Sprawozdanie - Metody numeryczne i optymailzacja

Jakub Andryszczak 259519,
Jakub Żak 244255,
Maciej Cierpisz 249163

Spis treści

1 Zadanie nr. 1

2 Zadanie nr. 2

3 Zadanie nr. 3

Dla funkcji kwadratowej:

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

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:

$$det[\alpha] > 0 \tag{2}$$

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

$$det \begin{bmatrix} \alpha & 1 & 1 \\ 3 & 2 & 1 \\ 2 & 2 & 3 \end{bmatrix} > 0 \tag{4}$$

$$\begin{cases} \alpha > 0 \\ \alpha \cdot 2 - 1 \cdot 3 > 0 \\ \alpha \cdot 2 \cdot 3 + 3 \cdot 2 \cdot 1 + 2 \cdot 1 \cdot 1 - 1 \cdot 2 \cdot 2 - 1 \cdot 2 \cdot \alpha - 3 \cdot 1 \cdot 3 > 0 \end{cases} \tag{5}$$

$$\begin{cases} \alpha > 0 \\ 2\alpha > 3 \\ 4\alpha > 5 \end{cases} \tag{6}$$

$$\begin{cases} \alpha > 0 \\ \alpha > \frac{3}{2} \\ \alpha > \frac{5}{2} \end{cases} \tag{7}$$

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

5 Zadanie nr. 6

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 ławoko 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)
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.

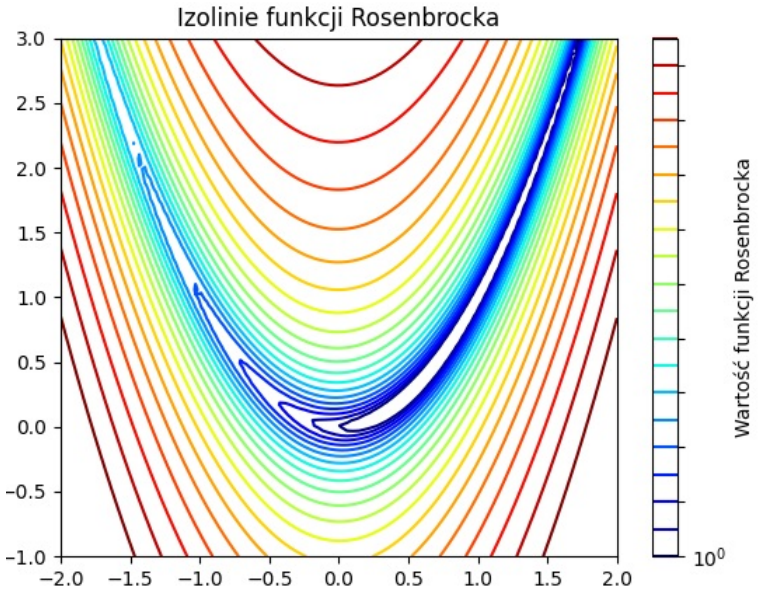
Natomiast w przypadku metody Polaka-Ribiere'a, wykorzystano method='BFGS', gdzie BFGS oznacza metodę quasi-Newtona, która wykorzystuje algorytm Quasi-

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
```



6 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
```

```python
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)])

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.