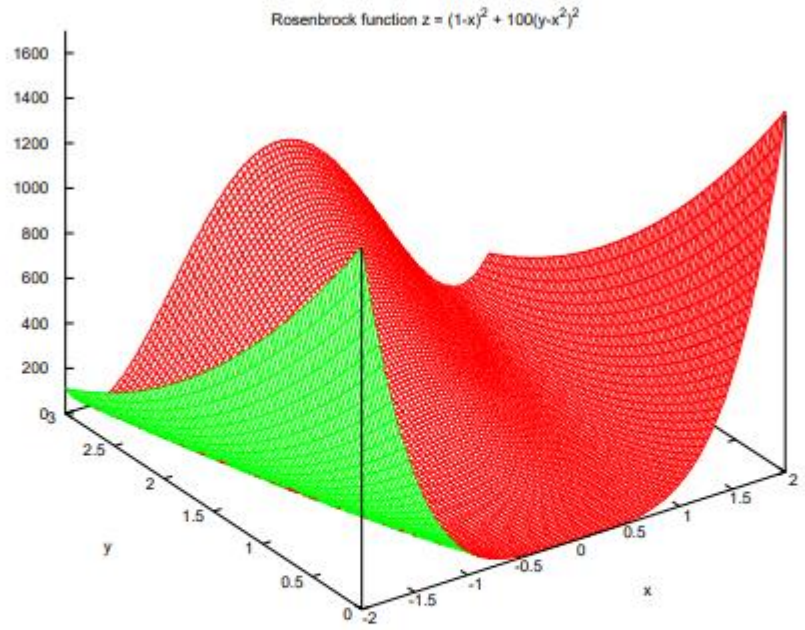
17 O. Znajdź numerycznie (analitycznie zrobić można to bardzo łatwo) minimum funkcji Rosenbrocka (zobacz rysunek)



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

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).

Kod w języku python:

```
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 = []
```

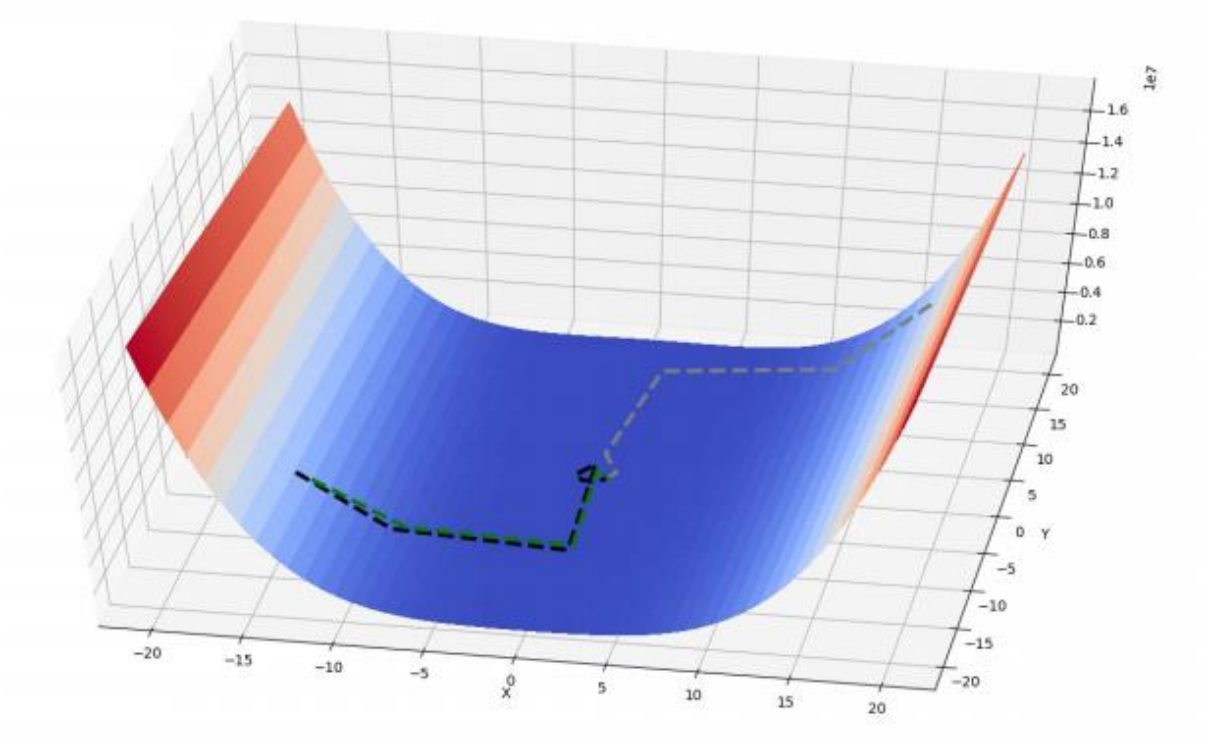
```
if (len(sys.argv) != 3):
    print("Opcje wywolania programu")
    print("<nazwa> <liczba punktow do wylosowania> <granica> <tryb>")
    print("<granica>- w jakich granicach program ma rysowac wykres- na przyklad 20 da
granice [-20,20]")
    exit(0)

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

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

# rysowanie funkcji rosenbrocka
X = numpy.linspace(-granica, granica, 1000)
Y = numpy.linspace(-granica, granica, 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',
'orange']

for i in range(ile_punktow):
    # wylosowanie punktu
    p = [random.uniform(-granica, granica), random.uniform(-granica, granica)]
    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
    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()

plt.show()
```

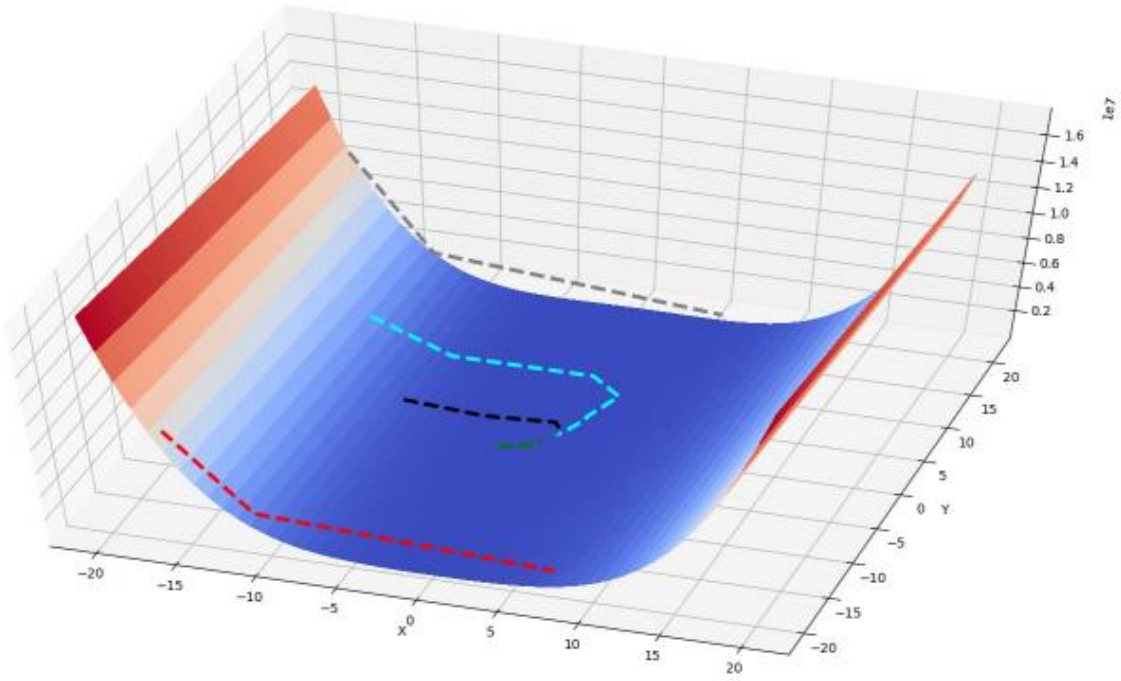
Przykładowe wywołanie programu:

```
$ python Zad17.py 10 20
```

Wyniki:

```
Wylosowano punkt (14.757265376657656,0.4195212631373195)
Znalezione minimum (0.9999955174187333, 0.9999910114736129) wartosc funkcji wynosi= 2.0148215711085998e-11, liczba iteracji=22

Wylosowano punkt (-18.90169605730558,2.811866448334051)
Znalezione minimum (1.0000022056900106, 1.000004437029258) wartosc funkcji wynosi= 4.930831803204015e-12, liczba iteracji=45

Wylosowano punkt (15.21422057011798,-13.81813351404379)
Znalezione minimum (0.9999955150153738, 0.9999910225440908) wartosc funkcji wynosi= 2.0120722260007385e-11, liczba iteracji=29

Wylosowano punkt (18.461442493665203,-11.7353862358293)
Znalezione minimum (0.9999982379284386, 0.99999647384831) wartosc funkcji wynosi= 3.1053008700340315e-12, liczba iteracji=44

Wylosowano punkt (0.8017321391190215,-8.625673990099681)
Znalezione minimum (1.0000011905595032, 1.0000024175047213) wartosc funkcji wynosi= 1.5498136400589799e-12, liczba iteracji=21

Wylosowano punkt (-0.6523359086649947,2.247747801390254)
Znalezione minimum (0.9999955226937037, 0.9999910325016725) wartosc funkcji wynosi= 2.006292758950243e-11, liczba iteracji=31

Wylosowano punkt (-11.498639134681037,8.621590322372683)
Znalezione minimum (0.9999968254915667, 0.9999936657442362) wartosc funkcji wynosi= 1.009926306842127e-11, liczba iteracji=29

Wylosowano punkt (-5.9801171616820525,12.647439231471338)
Znalezione minimum (0.9999955220447181, 0.9999910352285027) wartosc funkcji wynosi= 2.0059970697151994e-11, liczba iteracji=22

Wylosowano punkt (-0.28033556305557994,14.855571490585248)
Znalezione minimum (0.9999959975536913, 0.999991976893603) wartosc funkcji wynosi= 1.6052809012128602e-11, liczba iteracji=35

Wylosowano punkt (-13.821836121833071,19.34443271022956)
Znalezione minimum (0.9999976274584087, 0.9999952553878292) wartosc funkcji wynosi= 5.628975260567760 4e-12, liczba iteracji=41
```





Omówienie wyników:
Metodą gradientów sprzężonych daje szybkie i dokładne wyniki. Nie zawsze jednak znajduje minimum globalne(w niektórych przypadkach znajduje minimum lokalne – drugi wykres). Algorytm ten pozwala znaleźć minimum w dość małej liczbie iteracji.

Opis metody:
Przy szukaniu minimum funkcji Rosenbrocka skorzystałem z biblioteki SciPy. Algorytm korzysta z wariantu metody Fletchera-Reevesa do wyznaczania minimum funkcji wielowymiarowej.

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

Następnie algorytm wylicza alfę 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 alfy wyliczane są za pomocą algorytmu wyszukiwania More'a i Thuente'a.