Sprawozdanie - Metody numeryczne i optymailzacja

Jakub Andryszczak 259519,
Jakub Żak 244255,
Maciej Cierpisz 249163

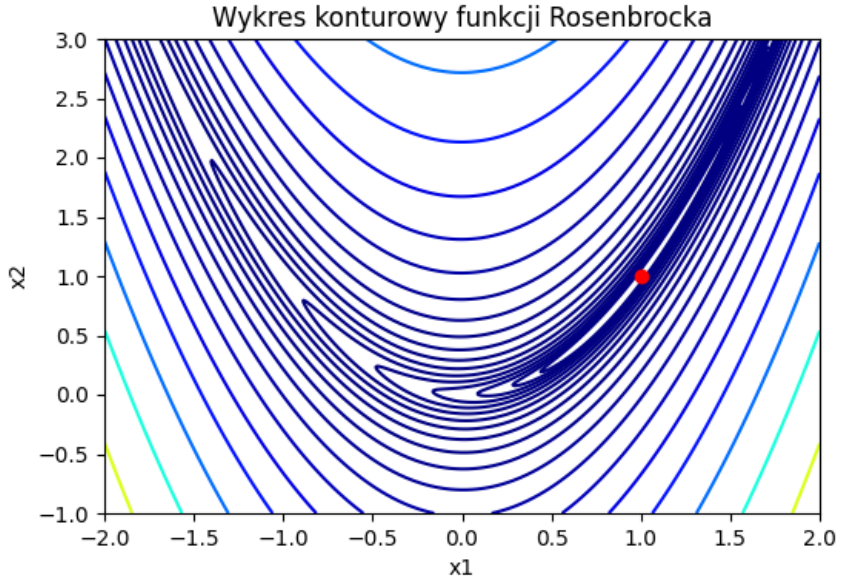
Spis treści

1 Zadanie nr. 1

Sprawdzić warunki optymalizacji pierwszego i drugiego rzędu w punkcie: $x = [11]^T$ funkcji Rosenbrocka: $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$. Narysuj wykres konturowy tej funkcji.

Poniżej algorytm realizujący zadanie:

```
  import numpy as np
import matplotlib.pyplot as plt

# Definicja funkcji Rosenbrocka
def rosenbrock(x):
    return 100 * (x[1] - x[0]**2)**2 + (1 - x[0])**2

# Punkt, w którym sprawdzamy warunki optymalizacji
punkt = np.array([1, 1])

# Gradient funkcji Rosenbrocka
def gradient_rosenbrock(x):
    return np.array([-400 * x[0] * (x[1] - x[0]**2) - 2 * (1 - x[0]),
                     200 * (x[1] - x[0]**2)])

# Sprawdzenie warunków pierwszego ęrzdu
grad = gradient_rosenbrock(punkt)
print(f"Gradient w punkcie {punkt}: {grad}")

# Rysowanie wykresu konturowego
x = np.linspace(-2, 2, 400)
y = np.linspace(-1, 3, 400)
X, Y = np.meshgrid(x, y)
Z = rosenbrock([X, Y])

plt.figure(figsize=(6, 4))
plt.contour(X, Y, Z, levels=np.logspace(-0.5, 3.5, 20), cmap='jet')
plt.plot(1, 1, 'ro') # Punkt optymalny
plt.title('Wykres konturowy funkcji Rosenbrocka')
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
```



Rys. 1.1. Wykres konturowy funkcji.

2 Zadanie nr. 2

Sprawdź warunki optymalizacji pierwszego i drugiego rzędu dla funkcji kwadratowych:

$$a) f(x) = 2x_1^2 - x_1x_2 + \frac{1}{2}x_2^2 - 3x_1 + 3.5, \tag{1}$$

$$a) f(x) = -\frac{3}{2}x_1^2 + x_1x_2 - \frac{1}{2}x_2^2 + 2x_1 - 1 \tag{2}$$

$$a) f(x) = x_1^2 + 8x_1x_2 + \frac{1}{2}x_2^2 - 10x_1 - 9x_2 + \frac{9}{2} \tag{3}$$

Narysuj ich wykresy konturowe. Czy te funkcje są wypukłe? Jaki rodzaju stacjonarność występuje w tych funkcjach?

a) do sprawdzenia optymalizacji pierwszego rzędu dla funkcji kwadratowej należy wyliczyć gradient tej funkcji.

$$\nabla f(x) = \begin{bmatrix} 4x_1 - x_2 - 3 \\ -x_1 + x_2 \end{bmatrix} \tag{4}$$

Następnie wyliczony wartoci wektora x. Do tego należy $\nabla f(x^*) = 0$.

$$\begin{cases} 4x_1 - x_2 - 3 = 0 \\ -x_1 + x_2 = 0 \end{cases} \tag{5}$$

Po przeliczeniu wyszło, że wektor dla tej funkcji jest równy:

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{6}$$

Do policzenia drugiego rzędu należy wyliczyć hesian tej funkcji. Pozwoli to nam wyznaczyć czy funkcja jest wypukła.

$$H = \begin{bmatrix} 4 & -1 \\ -1 & 1 \end{bmatrix} \tag{7}$$

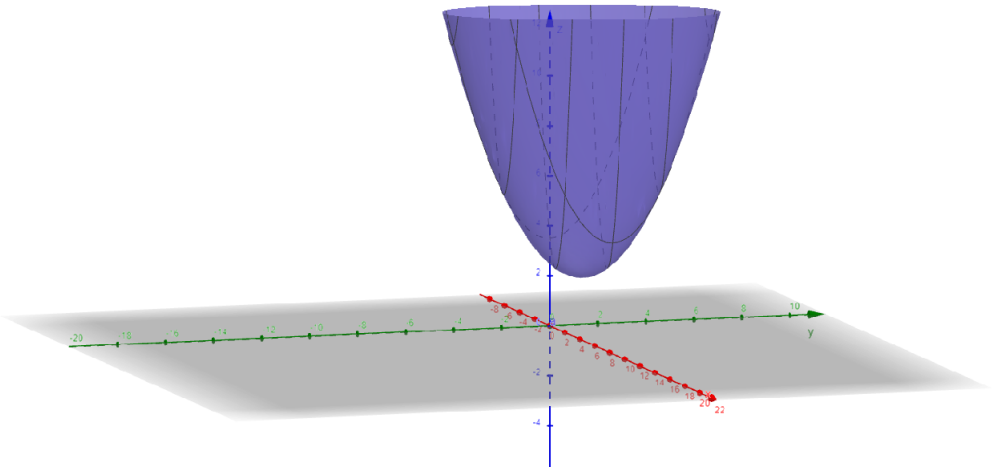
kolejno należy wyliczyć wyznacznik dla $det(H - I\lambda)$.

$$det(H - I\lambda) = (4 - \lambda)(1 - \lambda) - 1 \tag{8}$$

$$\lambda_1 = \frac{5 - \sqrt{13}}{2} \tag{9}$$

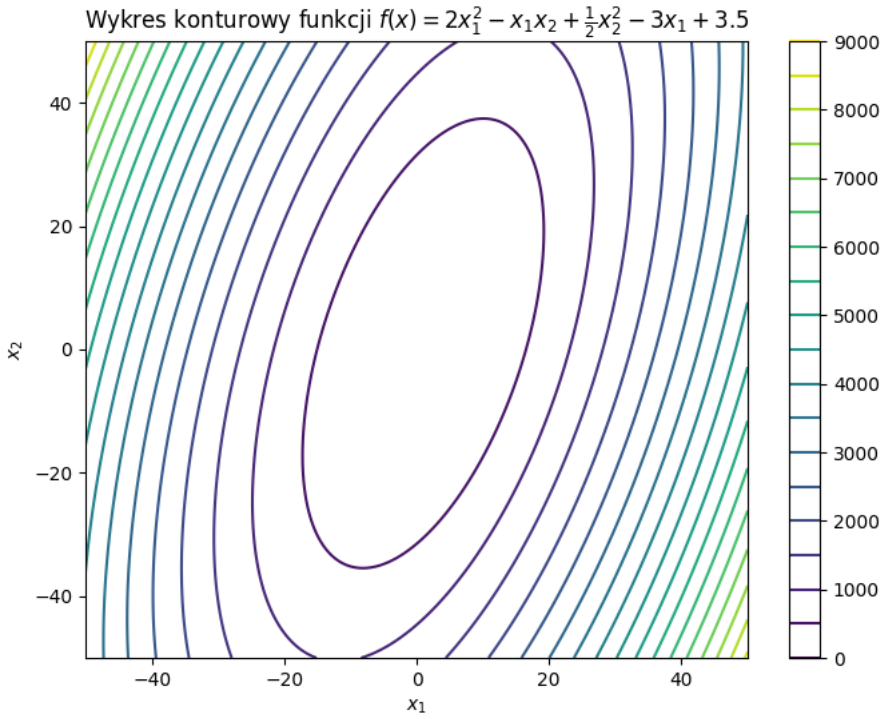
$$\lambda_1 = \frac{5 + \sqrt{13}}{2} \tag{10}$$

Obie wartości są dodatnie, a zatem funkcja należy do wypukłych. Wykres Wygenerowany w Geogebrze wygląda następująco.



Rys 2.1 Wykres 3d funkcji podpunktu a)

Wykres konturowy:



Rys 2.2 Wykres konturowy funkcji podpunktu a)

Dla podpunktu b) i c) wykonano podobne obliczenia dlatego pominięto rozpisywanie wszystkiego krok po kroku. b)

$$\nabla f(x) = \begin{bmatrix} -3x_1 + x_2 + 2 \\ x_1 - x_2 \end{bmatrix} \tag{11}$$

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{12}$$

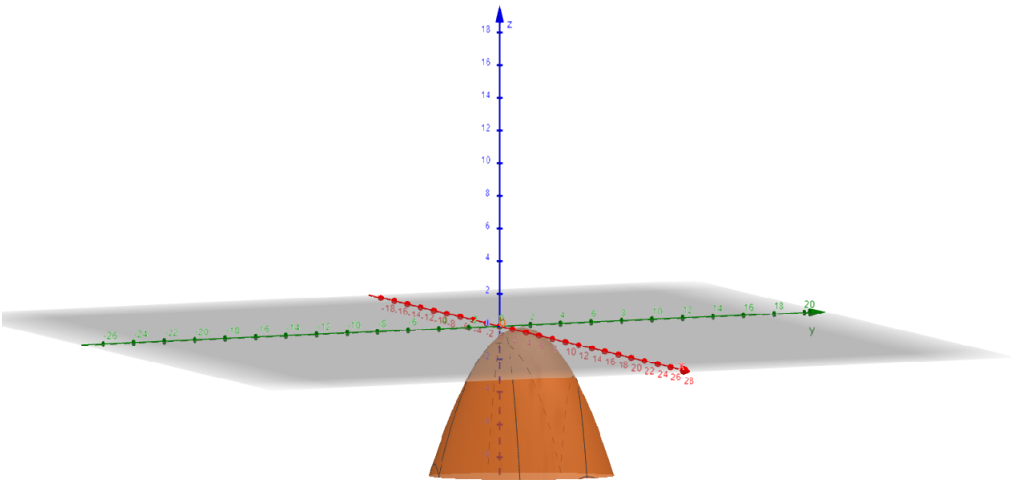
$$H = \begin{bmatrix} -3 & 1 \\ 1 & -1 \end{bmatrix} \tag{13}$$

$$det(H - I\lambda) = \lambda^2 + 4\lambda + 2 \tag{14}$$

$$\lambda_1 = -2 + \sqrt{2} \tag{15}$$

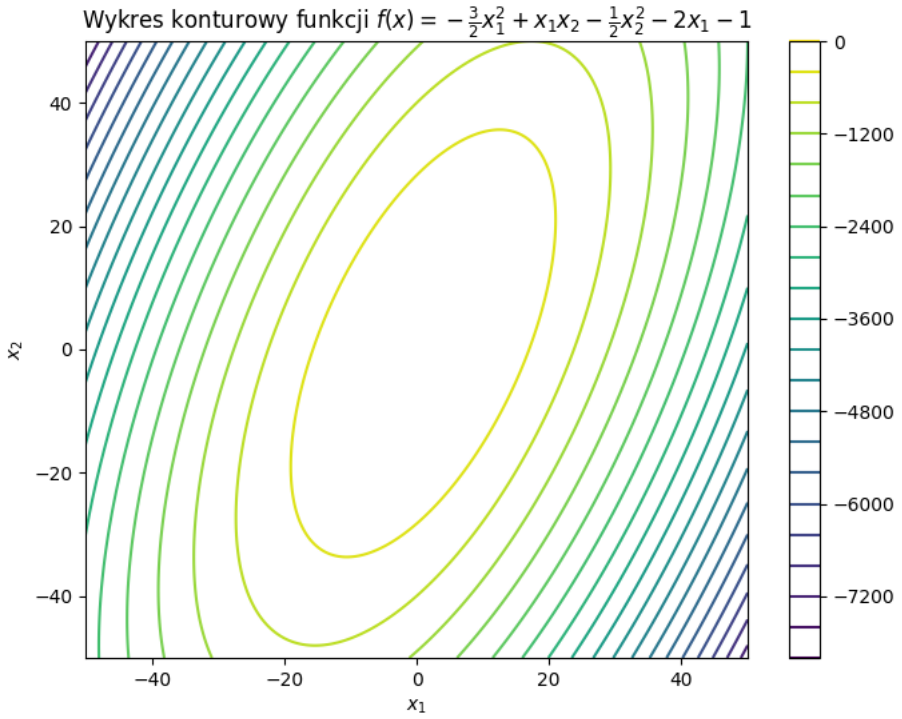
$$\lambda_2 = -2 - \sqrt{2} \tag{16}$$

W tym przypadku obie lambdy są ujemne, a zatem funkcja jest funkcją wklęsłą.



Rys 2.3 Wykres 3d funkcji podpunktu b)

Wykres konturowy:



Rys 2.4 Wykres konturowy funkcji podpunktu b)

c)

$$\nabla f(x) = \begin{bmatrix} 2x_1 + 8x_2 - 10 \\ 8x_1 + x_2 - 9 \end{bmatrix} \tag{17}$$

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{18}$$

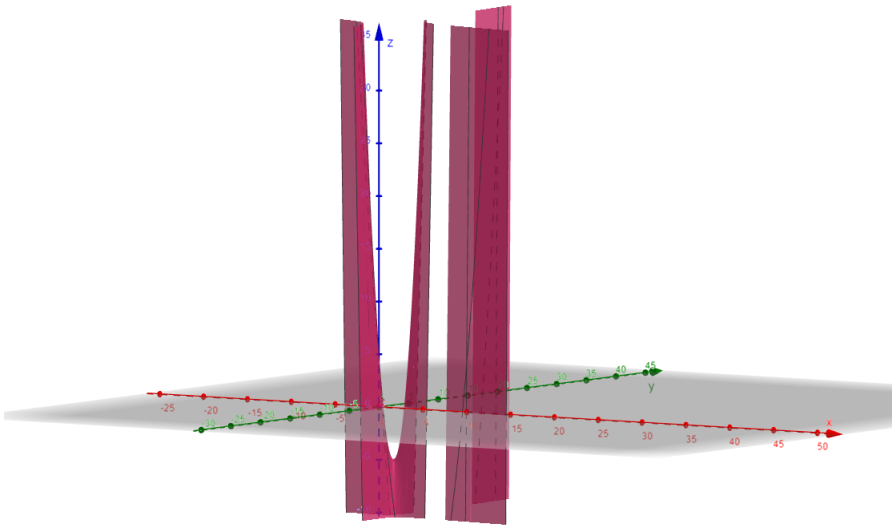
$$H = \begin{bmatrix} 2 & 8 \\ 8 & 1 \end{bmatrix} \tag{19}$$

$$det(H - I\lambda) = \lambda^2 - 3\lambda - 62 \tag{20}$$

$$\lambda_1 = \frac{3 - \sqrt{257}}{2} \tag{21}$$

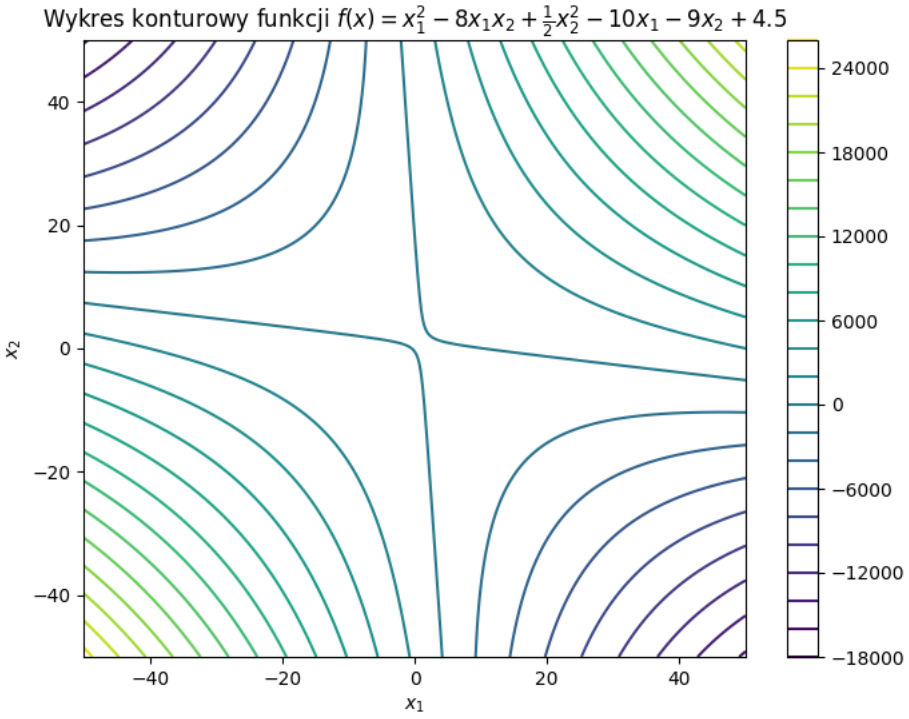
$$\lambda_2 = \frac{3 + \sqrt{257}}{2} \tag{22}$$

W tym wypadku rozwiązania wychodzą jedna ujemna i jedna nieujemna. Wskazuje to na to, że funkcja nie ma globalnego minimum ani maksimum. Kształtem przypomina siodło, stąd nazwa "punkt siodłowy"



Rys 2.5 Wykres 3d funkcji podpunktu c)

.



Rys 2.6 Wykres konturowy funkcji podpunktu c)

3 Zadanie nr. 3

Dla funkcji kwadratowej:

$$f(x) = (x^T)Gx, gdzie\, G = \begin{bmatrix} \alpha & 1 & 1 \\ 3 & 2 & 1 \\ 2 & 2 & 3 \end{bmatrix} \tag{23}$$

Określić parametr, dla którego ta funkcja jest ściśle wypukła.

Funkcja f(x) jest ściśle wypukła, jeżeli wszystkie minory główne macierzy mają dodatnie wyznaczniki. A zatem, jeżeli:

$$det[\alpha] > 0 \tag{24}$$

$$det \begin{bmatrix} \alpha & 1 \\ 3 & 2 \end{bmatrix} > 0 \tag{25}$$

$$det \begin{bmatrix} \alpha & 1 & 1 \\ 3 & 2 & 1 \\ 2 & 2 & 3 \end{bmatrix} > 0 \tag{26}$$

$$\begin{cases} \alpha > 0 \\ \alpha \cdot 2 - 1 \cdot 3 > 0 \\ \alpha \cdot 2 \cdot 3 + 3 \cdot 2 \cdot 1 + 2 \cdot 1 \cdot 1 - 1 \cdot 2 \cdot 2 - 1 \cdot 2 \cdot \alpha - 3 \cdot 1 \cdot 3 > 0 \end{cases} \tag{27}$$

$$\begin{cases} \alpha > 0 \\ 2\alpha > 3 \\ 4\alpha > 5 \end{cases} \tag{28}$$

$$\begin{cases} \alpha > 0 \\ \alpha > \frac{3}{2} \\ \alpha > \frac{5}{2} \end{cases} \tag{29}$$

Liczba, która spełnia wszystkie trzy warunki to $\frac{2}{3}$, a zatem funkcja będzie ściśle wypukła, jeżeli wartość parametru α będzie większa niż $\frac{3}{2}$.

4 Zadanie nr. 4

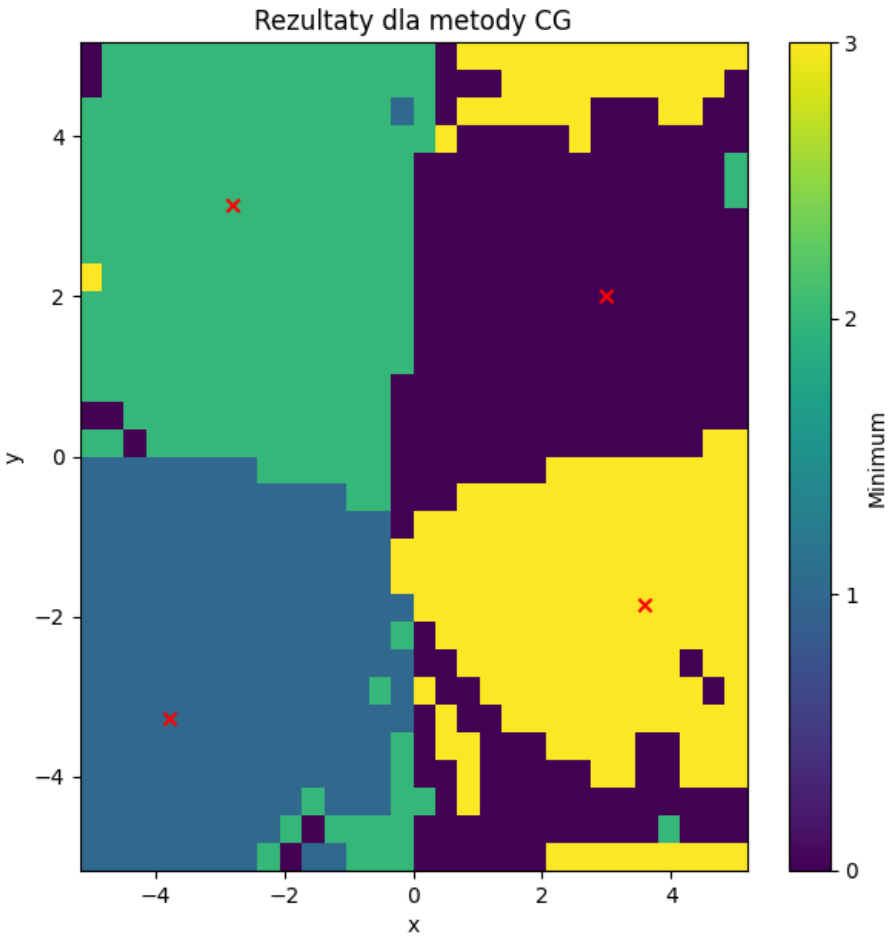
Funkcja Himmelblaua $f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$ ma cztery odrębne minima: $f(3; 2) = 0, f(-3.78; -3.28) = 0,0054, f(-2.81; 3.13) = 0,0085, f(3.58; -1.85) = 0,0011$ Zbadaj do których minimów zbiegają się algorytmy gradientowy i quasi-Newtona, gdy są zainicjalizowane w punktach (x_i, y_j), gdzie: $x_i = -5 + 10(\frac{i-1}{29}), y_j = -5 + 10(\frac{j-1}{29}), j = 1, ..., 30, i = 1, ..., 30$. Są to

punktu tworzące jednolitą siatkę prostokątną. Sklasyfikuj każdy punkt inicjalizacji według minimów, do których zbiegają się wybrane algorytmy, tworząc kolorową mapę punktów, gdzie kolory oznaczałyby minima, do których zbiega się analizowany algorytm z danego punktu inicjalizacyjnego. Wykonaj mapy punktów oddzielnie dla każdego algorytmu.

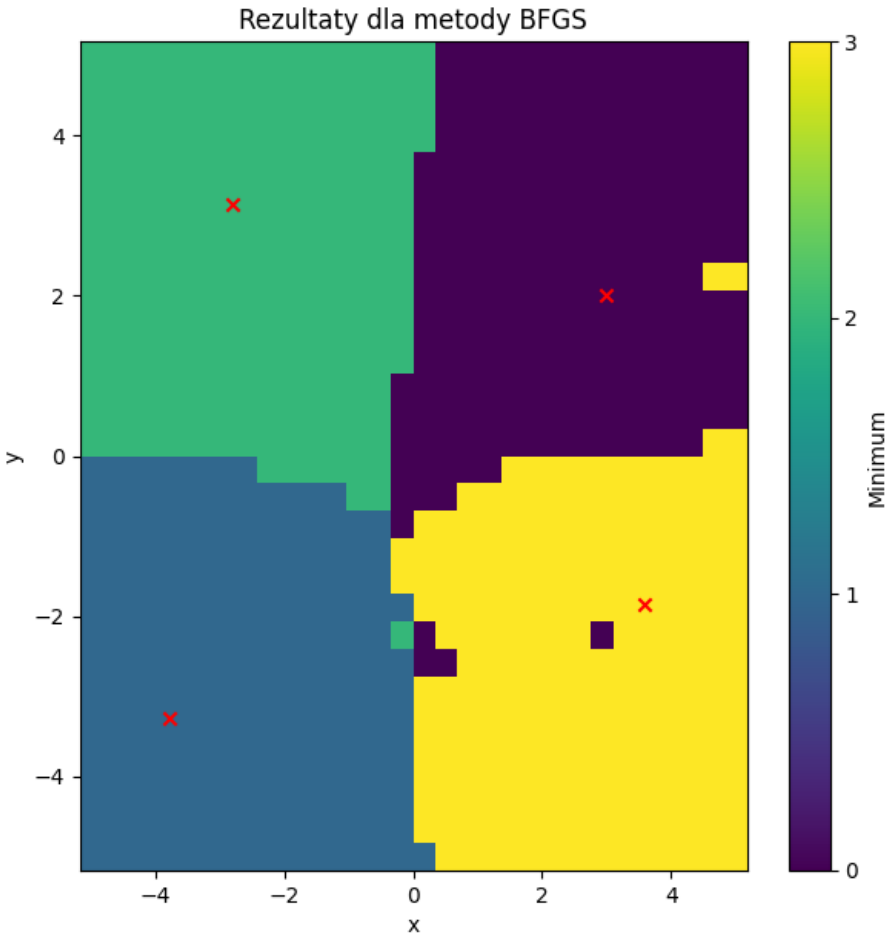
Poniżej implementacja rozwiązania w języku python:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# Definicja funkcji Himmelblaua
def himmelblau(x):
    return (x[0]**2 + x[1] - 11)**2 + (x[0] + x[1]**2 - 7)**2

# Znane minima
minima = np.array([
    [3, 2],
    [-3.78, -3.28],
    [-2.81, 3.13],
    [3.58, -1.85]
])

# Generowanie punktów siatki startowej
x = np.linspace(-5, 5, 30)
y = np.linspace(-5, 5, 30)
X, Y = np.meshgrid(x, y)
start_points = np.c_[X.ravel(), Y.ravel()]

# Algorytmy do zastosowania
methods = ['BFGS', 'CG']

# Funkcja do przypisania wyników do najbliższego znanego minimum
def classify_minimum(result, minima):
    distances = np.linalg.norm(minima - result, axis=1)
    return np.argmin(distances)

# Przeprowadzenie optymalizacji
results = {method: np.zeros_like(X, dtype=int) for method in methods}

for method in methods:
    for idx, point in enumerate(start_points):
        res = minimize(himmelblau, point, method=method)
        classified_min = classify_minimum(res.x, minima)
        results[method].ravel()[idx] = classified_min

# Wizualizacja wyników osobno dla każdej metody
for method in methods:
    plt.figure(figsize=(7, 7))
```

```
c = plt.pcolormesh(X, Y, results[method], shading='auto')
plt.scatter(minima[:, 0], minima[:, 1], c='red', marker='x')
plt.title(f'Rezultaty dla metody {method}')
plt.xlabel('x')
plt.ylabel('y')
plt.colorbar(c, ticks=[0, 1, 2, 3], label='Minimum')
plt.show()
```



Rys. 4.1. Mapa punktów dla algorytmu gradientowego.



Rys. 4.2. Mapa punktów dla algorytmu quasi-Newtona.

5 Zadanie nr. 5

Użyj algorytmu najszybszego spadku do znalezienia minimum funkcji Rosenbrocka. Najpierw wypróbuj punkt początkowy x_0=[1.2 1.2]^T a następnie punkt x_0=[-1.2 1]^T.

Funkcja Rosenbrocka jest to funkcja niewypukła używana w optymalizacji jako test dla algorytmów optymalizacji. Zwana jest też ze względu na swój kształ "doliną Rosenbrocka" lub "funkcją bananową Rosenbrocka". Funkcja ta

w dwóch wymiarach prezentuje się następująco:

$$f(x) = (a - x)^2 + b(y - x^2)^2 \tag{30}$$

Najczęściej stosuje się wartości parametrów a = 1 i b = 100.

Algorytm najszybszego spadku iteracyjnie aktualizuje punkt startowy w kierunku przeciwnym do gradientu. Aktualizacja punktu wykonuje się według wzoru:

$$x_{k+1} = x_k - \alpha \nabla f(x_k) \tag{31}$$

gdzie α to krok. Do przeliczenia wartości w podanych punktach wykorzystano zaimplementowany kod w pythonie.

```
  def rosenbrock(x):
    return 100 * (x[1] - x[0]**2)**2 + (1 - x[0])**2

# Define the gradient of the Rosenbrock function
def rosenbrock_grad(x):
    dfdx1 = -400 * x[0] * (x[1] - x[0]**2) - 2 * (1 - x[0])
    dfdx2 = 200 * (x[1] - x[0]**2)
    return np.array([dfdx1, dfdx2])

# Steepest descent algorithm
def steepest_descent(rosenbrock, rosenbrock_grad, x0,
     learning_rate=0.001, max_iter=10000, tol=1e-6):
    x = x0
    for i in range(max_iter):
        grad = rosenbrock_grad(x)
        x_new = x - learning_rate * grad
        if np.linalg.norm(x_new - x) < tol:
            break
        x = x_new
    return x, rosenbrock(x), i

# Initial points
initial_points = [np.array([1.2, 1.2]), np.array([-1.2, 1.0])]

# Perform steepest descent for each initial point
results = []
for x0 in initial_points:
    result = steepest_descent(rosenbrock, rosenbrock_grad, x0)
    results.append(result)
```

Wartości w obu przypadkach zbiegają do punkty minimalnego jaki występuje w tej funkcji (x=$[1\ 1]^T$) oraz do wartości (f(x)=0). Ze względu na to, że punkt pierwszy znajduje się bliżej finalnego punktu, liczba iteracji była mniejsza.

Poniżej przedstawiono wykres konturowy dla danej funkcji z zaznaczonymi punktami.

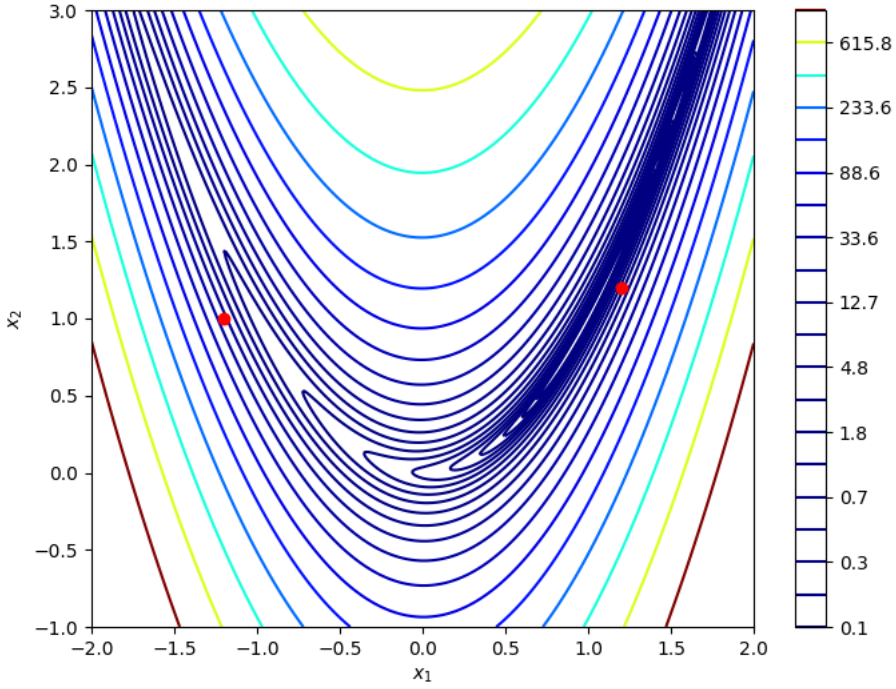
Punkty początkowe	[1.2 1.2]	[-1.2 1]
Znaleziony minimalny punkt	[1.00111864 1.00224301]	[0.99888303 0.99776284]
Wartość funkcji w punkcie	1.2533645426010813e-06	1.2496137290730113e-06
Liczba iteracji	11972	14789

6 Zadanie nr. 6

Zadana jest funkcja Rosenbrocka: $f(x) = 100(x_2 - x_1^2)^2 + (1 - x)^2$, narysuj jej izolinie, a następnie znajdź jej lokalne minimum przy pomocy metod Fletchera-Reevesa i Polaka-Ribiere'a, zaczynając od punktu początkowego: $x_0 = \begin{bmatrix} -1.2 & 1 \end{bmatrix}^T$.

Na potrzeby tego zdania wykorzystano algorytmy Fletchera-Reevesa oraz Polaka-Ribiere'a. Listing kodu przedstawiono poniżej.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from scipy.optimize import minimize

# Definicja funkcji Rosenbrocka
def rosenbrock(x):
    return 100*(x[1] - x[0]**2)**2 + (1 - x[0])**2

# Wygenerowanie siatki punktow łwoko obszaru, w którym chcemy
    narysowac izolinie
x = np.linspace(-2, 2, 100)
y = np.linspace(-1, 3, 100)
X, Y = np.meshgrid(x, y)
Z = rosenbrock([X, Y])

# Narysowanie izolinii funkcji Rosenbrocka
plt.figure()
plt.contour(X, Y, Z, levels=np.logspace(0, 3, 20), norm=LogNorm(),
    cmap=plt.cm.jet)
plt.colorbar(label='Wartosc funkcji Rosenbrocka')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Izolinie funkcji Rosenbrocka')
plt.show()

# Definicja punktu początkowego
x0 = np.array([-1, 2])

# Implementacja algorytmu optymalizacji z metodą Fletchera-Reevesa
result_fr = minimize(rosenbrock, x0, method='CG', jac=None, tol=1e-6)
```



Rys 5 Wykres konturowy dla funkcji Rosenbrocka

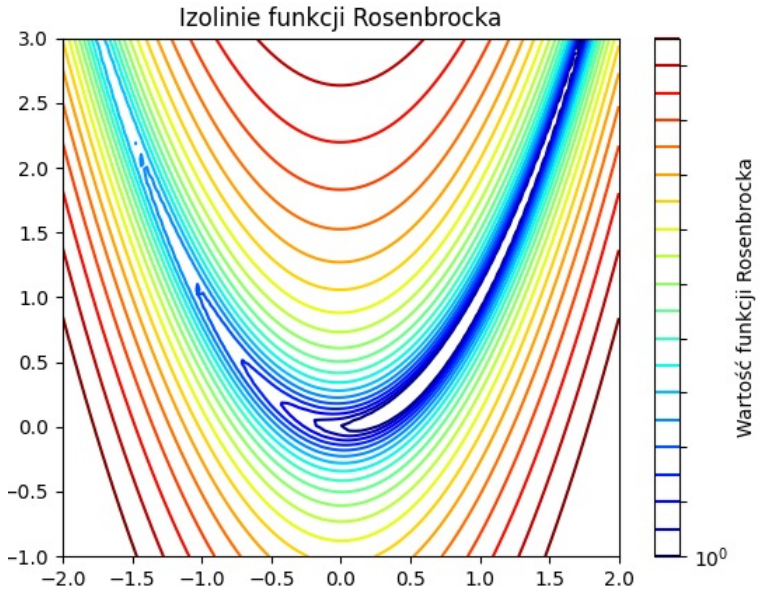
```
print("Minimum (Fletchera-Reevesa):", result_fr.x)

# Implementacja algorytmu optymalizacji z metodą Polaka-Ribiere'a
result_pr = minimize(rosenbrock, x0, method='BFGS', jac=None, tol=1e-6)
print("Minimum (Polaka-Ribiere'a):", result_pr.x)
```

W przypadku metody Fletchera-Reevesa, wykorzystano method='CG', gdzie CG oznacza metodę gradientową, która wykorzystuje algorytm sprzężonych gradientów (CG), który jest implementacją metody Fletchera-Reevesa.

Natomiast w przypadku metody Polaka-Ribiere'a, wykorzystano method='BFGS', gdzie BFGS oznacza metodę quasi-Newtona, która wykorzystuje algorytm Quasi-Newtona, w tym przypadku metoda Polaka-Ribiere'a.

Poniżej przedstawiono wynik działania kodu

```
Minimum (Fletchera-Reevesa): [0.99999552 0.99999102]
Liczba iteracji (Fletchera-Reevesa): 34

Minimum (Polaka-Ribiere'a): [0.99999552 0.99999104]
Liczba iteracji (Polaka-Ribiere'a): 37
```



7 Zadanie nr. 7

Korzystając z algorytmów takich jak quasi-Newton oraz CG, rozwiązać układ równań liniowych: Ax = b, gdzie $A = [a_{ij}] \epsilon \Re^{NxN}$, $a_{ij} = \frac{1}{i+j-1}$ jest macierzą Hilberta, $b = [1, ..., 1] \epsilon \Re^N$. Rozpocząć iterację od $x_0 = 0$. Jeśli nie jest to określone przez algorytm, do pomiaru błędu residualnego należy użyć odległości euklidesowej. Wypróbować wymiary $N = 5, 8, 12, 20$ podać liczbę iteracji potrzebnych do zmniejszenia błędu residualnego poniżej 10^{-6}.

Na potrzeby tego zdania wykorzystano kod przedstawiony na listingu poniżej.

```
import numpy as np
from scipy.optimize import minimize
from scipy.sparse.linalg import cg

def hilbert_matrix(n):
    """
    Funkcja tworząca macierz Hilberta o rozmiarze nxn.
    """
    return np.array([[1/(i + j - 1) for j in range(1, n+1)] for i in
        range(1, n+1)])
```

```python
def residual_error(x, A, b):
    """
    Oblicza blad residualny ||Ax - b||.
    """
    return np.linalg.norm(np.dot(A, x) - b)

# Wymiary macierzy Hilberta
N_values = [5, 8, 12, 20]

for N in N_values:
    A = hilbert_matrix(N)
    b = np.ones(N)
    x0 = np.zeros(N)

    # Metoda quasi-Newtona (BFGS)
    result_bfgs = minimize(residual_error, x0, args=(A, b),
        method='BFGS', tol=1e-6)
    iterations_bfgs = result_bfgs.nit

    # Metoda gradientów sprzezonych (CG)
    x_cg, info_cg = cg(A, b, x0=x0, tol=1e-6)
    iterations_cg = 0 if isinstance(info_cg, int) else len(info_cg)
    residual_cg = np.linalg.norm(np.dot(A, x_cg) - b) if
        isinstance(info_cg, int) else info_cg

    print(f"Rozmiar macierzy: {N}")
    print(f"Liczba iteracji (BFGS): {iterations_bfgs}")
    print(f"Liczba iteracji (CG): {iterations_cg}")
    print(f"Residuum (CG): {residual_cg}")
```

Wyniki działania kodu przedstawiono poniżej.

rozmiar macierzy	iteracje(BFGS)	residuum(BFGS)	iteracje(CG)	residuum(CG)
5	20	1.0747	0	2.62E-09
8	20	1.6587	0	1.89E-08
12	20	2.3271	0	6.53E-07
20	20	3.4078	0	4.26E-06

Metoda BFGS (quasi-Newton) zakończyła się dla każdego rozmiaru macierzy po 20 iteracjach. Liczba iteracji dla metody CG (gradientów sprzężonych) wynosiła 0 dla każdego rozmiaru macierzy, co sugeruje, że metoda osiągnęła wymaganą dokładność w jednej iteracji. Residuum dla metody CG było bardzo niskie dla każdego rozmiaru macierzy, co wskazuje na dobrą jakość rozwiązania. Residuum dla metody BFGS było większe niż dla metody CG, co jest zrozumiałe, ponieważ metoda BFGS nie działa bezpośrednio na residuum.