasem w porównaniu z funkcją wbudowaną, która z tym samym zadaniem poradziła sobie w 13 sekund, osiągając przy tym wyższą dokładność. Długi czas obliczeń jest również spowodowany rekurencją w wielomianie interpolacyjnym.

4.4.2 Całka z funkcji aproksymacyjnej

W obliczeniach wykorzystana zostanie metoda trapezów. Dla przypomnienia wzór uzyskanego wielomianu aproksymacyjnego $ax + b$:

```
Wielomian aproksymacyjny:
-245.909147 * x^1 + -532.856932
```

Rysunek 4.5: Wielomian aproksymujący do obliczeń całki

Wywołanie funkcji:

Kod źródłowy 4.10: main.m (Fragment 11)

```
1 fapr=@(x) -245.909147 * x^2 + -532.856932 * x^1 +-532.856932;
2
3 tic(); printf("\n\nCalka z funkcji aproksymacyjnej dla 100 podzialow:
      %f \n",calka_f(Y(1,1), Y(1,20), 100,fapr)); toc
4 tic(); printf("\n\nCalka z funkcji aproksymacyjnej dla 250 podzialow:
      %f \n",calka_f(Y(1,1), Y(1,20), 250,fapr)); toc
5 tic(); printf("\n\nCalka z funkcji aproksymacyjnej dla 400 podzialow:
      %f \n",calka_f(Y(1,1), Y(1,20), 400,fapr)); toc
6 tic(); printf("\n\nCalka z funkcji aproksymacyjnej dla 1000 podzialow
    : %f \n",calka_f(Y(1,1), Y(1,20), 1000,fapr)); toc
7 tic(); printf("\nSprawdzenie funkcja wbudowana Octave: %f\n", quad(
    fapr,Y(1,1),Y(1,20))); toc
```

Wynikiem działania funkcji jest pole powierzchni pod wykresem. Zwracany jest również czas wykonywanych obliczeń.

```
Calka z funkcji aproksymacyjnej dla 100 podzialow: -655.823801
Elapsed time is 0.00287795 seconds.

Calka z funkcji aproksymacyjnej dla 250 podzialow: -655.813473
Elapsed time is 0.00683808 seconds.

Calka z funkcji aproksymacyjnej dla 400 podzialow: -655.812274
Elapsed time is 0.012187 seconds.

Calka z funkcji aproksymacyjnej dla 1000 podzialow: -655.811629
Elapsed time is 0.0281639 seconds.

Sprawdzenie funkcja wbudowana Octave: -655.811506
Elapsed time is 0.000180006 seconds.
```

Rysunek 4.6: Wyniki całkowania wielomianu aproksymacyjnego

W przypadku mniejszego stopnia wielomianu obliczenia są znacznie szybsze i dokładniejsze. Już przy 250 podziałach wynik jest satysfakcjonujący (w porównaniu z funkcją wbudowaną *quad*).

4.5 Pochodne i monotoniczność

4.5.1 Różniczkowanie numeryczne macierzy Z

Iloraz różnicowy jest to wielkość (liczba) opisująca przyrost funkcji na zadanym przedziale. Wykorzystujemy go najczęściej w przypadkach, gdy mamy: [4]

- Funkcję podaną w postaci dyskretnej (punkty)
- Funkcja jest na tyle skomplikowana, że poszukiwanie wartości pochodnych w sposób klasyczny (analityczny) jest utrudnione (ze względu na ograniczony czas, zasoby itp.)

Wzór na iloraz centralny:

$$f^1(x_0) = \frac{f(x_1) - f(x_{-1})}{(x_1 - x_{-1})} \tag{4.8}$$

Przy czym możemy również wyznaczyć pochodne brzegowe lewostronne i prawostronne:

$$\begin{aligned} f^1(x_0) &= \frac{f(x_0) - f(x_{-1})}{(x_0 - x_{-1})} \\ f^1(x_0) &= \frac{f(x_1) - f(x_0)}{(x_1 - x_0)} \end{aligned} \tag{4.9}$$

Kod **Octave** wykorzystany do różniczkowania wygląda następująco:

Kod źródłowy 4.11: pochodna_f.m

```
function [PoX, PoY] = pochodna_f(X,Y,Z,wykres=0)
  % Funkcja przyjmuje macierze X,Y,Z oraz zmienna wykres
  % Jesli zmienna wykres bedzie rowna 1 to skrypt rysuje wykres
  PoX = zeros(20);
  PoY = zeros(20);

  for i=1:20
    PoX(1,i)=(Z(2,i)-Z(1,i))/(X(2)-X(1));
    PoX(20,i)=(Z(20,i)-Z(19,i))/(X(20)-X(19));
    PoY(i,1)=(Z(i,2)-Z(i,1))/(Y(2)-Y(1));
    PoY(i,20)=(Z(i,20)-Z(i,19))/(Y(20)-Y(19));
    for j=2:19
      PoX(j,i)=((Z(i,j+1)-Z(i,j-1))/(X(j+1)-X(j-1)));
```

```
      PoY(i,j)=((Z(j+1,i)-Z(j-1,i))/(Y(j+1)-Y(j-1)));
    endfor
  endfor
  if(wykres==1)
    figure()
    contourf(X,Y,PoY); % Wykres pochodnej po X
    title("Pochodna po X")
    xlabel("Y");
    ylabel("X");
    colormap('hot');
    colorbar
    figure()
    contourf(X,Y,PoX); % Wykres pochodnej po Y
    title("Pochodna po Y")
    colormap('hot');
    xlabel("X");
    ylabel("Y");
    colorbar
  endif
endfunction
```

Wywołanie funkcji, w odpowiedzi zwrócone są macierz px i py z wartościami pochodnych.

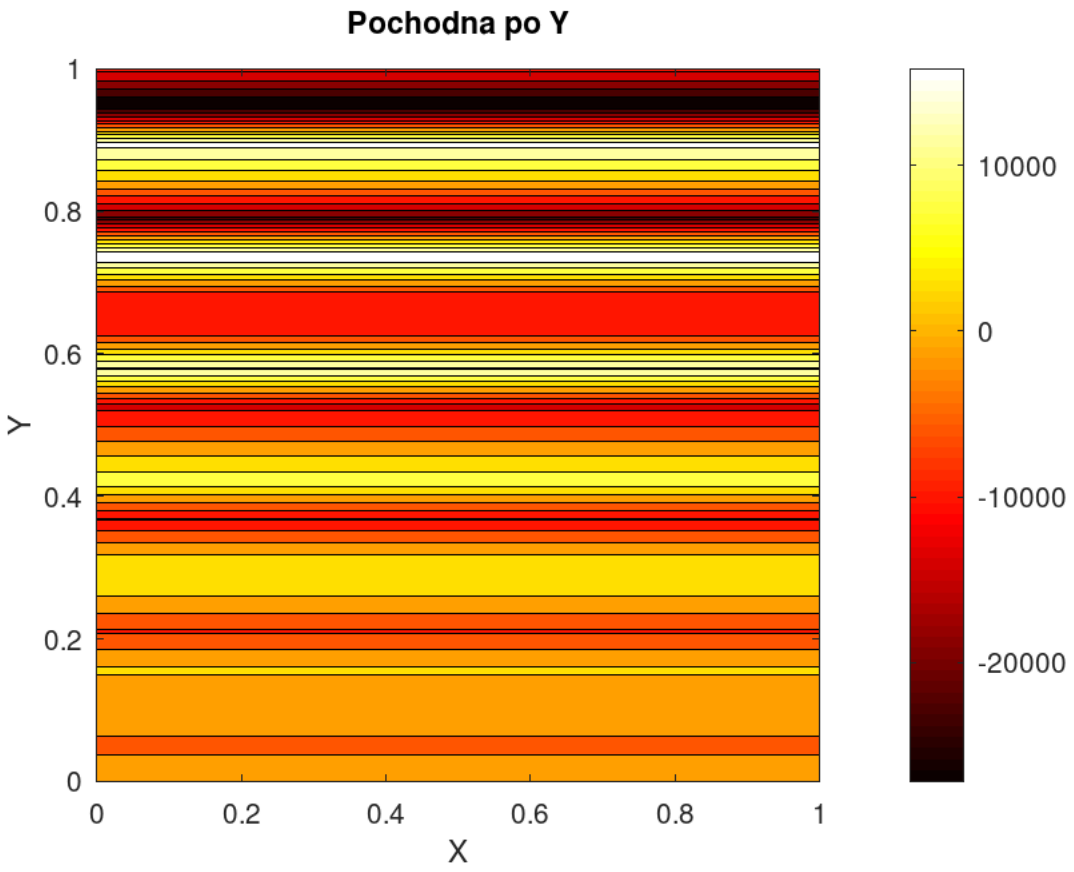
Kod źródłowy 4.12: main.m (Fragment 12)

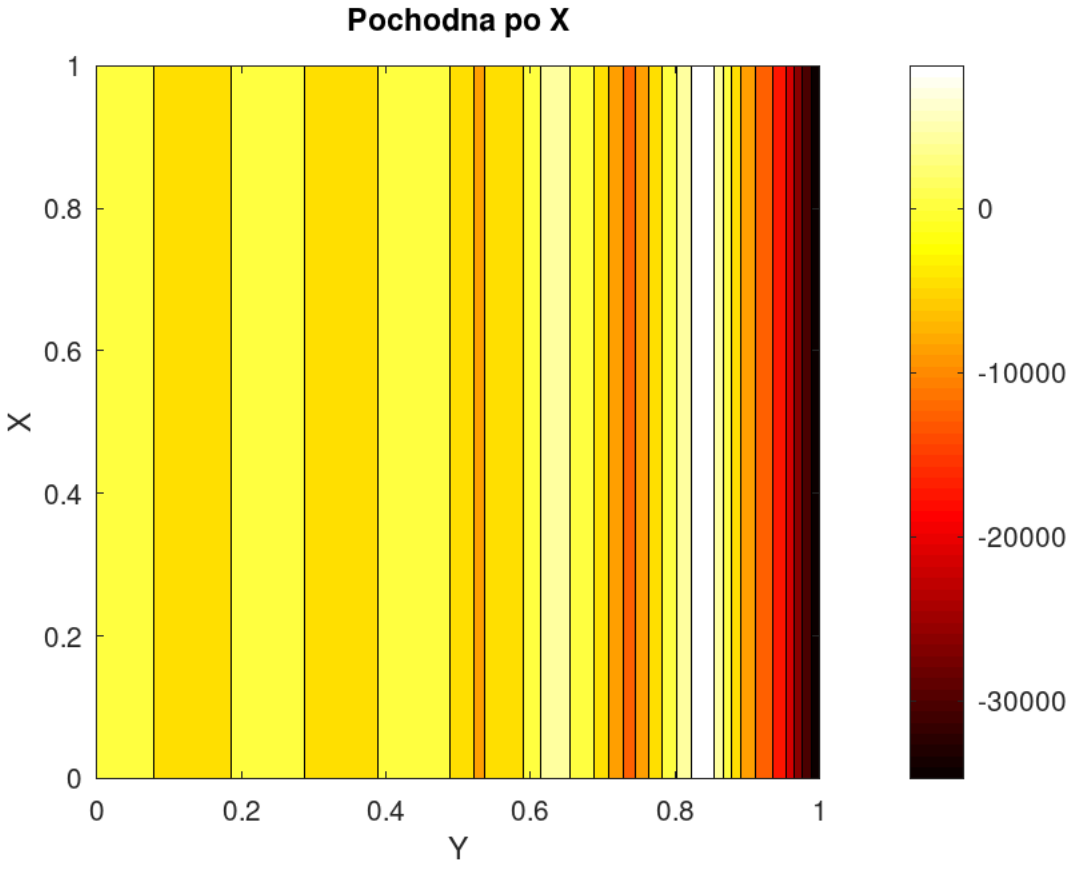
```
[px, py]=pochodna_f(X,Y,Z);
```

4.5.2 Monotoniczność punktów dyskretnych na podstawie ilorazów różnicowych

W celu określenia monotoniczności punktów dyskretnych utworzone zostały dwa wykresy. Kolory na wykresach stanowią o monotoniczności przy czym odczytane z nich wartości należy interpretować w następujący sposób:

$$\begin{aligned} f^{(1)}(x) > 0 &\quad \text{- funkcja rosnąca} \\ f^{(1)}(x) < 0 &\quad \text{- funkcja malejąca} \\ f^{(1)}(x) = 0 &\quad \text{- funkcja stała} \end{aligned} \tag{4.10}$$


Pochodna po Y
1
0.8
0.6
0.4
0.2
0
Y
0
0.2
0.4
0.6
0.8
1
X
10000
0
-10000
-20000


Pochodna po X
1
0.8
0.6
0.4
0.2
0
X
0
0.2
0.4
0.6
0.8
1
Y
0
-10000
-20000
-30000

4.6 Pole powierzchni figury 3D

Pole powierzchni, potocznie zwane krótko polem lub powierzchnią figury, to wielkość przyporządkowująca danej figurze nieujemną liczbę w pewnym ujęciu charakteryzującą jej rozmiar.

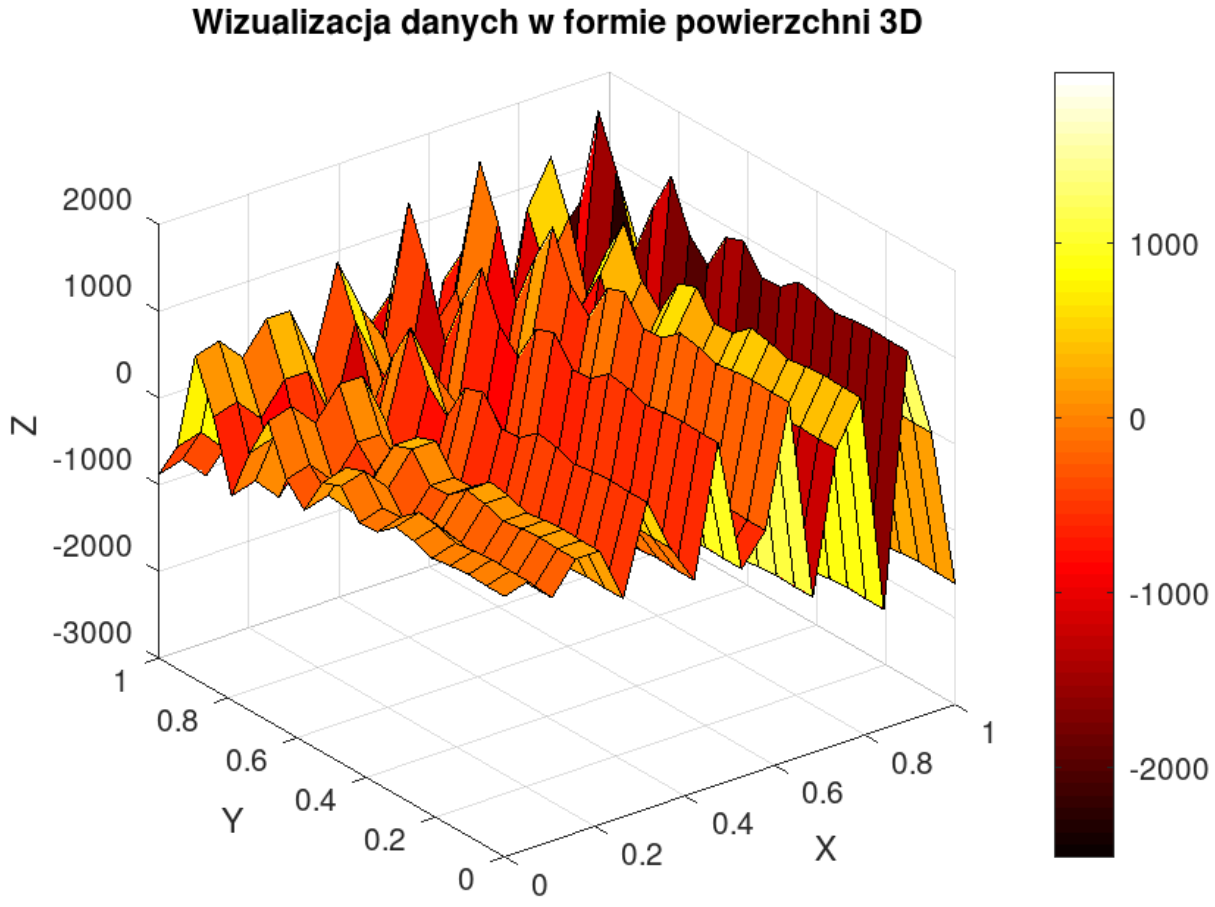
W celu obliczenia pola powierzchni figury, została ona podzielona na trójkąty. Do obliczeń zastosowany został *wzór herona*:

$$S = \sqrt{p(p-a)(p-b)(p-c)} \tag{4.11}$$

Gdzie wzór na *połowę obwodu* (p) wykorzystany we *wzorze herona*:

$$p = \frac{a+b+c}{2} \tag{4.12}$$

Wzory te zostały zastosowane dla figury widocznej na poniższym wykresie:



Do obliczenia pola wykorzystane zostały struktury w programie **Octave**, w celu łatwiejszego przekazywania poszczególnych punktów do funkcji. Poniżej zamieszczony został kod źródłowy poszczególnych metod:

Kod źródłowy 4.13: pole_figury_f.m

```
function suma_pola=pole_figury_f(X,Y,Z)
a=struct;
b=struct;
c=struct;
d=struct;
   suma_pola = 0.0;
    for i = 1:19
       for j=1:19
           a.x=X(i); a.y=Y(j); a.z=Z(i,j);
           b.x=X(i + 1); b.y=Y(j); b.z=Z(i + 1,j);
           c.x=X(i); c.y=Y(j + 1); c.z=Z(i,j + 1);
           d.x=X(i + 1); d.y=Y(j + 1); d.z=Z(i + 1, j + 1);
           suma_pola += pole(a,b,c,d);
       endfor
    endfor
endfunction
```

Kod źródłowy 4.14: pole_trojkata.m

```
function pole=pole_trojkata(r,t,g)
    p = (r + t + g) / 2;
    pole = sqrt(p * (p - r)*(p - t)*(p - g));
endfunction
```

Kod źródłowy 4.15: pole.m

```
function pole_obliczane = pole(a, b, c, d)
    e = sqrt((a.x - b.x)^2 + (a.z - b.z)^2);
    f = sqrt((c.y - a.y)^2 + (c.z - a.z)^2);
    podstawa = sqrt((b.x - a.x)^2 + (a.y - c.y)^2);
    g = sqrt((c.z - b.z)^2 + (podstawa)^2);
    pole_obliczane = pole_trojkata(e, f, g);
    e = sqrt((c.x - d.x)^2 + (c.z - d.z)^2);
    f = sqrt((d.y - b.y)^2 + (d.z - b.z)^2);
    pole_obliczane += pole_trojkata(e, f, g);
endfunction
```

Poniżej prezentuję wywołanie funkcji:

Kod źródłowy 4.16: main.m (Fragment 13)

```
printf("Pole powierzchni figury wynosi: %f\n", pole_figury_f(X,Y,Z));
```

Instrukcja wyświetla na ekranie pole powierzchni figury zmierzone według przyjętych wartości macierzy X, Y, Z

```
Pole powierzchni figury wynosi: 23245.948966
```

Rysunek 4.7: Pole figury wyliczone wzorem herona

Bibliografia

[1] Dokumentacja programu Octave, Descriptive Statistics. https://octave.org/doc/v7.1.0/Descriptive-Statistics.html.

[2] Całkowanie numeryczne. *Wikipedia*, 2022.

[3] Adam Kulawik. Interpolacja część 3, Interpolacja Czebyszewa, Maj 2022.

[4] Adam Kulawik. Różniczkowanie numeryczne cz.1, Maj 2022.

[5] Jan Pyka and Izabella Foltynowicz. Aproksymacja metodą najmniejszych kwadratów. https://qcg.home.amu.edu.pl/pliki/Aproksymacja.pdf, Maj 2022.