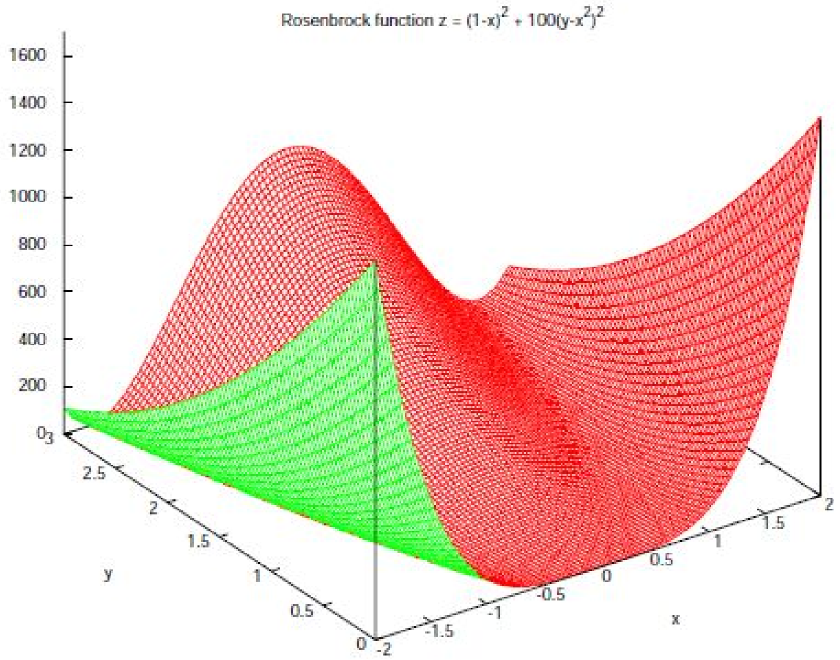
METODY NUMERYCZNE

ZADANIE 17

Artur Guniewicz

17 O. Znajdź numerycznie (analitycznie zrobić można to bardzo łatwo) minimum funkcji Rosenbrocka (zobacz rysunek)



$$f(x,y) = (1-x)^2 + 100(y-x^2)^2 \,. \tag{14}$$

Rozpocznij poszukiwania od kilku–kilkunastu różnych, losowo wybranych punktów i oszacuj, ile trzeba kroków aby zbliżyć się do minimum narozsądną odległość. Przedstaw graficznie drogę, jaką przebywa algorytm poszukujący minimum (to znaczy pokaż położenia kolejnych minimalizacji kierunkowych lub kolejnych zaakceptowanych kroków wykonywanych w metodzie Levenberga–Marquardta).

Metoda: gradienty sprzężone biblioteki Scipy + inne

Program oparty jest o metodę gradientów sprzężonych biblioteki Scipy. Umożliwia ona znalezienie miniumum funkcji Rosenbrocka. Dodatkowo algorytm szukania minimum korzysta z wariantu metody Fletchera-Reevesa (do obliczenia beta - metoda Polaka Ribiere'a), która wyznacza minimum funkcji wielu w wielu wymiarach.

Algorytm wygląda następująco:

$$
\begin{aligned}
Majac\ dany\ punkt\ p:\\
Oblicz\ f_0 = f(x_0)\ \nabla f_0 = \nabla f(x_0)\\
p_0 = -\nabla f_0\ k = 0\\
Dopoki\ \nabla f_k \neq 0:\\
Oblicz\ \alpha_k\ i\ ustaw\ x_{k+1} = x_k + \alpha_k * p_k\\
Oblicz\ \nabla f_{k+1}\\
\beta_{k+1} = \frac{\nabla f_{k+1}^T(\nabla f_{k+1} - \nabla f_k)}{||\nabla f_k||^2}\\
p_{k+1} = -\nabla f_{k+1} + \beta_{k+1} p_k\\
k = k + 1
\end{aligned}
$$

Aby dokonać minimalizacji kierunkowej należy znaleźć alfa. Teraz wyliczony krok alfa będzie szukany dopóki nie będą spełnione dwa warunki Wolfe'a:

A step length α_k is said to satisfy the *Wolfe conditions*, restricted to the direction $\mathbf{p}_k$, if the following two inequalities hold:

i) $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_k \mathbf{p}_k^{\mathrm{T}} \nabla f(\mathbf{x}_k)$,

ii) $-\mathbf{p}_k^{\mathrm{T}} \nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq -c_2 \mathbf{p}_k^{\mathrm{T}} \nabla f(\mathbf{x}_k)$,

Kolejne alfa obliczane będą algorytmem More'a i Thuente'a.

Kod programu:

```
#
#*************************************************************
#*                                                           *
#*                Artur Guniewicz - Zadanie 17               *
#*                                                           *
#*************************************************************
#


# minimalizacja metodą gradientow sprzezonych

import numpy                                  # operacje macierzowe
from scipy.optimize import minimize           # metoda szukania minimum
import random                                 # losowanie
import matplotlib.pyplot as plt               # wykres
from mpl_toolkits.mplot3d import Axes3D       # wykres 3d
from matplotlib import cm                     # kolorowanie wykresu
import sys                                    # argumenty wywolania
programu


# zwraca wartosc funkcji rosenbrocka
def rosenbrock(p):
   return((1-p[0])*(1-p[0]) + 100*(p[1]-p[0]*p[0])*(p[1]-p[0]*p[0]))


# dodaje punkt do listy krokow metody CG
def add_point(x):
   global steps
   steps.append(x)


# globalna zmienna pokazujaca kolejne kroki minimalizacji
steps=[]

if __name__ == "__main__":
   if(len(sys.argv)!=4):
       print("Opcje wywolania programu")
```

```
        print("<nazwa> <liczba punktow do wylosowania> <granica> 
<tryb>")
        print("<granica>- w jakich granicach program ma rysowac wykres-
na przyklad 20 da granice [-20,20]")
        print("<tryb> calc-tylko wylicza minima bez szkicowania 
wykresu")
        print("<tryb> show-wylicza minima i szkicuje droge 
minimalizacji(tylko do 10 punktow)")


   # 10 punktow maksymalnie dla trybu rysowania
   ile_punktow=int(sys.argv[1])
   border=int(sys.argv[2])

   # przygotowanie do wykresu
   fig = plt.figure()
   ax = fig.add_subplot(111, projection='3d')

   # rysowanie funkcji rosenbrocka
   if sys.argv[3]=="show":
       X = numpy.linspace(-border,border,1000)
       Y = numpy.linspace(-border,border,1000)
       X, Y = numpy.meshgrid(X, Y)
       Z = rosenbrock([X,Y])
       surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, linewidth=0, 
antialiased=False)

color=['black','green','gray','red','cyan','magenta','yellow','white','
blue','o range']

   for i in range(ile_punktow):
       # wylosowanie punktu

p=[random.uniform(-border,border),random.uniform(-border,border)]
       print(f"Wylosowano punkt ({p[0]},{p[1]})")
       steps.append(p)

       # zastosowanie minimalizacji metoda gradientow sprzezonych
       found=minimize(rosenbrock,p,method='CG',callback=add_point)
```

```
        print(f"Znalezione minimum ({found.x[0]}, {found.x[1]}) wartosc
funkcji wynosi= {rosenbrock(found.x)}, liczba iteracji={len(steps)}")
        print()

        # dodawanie sciezki do wykresu
        if(sys.argv[3]=="show"):
            temp_x=[i[0] for i in steps]
            temp_y=[i[1] for i in steps]
            temp_z=[rosenbrock([temp_x[i],temp_y[i]]) for i in
range(0,len(temp_x))]
            ax.plot3D(temp_x,temp_y,temp_z,color=color[i],
linestyle='dashed', linewidth=2, markersize=12)

        steps.clear()

    # wyswietlenie wykresu 3D
    if sys.argv[3]=="show":
        plt.show()
```

Kompilacja:

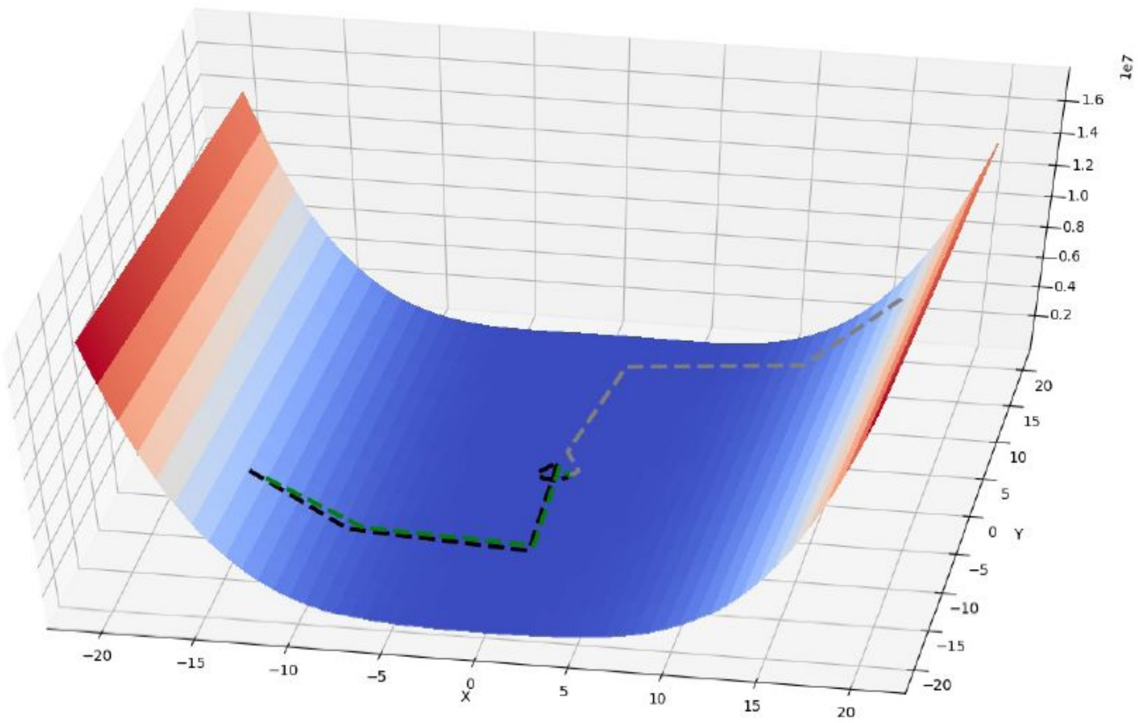
Przykładowe uruchomienia:

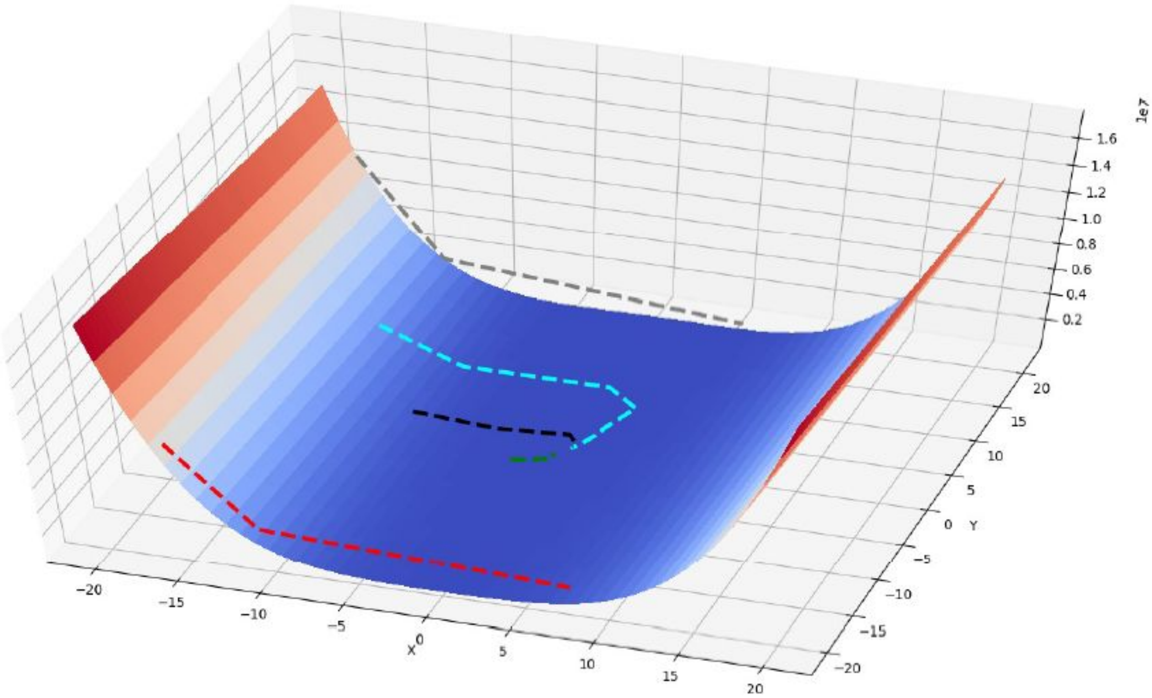
- python3 Zadanie17.py 10 20 show
- python3 Zadanie17.py 10 30 calc

Wyniki:

(dla uruchomienia python3 Zadanie17.py 10 20 calc)

```
Wylosowano punkt (0.914811346185374,5.189783572713932)
Znalezione minimum (0.9999968775579545, 0.9999937530678631) wartosc funkcji wynosi= 9.750067779641259e-12, liczba iteracji=22

Wylosowano punkt (-14.408474518059325,15.439781324822931)
Znalezione minimum (0.9999835907913268, 0.9999671354130297) wartosc funkcji wynosi= 2.694777862940664e-10, liczba iteracji=32

Wylosowano punkt (3.028465080399304,17.382363426246336)
Znalezione minimum (0.9999837428194388, 0.9999674428966894) wartosc funkcji wynosi= 2.644808755672583e-10, liczba iteracji=43

Wylosowano punkt (-13.337414581717688,17.11502048736883)
Znalezione minimum (0.9999923492514012, 0.9999846567812058) wartosc funkcji wynosi= 5.870851205259977e-11, liczba iteracji=42

Wylosowano punkt (-1.758618374965053,17.57093629803935)
Znalezione minimum (-4.118735202907972, 16.943727732837424) wartosc funkcji wynosi= 26.24246418015083, liczba iteracji=4

Wylosowano punkt (-8.1137559962968,-10.63890588933531)
Znalezione minimum (1.049517676707145, 1.1036530843970942) wartosc funkcji wynosi= 0.002921039242713336, liczba iteracji=21

Wylosowano punkt (15.329188831855745,-13.622011788702185)
Znalezione minimum (1.0000020299947239, 1.0000040770022367) wartosc funkcji wynosi= 4.1498080577675375e-12, liczba iteracji=23

Wylosowano punkt (12.377681867626201,-13.919898615421467)
Znalezione minimum (0.9968374019604526, 0.9936426333232372) wartosc funkcji wynosi= 1.0179879381486744e-05, liczba iteracji=14

Wylosowano punkt (18.049945873455798,2.7189850321869855)
Znalezione minimum (-4.9117537242787215, 3.5476860606916393) wartosc funkcji wynosi= 42378.869732633786, liczba iteracji=3

Wylosowano punkt (6.208523722260434,2.750550339032244)
Znalezione minimum (0.999987334168533, 0.9999974725383121) wartosc funkcji wynosi= 1.607485289746455e-12, liczba iteracji=11
```





Komentarz:

Metoda gradientów sprzężonych jest szybka i dokładna, jednak nie zawsze znajduje minimum globalne. Zdarza się też tak, że zamiast niego znajdzie minimum lokalne. Dokładność tego algorytmu już dla niewielkiej liczby iteracji jest satysfakcjonująca. Metoda ta korzysta z algorytmu More'a i Thuente'a do znalezienia funkcji jednowymiarowej.

Program został napisany w języku Python3.