|  |  |
| --- | --- |
| Optymalizacja funkcji wielu zmiennych metodami gradientowymi | |
| Autorzy | Kozłowski Bartosz,  Kopeć Jakub,  Kobyłecki Emil |
| Data wysłania | 15.12.2020 |

**1. Wejściowe parametry algorytmu najszybszego spadku:**



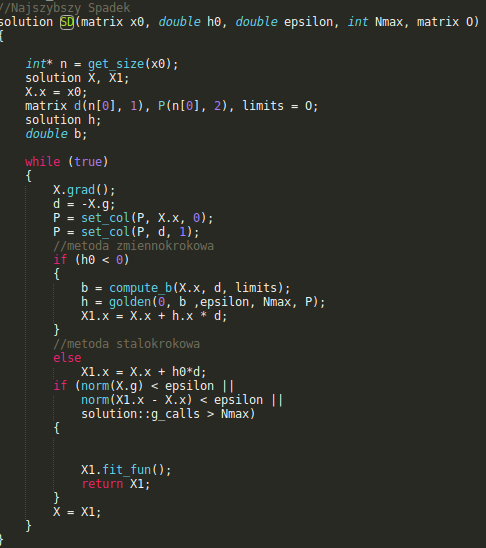
x0- macierz, w której przechowywane są początkowe wartości x.

h0- wielkość kroku

epsilon- dokładność obliczeń

Nmax- maksymalna ilość iteracji

*Kod metody:*

**

**2. Wejściowe parametry algorytmu gradientów sprzężonych:**



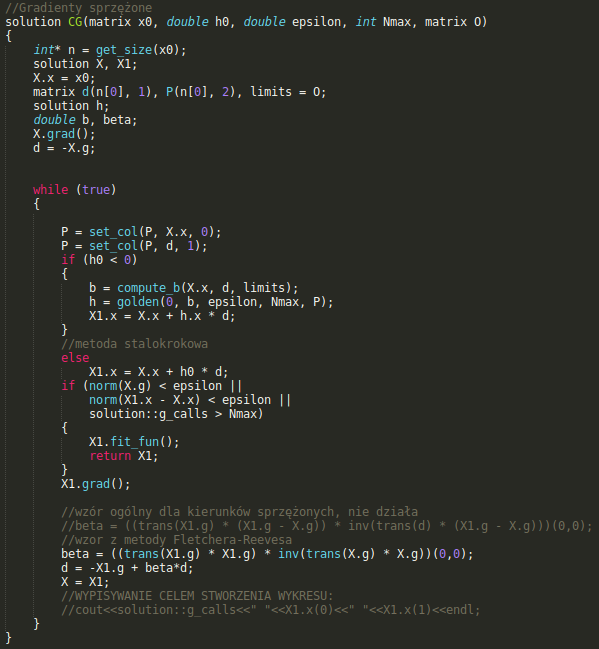
x0- macierz, w której przechowywane są początkowe wartości x.

h0- wielkość kroku

epsilon- dokładność obliczeń

Nmax- maksymalna ilość iteracji

*Kod metody:*

**

**3.Wejściowe parametry algorytmu Newtona:**

****

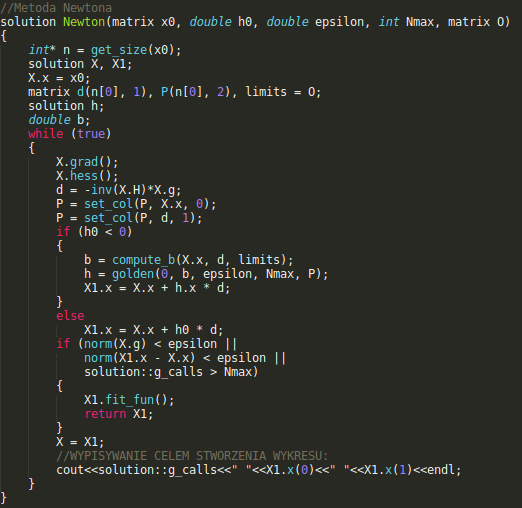
x0- macierz, w której przechowywane są początkowe wartości x.

h0- wielkość kroku

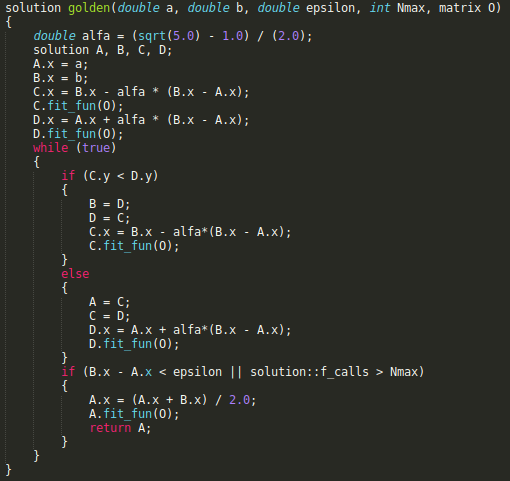
epsilon- dokładność obliczeń

Nmax- maksymalna ilość iteracji

*Kod metody:*

**

*Złoty podział wykorzystywany przez wszystkie opisane wcześniej algorytmy:*

**

**4. Dyskusja wyników oraz wnioski:**

*Funkcja testowa:*

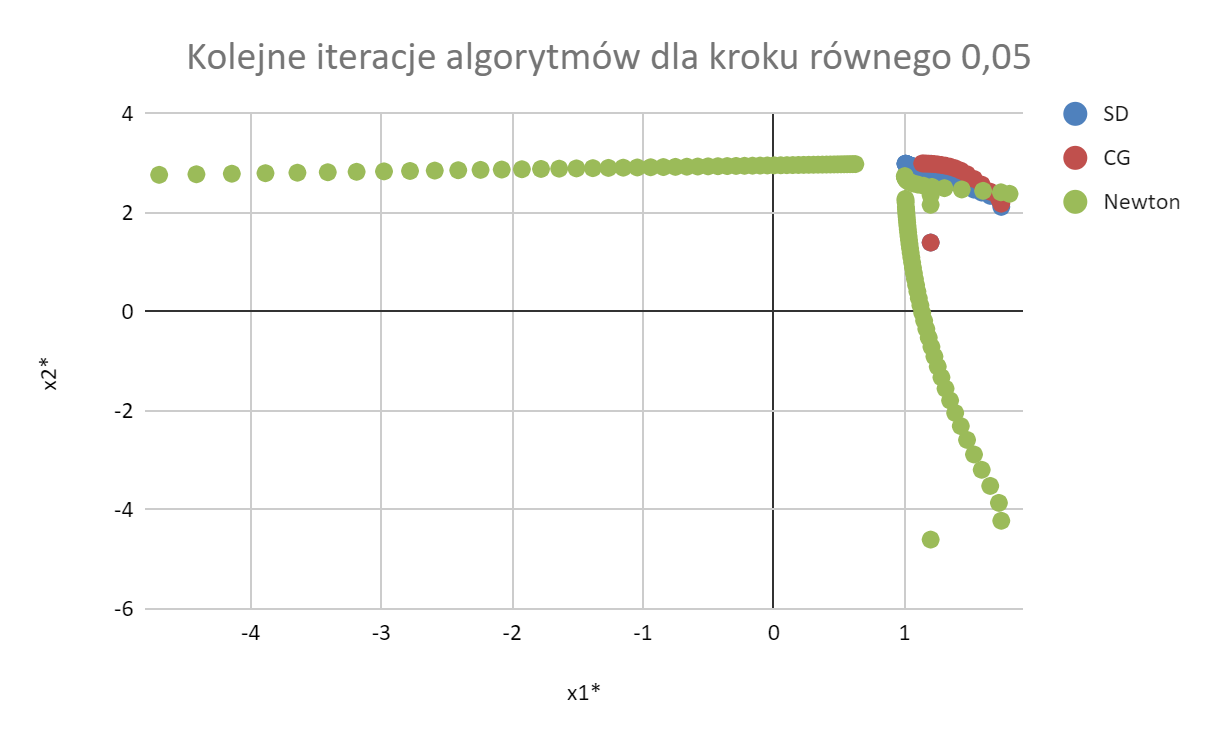
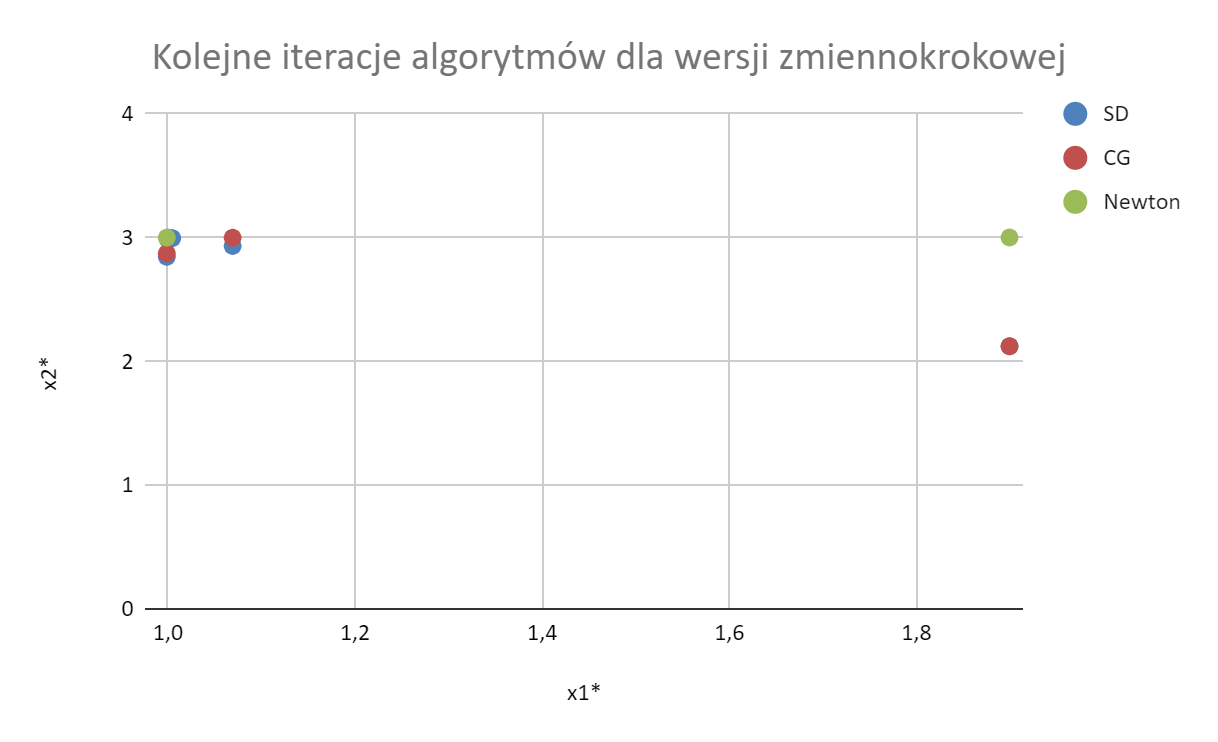
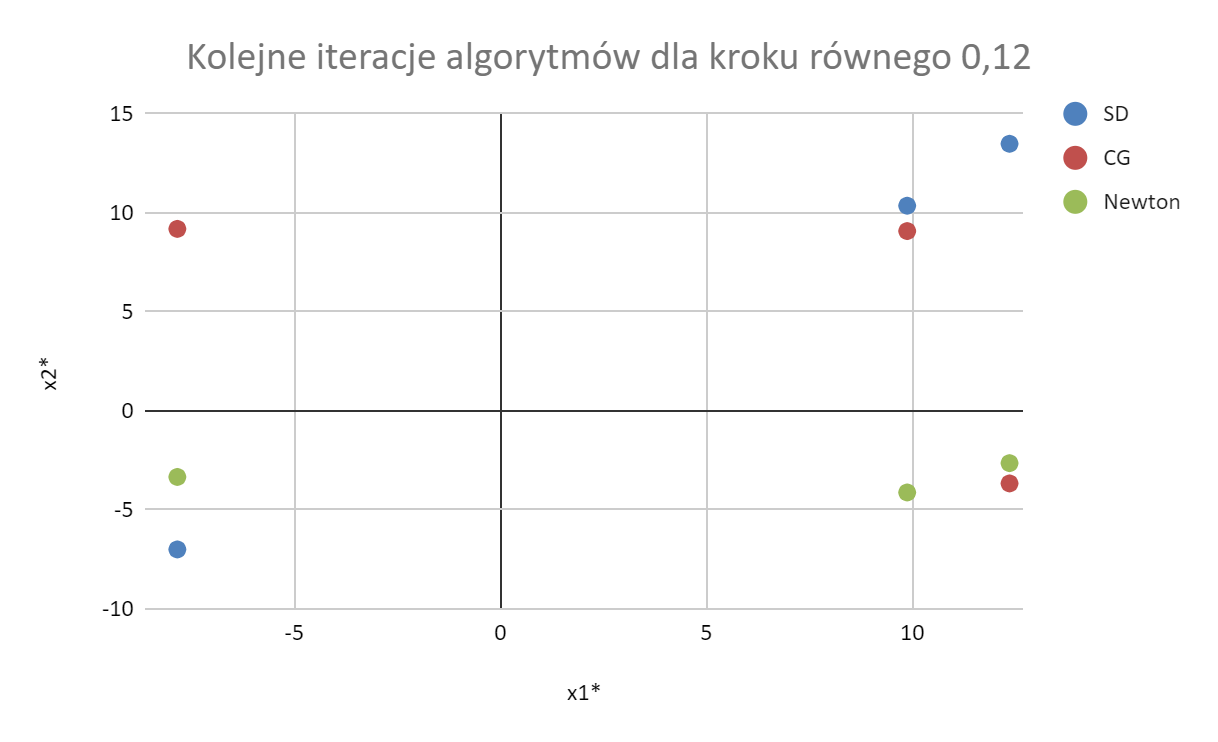
**

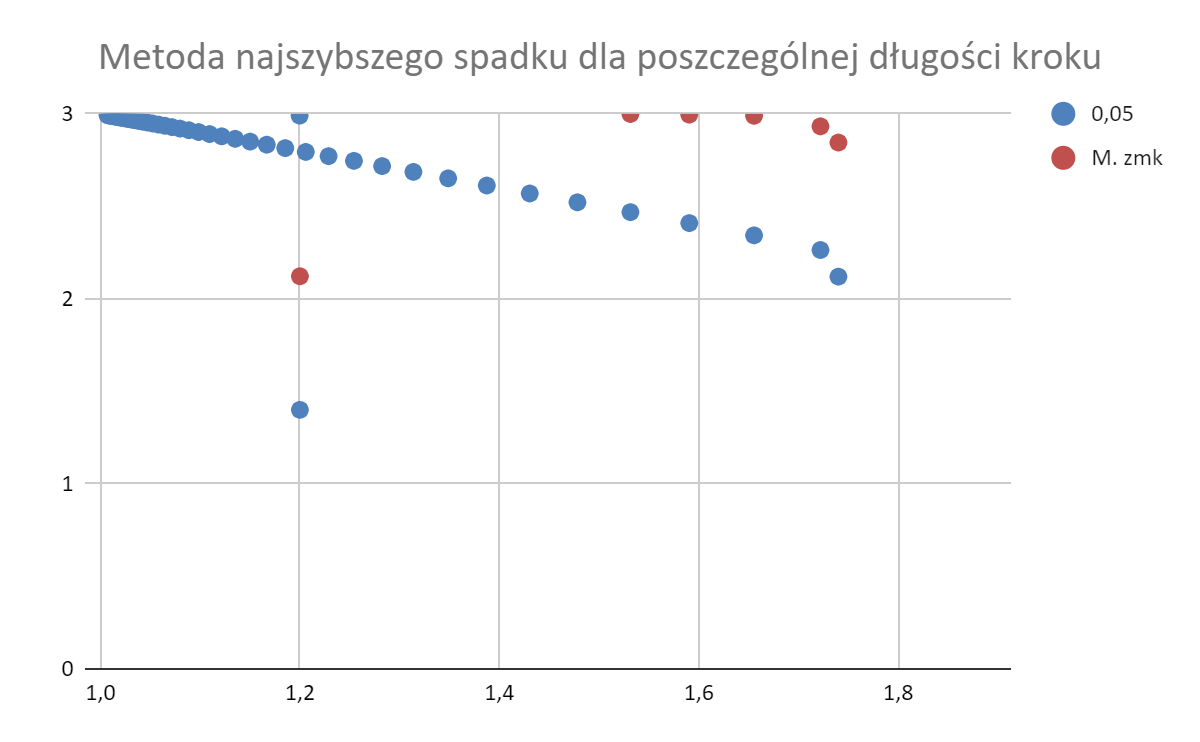
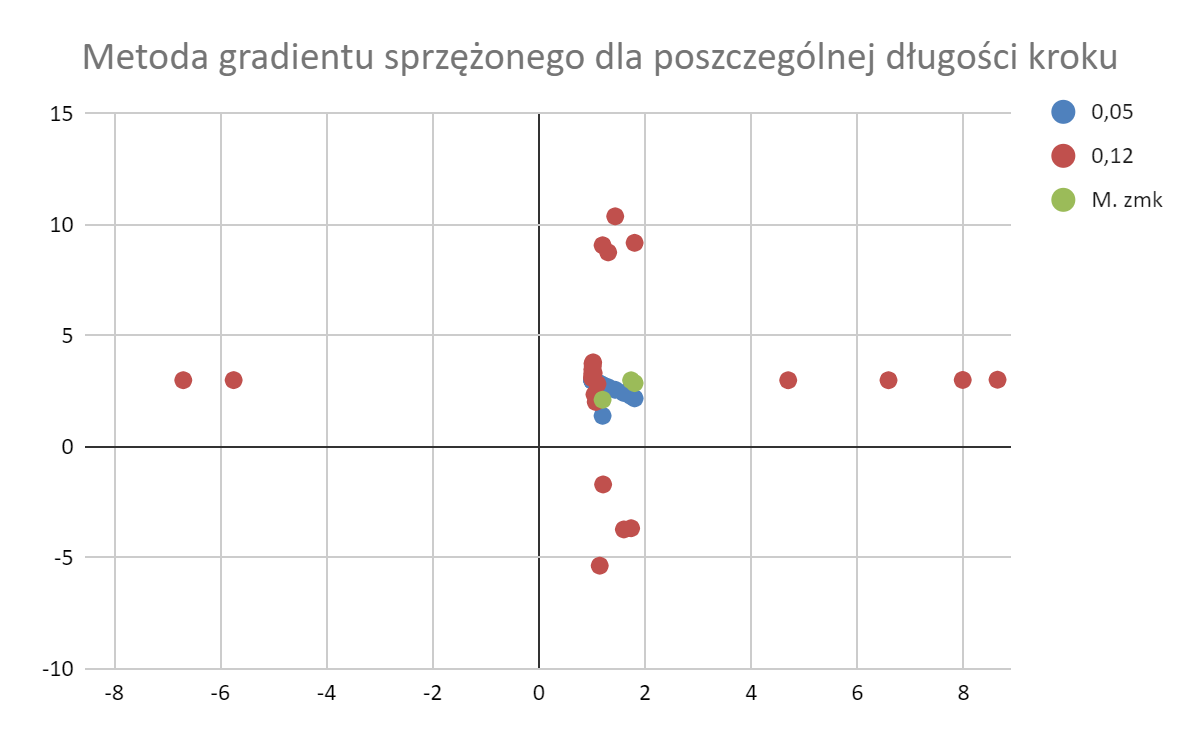
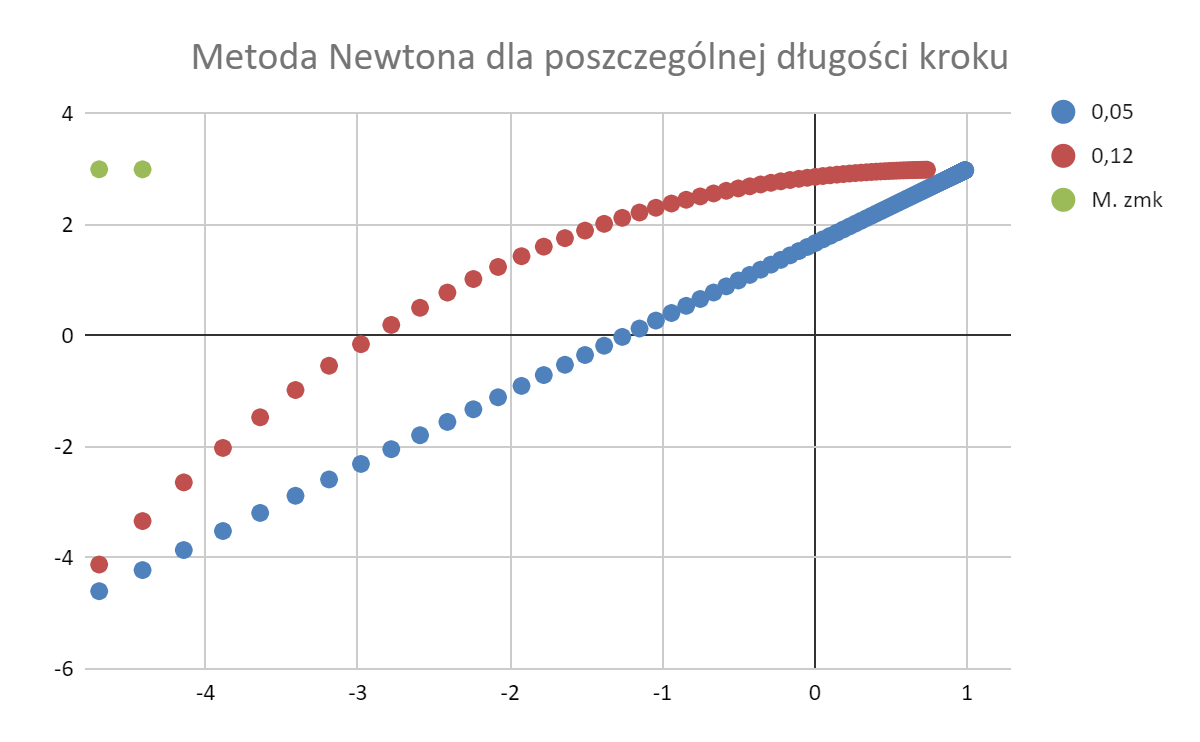
Wyniki osiągane dla powyższej funkcji testowej przez algorytmy są zbliżone do rzeczywistych. Wyjątkiem jest algorytm najszybszego spadku gdzie przy ustawieniu wielkości kroku większej od 0.11 osiągane minima nie są nawet zbliżone do poprawnych wartości. Co dość oczywiste im mniejsza wielkość kroku tym znajdowane minimum jest dokładniejsze.

*Wybrany punkt do sprawdzenia ilości wywołań (g\_calls()):*

(-5,-5)

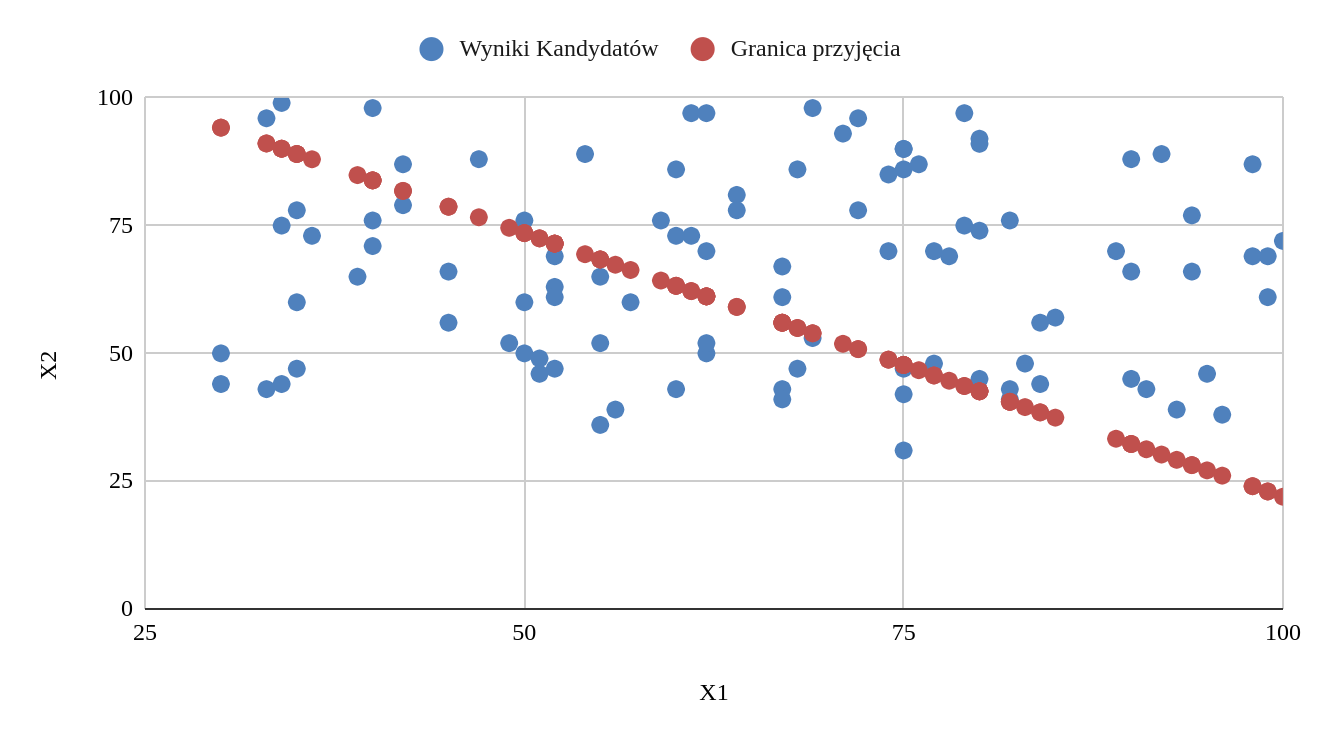
Jak wynika z uzyskanych wartości największa ilość wywołań pojawiała się przy długości kroku równej 0.12 dla każdej z metod. Najmniejsza natomiast dla zmiennej wielkości kroku.

Z powodu dążenia współrzędnych poszukiwanego minimum do nieskończoności, postanowiliśmy nie uwzględniać większości punktów w metodzie SD dla kroku równego 0,12.  
  
Serie danych w poniższych wykresach mają na celu porównanie schematu poszukiwania minimum wszystkimi trzema metodami dla różnej długości kroku:  
W pierwszym przypadku całkowicie usunąłem serię danych dla kroku równego 0,12, ponieważ jego długość jest zbyt duża aby algorytm mógł działać poprawnie, przez co współrzędne minimum dążą do nieskończoności.

Wartości teta,J(teta),P(teta) opisane w arkuszu “klasyfikator” są zbliżone do rzeczywistych. Wyjątek stanowią dla długości kroku równej 0.01 ,gdzie podobnie jak w przypadku opisanym wcześniej mocno odbiegają od rzeczywistości. W obu miejscach wynika to ze zbyt małej długości kroku co stanowi źródło tych błędów.

*Granica klasyfikacji:*



Jak wynika z powyższej grafiki osiągnięta “granica przyjęcia” jest zbliżona do założonej.

Wykorzystane w zadaniu algorytmy działają poprawnie o ile wielkość kroku jest odpowiednio mała. Podobnie ma się sprawa z parametrem epsilon- w problemie rzeczywistym dla jego zbyt dużych wartości osiągane wyniki nie były zgodne z przyjętymi założeniami.  
  
Na podstawie wykresów można stwierdzić, że używając metod o określonej długości kroku, należy być ostrożnym przy jego doborze. Nieodpowiedni jego dobór skutkuje błędnymi obliczeniami przez co zalecane jest sprawdzenie działania algorytmu dla różnych długości kroków.   
  
Metoda zmiennokrokowa jest znacznie bardziej zbieżna od metod stałokrokowych. Szukanie minimum tą metodą wymaga mniejszej liczby iteracji niż w przypadku pozostałych algorytmów. Jednakże każda iteracja w tej metodzie jest bardziej złożona niż w metodach stałokrokowych.  
  
W arkuszu kalkulacyjnym zawarte są także zakładki o nazwach wykres123 i wykres456. Powstały one tylko po to, aby połączyć dane z odpowiednich kolumn w pojedyncze, duże kolumny oraz na ich podstawie wykonać wykresy(wszystkie serie danych muszą być w jednej kolumnie aby wykres został wygenerowany poprawnie) Nie zrobiliśmy tego łączenia w arkuszu “wykresy” aby zachować przejrzystość danych wyjściowych algorytmów.  
  
**5. Kod źródłowy:**  
*fit\_fun(),grad(),hess() dla problemu rzeczywistego:*

**void solution::fit\_fun(matrix O)**

**{**

**//ODCZYTYWANIE DANYCH Z PLIKU**

**int m=100;**

**int \*n=get\_size(x);**

**static matrix X(n[0],m),Y(1,m); //Wektor z ocenami**

**if(solution::f\_calls==0)**

**{**

**ifstream S("XData.txt");**

**S>>X;**

**S.close();**

**S.open("YData.txt");**

**S>>Y;**

**S.close();**

**}**

**double h=0;**

**y(0)=0;**

**int P=0;**

**for(int i=0;i<m;i++)**

**{**

**h=(trans(x)\*X[i])(0); //mnożenie macierzy**

**h=1.0/(1.0+exp(-h)); //h\_teta(x\_i)**

**y=y-Y(0,i)\*log(h)-(1-Y(0,i))\*log(1-h);**

**if(h>=0.5&&Y(0,i)==1)P++; //warunek dla P to h>=0.5**

**if(h<=0.5&&Y(0,i)==0)P++; //warunek dla P to h>=0.5**

**cout<<"\nIteracja : "<<i<<endl;**

**cout<<"h: "<<h<<endl;**

**cout<<"y: "<<y<<endl;**

**}**

**y(0)=y(0)/m; // Wyliczenie J(teta) czyli y(0) z main**

**cout<<"P: "<<P<<endl;**

**f\_calls++;**

**}**

**void solution::grad(matrix O)**

**{**

**//ODCZYTYWANIE DANYCH Z PLIKU**

**int m=100;**

**int \*n=get\_size(x);**

**static matrix X(n[0],m),Y(1,m);**

**if(solution::g\_calls==0)**

**{**

**ifstream S("XData.txt");**

**S>>X;**

**S.close();**

**S.open("YData.txt");**

**S>>Y;**

**S.close();**

**}**

**double h=0;**

**g=matrix(n[0],1);**

**for(int j=0;j<n[0];++j)**

**{**

**for(int i=0;i<m;++i)**

**{**

**h=(trans(x)\*X[i])(0);**

**h=1.0/(1+exp(-h));**

**g(j)=g(j)+X(j,i)\*(h-Y(0,i));**

**}**

**g(j)=g(j)/m; //POCHODNE J(teta)**

**}**

**/\***

**g = matrix(2,1);**

**g(0) = 10 \* x(0) + 8 \* x(1) - 34;**

**g(1) = 8 \* x(0) + 10 \* x(1) - 38;**

**\*/**

**++g\_calls;**

**}**

**void solution::hess(matrix O)**

**{**

**H = matrix(2, 2);**

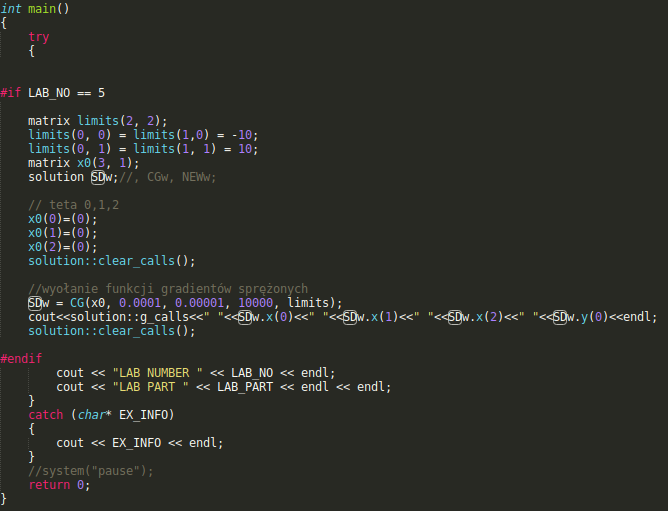
**H(0,0) = H(1,1) = 10;**

**H(0, 1) = H(1, 0) = 8;**

**++H\_calls;**

**}**

*Funkcja main dla problemu rzeczywistego:*

**