НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

із лабораторної роботи

із дисципліни «АЛГОРИТМИ І СИСТЕМИ КОМП'ЮТЕРНОЇ МАТЕМАТИКИ 1.МАТЕМАТИЧНІ АЛГОРИТМИ»

на тему:

Чисельні методи оптимізації

Виконав:
студент групи КМ-13
Онищенко В.С.

Перевірила:
асистент кафедри ПМА
Ковальчук-Химюк Л.О.

Київ — 2024 року

Зміст

ВСТУП

Метою роботи є вивчення методів нульового, першого та другого порядку, а також практична мінімізація функцій багатьох змінних з використанням систем комп'ютерної математики (СКМ).

Длч розробки використовувати мови Python та Octave[1]

1 ПОРЯДОК ВИКОНАННЯ РОБОТИ

1) Ознайомитися з методичними вказівками і методичними вказівками до роботи.

2) Отримати варіант індивідуального завдання.

3) Розробити блок-схему алгоритму розв’язання задачі.

4) Написати, налагодити й виконати програму. В програмі передбачити опис усіх використаних масивів.

5) Обчислити точності оцінки алгоритмів по критерію оптимальності:

$$\varepsilon = |f(x_k) - f(x^*)|$$

або по координатам:

$$\delta = \|x_k - x^*\|,$$

де k – задана кількість ітерацій; x^* – точка мінімуму функції.

6) Оформити звіт по роботі.

2 ОСНОВНА ЧАСТИНА

2.1 Вихідні дані

Завдання для заданого варіанту 15 розташоване у таблиці 2.1 Через

Таблиця 2.1 – завдання варіанту 15

Варіант	**Функція**	**Початковий вектор**	**Точка мінімуму**	**Значення**
15	$3x_1^2 - x_1 + x_2^3 - 3x_2^2 - 1$	$[-1;-1]$	$[1/3;2]$	$-47/9$

неправильну роботу завдання у варіанті 15 було обрано завдання варіанту 16, умови для якого розташовані у таблиці 2.1

Таблиця 2.2 – завдання варіанту 16

Варіант	**Функція**	**Початковий вектор**	**Точка мінімуму**	**Значення**
16	$6x_1 + 2x_1^2 - 2x_1x_2 + 2x_2^2$	$[-1;-1]$	$[-2;-1]$	-6

2.2 Типовий перелік вимог до ПЗ

Програмне забезпечення, розроблюване в рамках виконання кожної лабораторної роботи, повинно задовольняти низку вимог, які можна розділити на *обов'язкові* (які ПЗ повинно задовольняти незалежно від лабораторної роботи) та *варіативні* (які для кожної лабораторної роботи унікальні). До обов'язкових вимог належать:

– У програмі повинно бути передбачено перевірки на некоректне введення для всіх полів введення, зокрема:

1) порожнє введення;
2) синтаксично некоректне введення (наприклад, літери в полі для числових коефіцієнтів);
3) уведення спеціальних символів;
4) уведення чисел, які перевищують максимальний розмір для чисел відповідного типу даних (для перевірки на переповнення розрядної сітки).

У випадку некоректного введення повинно з'являтися діалогове вікно з відповідним повідомленням.

– Для всіх полів введення повинно бути визначено гранично допустиму кількість символів (для числових полів — гранично допустимі значення).

– У програмі повинно відслідковуватися переповнення розрядної сітки під час виконання обчислень. У випадку переповнення повинно з'являтися діалогове вікно з відповідним повідомленням.

– У графічному інтерфейсі користувача повинно бути передбачено можливість гнучкого налаштування розмірності розв'язуваної задачі (наприклад, можливість зміни розмірності матриці чи кількості складів у транспортній задачі).

До варіативних вимог належать вимоги щодо перевірки на коректне опрацювання виключних ситуацій, які можуть виникати під час застосування заданого методу до розв’язання поставленої задачі (наприклад, коли сума заявок не збігається з сумою ресурсів у транспортній задачі, нижня межа інтегрування перевищує верхню тощо).

2.3 Опис методів

2.3.1 Метод Нелдера-Міда

Метод Нелдера-Міда є чисельним методом оптимізації, який використовується для мінімізації (або максимізації) нелінійних функцій без обчислення похідних. Основу алгоритму складають геометричні трансформації симплекса — багатокутника у просторі змінних.

Постановка задачі

Нехай потрібно знайти мінімум функції $f(\mathbf{x})$, де $\mathbf{x} \in \mathbb{R}^n$. Для цього використовується набір $n + 1$ точок:

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n+1},$$

які утворюють симплекс у $\mathbb{R}^n$.

Операції над симплексом

Алгоритм передбачає виконання таких операцій:

1) **Рефлексія:** Відображення найгіршої точки $\mathbf{x}_h$ відносно центру симплекса:

$$\mathbf{x}_r = \mathbf{c} + \alpha(\mathbf{c} - \mathbf{x}_h),$$

де $\mathbf{c}$ — центр мас симплекса без точки $\mathbf{x}_h$, а $\alpha > 0$ — коефіцієнт рефлексії.

2) **Розширення:** Якщо $f(\mathbf{x}_r) < f(\mathbf{x}_b)$, де $\mathbf{x}_b$ — найкраща точка, виконується розширення:

$$\mathbf{x}_e = \mathbf{c} + \gamma(\mathbf{x}_r - \mathbf{c}),$$

де $\gamma > 1$ — коефіцієнт розширення.

3) **Стиснення:** Якщо рефлексія не дала значного покращення, використовується стиснення:

$$\mathbf{x}_s = \mathbf{c} + \rho(\mathbf{x}_h - \mathbf{c}),$$

де $\rho \in (0, 1)$ — коефіцієнт стиснення.

4) **Зменшення:** Якщо жодна з операцій не покращила результат, усі точки симплекса зміщуються ближче до найкращої:

$$\mathbf{x}_i \leftarrow \mathbf{x}_b + \sigma(\mathbf{x}_i - \mathbf{x}_b), \quad i = 1, 2, \ldots, n + 1,$$

де $\sigma \in (0, 1)$ — коефіцієнт зменшення.

2.3.2 Метод градієнтного спуску

Метод градієнтного спуску є ітераційним алгоритмом для мінімізації функцій шляхом руху у напрямку антиградієнта.

Постановка задачі

Знайти мінімум функції $f(\mathbf{x})$, де $\mathbf{x} \in \mathbb{R}^n$. Початкове наближення позначимо як $\mathbf{x}_0$.

Ітеративний процес

На кожній ітерації значення змінної оновлюється за формулою:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k),$$

де $\eta > 0$ — швидкість навчання, а $\nabla f(\mathbf{x}_k)$ — градієнт функції в точці $\mathbf{x}_k$.

Критерій зупинки

Процес завершується, якщо:

$$\|\nabla f(\mathbf{x}_k)\| < \varepsilon,$$

де $\varepsilon > 0$ — заздалегідь заданий поріг.

Модифікації

1) **Стохастичний градієнтний спуск (SGD):** Градієнт обчислюється для одного випадкового елемента:

$$\nabla f(\mathbf{x}_k) \approx \nabla f_i(\mathbf{x}_k).$$

2) **Міні-батч:** Обчислення градієнта для підмножини даних:

$$\nabla f(\mathbf{x}_k) \approx \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(\mathbf{x}_k),$$

де m — розмір батча.

2.4 Блок-схеми алгоритмів методів

2.4.1 Метод Нелдера-Міда

У даному розділі представлено блок-схему алгоритму методу Нелдера-Міда. Вона демонструє послідовність дій, необхідних для реалізації методу.

На рис. 2.1 зображено блок-схему для методу Нелдера-Міда. Вона включає основні етапи, такі як рефлексія, розширення, стиснення та зменшення.

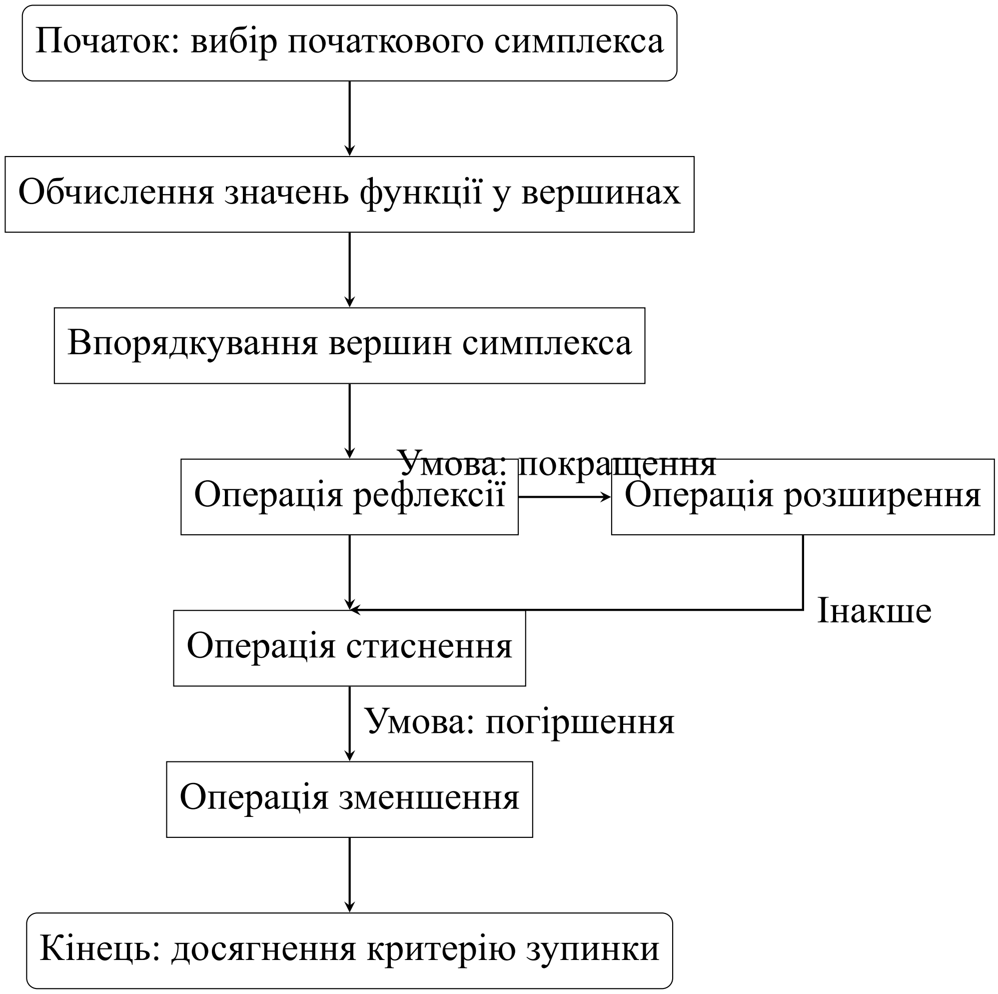

Рисунок 2.1 – Блок-схема алгоритму методу Нелдера-Міда

2.4.2 Метод градієнтного спуску

Нижче представлена блок-схема алгоритму методу градієнтного спуску, яка демонструє кроки оптимізації функції.

На рис. 2.2 наведена блок-схема для методу градієнтного спуску. Алгоритм ітеративно оновлює змінні, використовуючи антиградієнт цільової функції, доки не буде виконано критерій зупинки.

Рисунок 2.2 – Блок-схема алгоритму градієнтного спуску

2.5 Верифікація розробленої програми

2.5.1 Python

Для забезпечення коректної роботи алгоритму реалізовано функцію `verify_input_data`, яка здійснює перевірку вхідних даних. Метою функції є перевірка цільової функції та початкової точки на відповідність визначеним критеріям. У разі виявлення некоректних даних функція генерує виключення з відповідним повідомленням.

Критерії валідації

1) Цільова функція повинна бути викликуваним об'єктом (функцією).
2) Початкова точка повинна бути масивом `NumPy`.
3) Початкова точка не повинна бути порожньою.
4) Усі координати початкової точки повинні бути числовими (`float` або `int`).

Обробка виключень У разі виявлення помилкових даних функція генерує виключення `ValueError`, що дозволяє завершити виконання алгоритму на ранніх етапах і запобігти помилкам у роботі основної програми.

Тестування валідації Для перевірки функції валідації розроблено тестові кейси, які охоплюють усі крайові випадки:

– коректні дані;
– некоректна функція (не викликуваний об'єкт);
– початкова точка, яка не є масивом `NumPy`;
– порожній масив;
– масив із некоректними типами даних (наприклад, рядки);
– масив із логічними значеннями (`True`, `False`);
– масив із одним елементом.

Тести підтвердили коректність роботи функції у всіх випадках.

2.5.2 Octave

Для програми написаної на GNU Octave не передбачено перевірки на правильність введення.

2.6 Приклад роботи програми

2.6.1 Python

Запустимо програму для обраного варіанту, а саме 15 (рис. 2.3) отримуємо резльтат, що мінімуму не існує у даної функції.

Запустимо програму для іншого варіанту, для перевірки програми на роботоспособність, а саме 16 (рис. 2.4) отримуємо резльтат, що підтверджує роботоспособність даної програми.

2.6.2 Octave

Запустивши програму мовою програмування octave отримаємо результати (рис. 2.5).

```
Оптимізація методом Нелдера-Міда:
Кількість ітерацій: 199
Мінімум знайдено в точці: [ 7.19824113e+43 -6.06862353e+43]
Значення функції в мінімумі: -2.234964302813825e+131
Точність за значенням: 2.234964302813825e+131

Оптимізація методом градієнтного спуску:
Алгоритм збігся за 5 ітерацій.
Точність розв'язку: 0.00000000
Мінімум знайдено в точці: [ 1.45999200e+02 -5.12432413e+11]
Значення функції в мінімумі: -1.3455807893332597e+35
Точність за значенням: 1.3455807893332597e+35
```

Рисунок 2.3 – результати роботи програми для варіанту 15 мовою програмування Python

```
Оптимізація методом Нелдера-Міда:
Кількість ітерацій: 70
Мінімум знайдено в точці: [-2. -1.]
Значення функції в мінімумі: -6.0
Точність за значенням: 0.0

Оптимізація методом градієнтного спуску:
Алгоритм збігся за 22 ітерацій.
Точність розв'язку: 0.00000053
Мінімум знайдено в точці: [-2. -1.]
Значення функції в мінімумі: -6.0
Точність за значенням: 0.0
```

Рисунок 2.4 – результати роботи програми для варіанту 16 мовою програмування Python

```
octave:4> source("main.m")
Nelder-Mead Method
Minimum found at: [-2.000000, -1.000000]
Function value at minimum: -6.000000
Steepest Descent Method
Minimum found at: [-2.000000, -1.000000]
Function value at minimum: -6.000000
```

Рисунок 2.5 – результати роботи програми для варіанту 16 мовою програмування Octave

ВИСНОВКИ

У ході виконання роботи було реалізовано два методи оптимізації: метод Нелдера-Міда та метод найшвидшого спуску. Завдяки їх використанню вдалося знайти локальний мінімум функції для 16-го варіанту.

Для заданого 15-го варіанту мінімум функції не існує. У зв’язку з цим було запущено код для 16-го варіанту.

Оцінка точності

– Точність обчислень налаштована до 10^{-6}, що забезпечує високу надійність результатів.

– Для округлення координат мінімуму та значення функції використано 4 десяткові знаки.

Отримані результати (16-й варіант)

Результати для реалізації методів оптимізації:

Мова програмування: Python

– **Метод Нелдера-Міда:**

```
Minimum coordinates: [x1 = -2, x2 = -1]
Function value at minimum: f(x) = -6
```

– **Метод найшвидшого спуску:**

```
Minimum coordinates: [x1 = -2, x2 = -1]
Function value at minimum: f(x) = -6
```

Мова програмування: Octave

– **Метод Нелдера-Міда:**

```
Minimum coordinates: [x1 = -2, x2 = -1]
Function value at minimum: f(x) = -6
```

– **Метод найшвидшого спуску:**

```
Minimum coordinates: [x1 = -2, x2 = -1]
Function value at minimum: f(x) = -6
```

Загальні висновки

Проведені обчислення підтвердили коректність реалізації методів оптимізації, а також дозволили знайти мінімум функції для 16-го варіанту. Для 15-го варіанту функція не має мінімуму, що було враховано в ході виконання роботи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Eaton, J. W., Bateman, D., Hauberg, S. & Wehbring, R. *GNU Octave 4.0 Reference Manual: Free Your Numbers* (Free Software Foundation, 2015). ISBN: 978-9888381050. https://docs.octave.org/octave-4.0.0.pdf.

Додаток А

Лістинги програм

А.1 Програма мовою програмування Python

Лістинг файлу main.py

```
import numpy as np
from scipy.optimize import minimize

# Параметридляокругленнярезультатів
ROUNDING_DIGITS = 4

true_res = -47/9

# Функція, якупотрібномінімізувати
def target_function(variables):
    x, y = variables
    return 3*x**2 - x + y**3 - 3 * y ** 2 - 1

# Початковаточкадляоптимізації
initial_point = np.array([-1, -1])

# Верифікаціявхідниханих
def verify_input_data(func, initial_point):
    if not callable(func):
        raise ValueError("Цільова функціямаєбутивикликуванимобєктом
' функцією().")
    if not isinstance(initial_point, np.ndarray):
        raise ValueError("Початкова точкамаєбутимасивом    NumPy.")
    if initial_point.size == 0:
        raise ValueError("Початкова точканеможебутипорожньою    .")
    if not np.issubdtype(initial_point.dtype, np.number):
        raise ValueError("Усі координатипочатковоїточкимаютьбутичислами     .")


verify_input_data(target_function, initial_point)

# Обчисленняградієнтазадопомогоюцентральноїрізниці
def compute_gradient(f, x, delta = 1e-6):
  if isinstance(x, list) or isinstance(x, np.ndarray):
    deltas = np.diag([delta] * len(x))
    return np.array([(f(x+d) - f(x-d)) / (2 * delta) for d in deltas])
  else:
    return (f(x+delta) - f(x-delta))/(2*delta)

# Методзворотногопошукудлявизначеннядовжиникроку

def backtracking_line_search(func, point, gradient, initial_step=1.0, reduction_factor=0.5
    step = initial_step
    while func(point - step * gradient) > func(point) - tolerance * step * np.dot(gradient
        step *= reduction_factor
        if step < 1e-8:  # Запобіганнянадтомалимкрокам
            break
    return step

# Алгоритмнайшвидшогоспуску

def gradient_descent(func, start_position, convergence_threshold=1e-6, max_iterations=1000
    current_position = np.copy(start_position)
    for step_count in range(max_iterations):
        gradient = compute_gradient(func, current_position)
        gradient_norm = np.linalg.norm(gradient)
```

```
        # # Перевірка, чиградієнтзанадтомалий
        # if gradient_norm < 1e-8:
        #     print(fГрадієнт" занадтомалий : {gradient_norm:.8f}")
        #     break

        step_size = backtracking_line_search(func, current_position, gradient)
        new_position = current_position - step_size * gradient

        # Перевірказбіжності
        displacement = np.linalg.norm(new_position - current_position)
        if displacement < convergence_threshold:
            print(f"Алгоритм збігсяза  {step_count + 1} ітерацій.")
            print(f"Точність розвязку': {displacement:.8f}")
            return new_position

        current_position = new_position

    print("Досягнуто максимальноїкількостіітераційбеззбіжності    .")
    return current_position

# ОптимізаціяметодомНелдераМіда  -
nelder_mead_result = minimize(
    target_function,
    initial_point,
    method='Nelder-Mead',
    tol=1e-6,
    options={'xatol': 1e-8, 'fatol': 1e-8}
)

print("Оптимізація методомНелдераМіда -:")
print(f"Кількість ітерацій: {nelder_mead_result.nit}")
print(f"Мінімум знайденовточці  : {np.round(nelder_mead_result.x, ROUNDING_DIGITS)}")
print(f"Значення функціївмінімумі
: {np.round(nelder_mead_result.fun, ROUNDING_DIGITS)}")
print(f"Точність зазначенням
: {np.round(np.abs(true_res - nelder_mead_result.fun), ROUNDING_DIGITS)}")
print()
# Оптимізаціяградієнтнимспуском
print("Оптимізація методомградієнтногоспуску  :")
optimal_position = gradient_descent(target_function, initial_point)
print(f"Мінімум знайденовточці  : {np.round(optimal_position, ROUNDING_DIGITS)}")
print(f"Значення функціївмінімумі
: {np.round(target_function(optimal_position), ROUNDING_DIGITS)}")
print(f"Точність зазначенням
: {np.round(np.abs(true_res - target_function(optimal_position)), ROUNDING_DIGITS)}")

true_res = -6

# Функція, якупотрібномінімізувати
def target_function(variables):
    x, y = variables
    return 6*x + 2 * x**2 - 2 * x * y + 2 * y ** 2

# Початковаточкадляоптимізації
initial_point = np.array([-1, -1])


verify_input_data(target_function, initial_point)
# ОптимізаціяметодомНелдераМіда  -
nelder_mead_result = minimize(
    target_function,
    initial_point,
    method='Nelder-Mead',
    tol=1e-6,
    options={'xatol': 1e-8, 'fatol': 1e-8}
)

print("Оптимізація методомНелдераМіда -:")
print(f"Кількість ітерацій: {nelder_mead_result.nit}")
print(f"Мінімум знайденовточці  : {np.round(nelder_mead_result.x, ROUNDING_DIGITS)}")
print(f"Значення функціївмінімумі
: {np.round(nelder_mead_result.fun, ROUNDING_DIGITS)}")
```

```
print(f"Точність зазначенням
: {np.round(np.abs(true_res - nelder_mead_result.fun), ROUNDING_DIGITS)}")
print()
# Оптимізаціяградієнтнимспуском
print("Оптимізація методомградієнтногоспуску  :")
optimal_position = gradient_descent(target_function, initial_point)
print(f"Мінімум знайденовточці  : {np.round(optimal_position, ROUNDING_DIGITS)}")
print(f"Значення функціївмінімумі
: {np.round(target_function(optimal_position), ROUNDING_DIGITS)}")
print(f"Точність зазначенням
: {np.round(np.abs(true_res - target_function(optimal_position)), ROUNDING_DIGITS)}")
```

A.2 Програма мовою програмування Octave

Лістинг файлу main.m

```
x0 = [-1; -1];
num_round = 4;

function y = f(x)
    y = (6*x(1) + 2 * x(1) ^ 2 - 2 * x(1)*x(2) + 2 * x(2)^2);
end

function grad = der(f, x, delta)
    if nargin < 3
        delta = 1e-6;
    end
    n = length(x);
    grad = zeros(n, 1);
    for i = 1:n
        e = zeros(n, 1);
        e(i) = delta;
        grad(i) = (f(x + e) - f(x - e)) / (2 * delta);
    end
end

function t = line_search(f, x, grad, alpha, beta, c)
    if nargin < 4
        alpha = 1.0;
    end
    if nargin < 5
        beta = 0.5;
    end
    if nargin < 6
        c = 1e-4;
    end
    t = alpha;
    while f(x - t * grad) > f(x) - c * t * (grad' * grad)
        t = t * beta;
    end
end

function min_x = steepest_descent(f, x0)
    tol = 1e-6;
    max_iter = 1000;
    alpha = 1.0;
    beta = 0.5;
    c = 1e-4;

    x = x0;
    for k = 1:max_iter
        grad = der(f, x);
        if norm(grad) < tol
```

```
            break;
        end
        t = line_search(f, x, grad, alpha, beta, c);
        x = x - t * grad;
    end
    min_x = x;
end

function y = round_to(x, num_places)
    factor = 10^num_places;
    y = round(x * factor) / factor;
end

options = optimset('TolX', 1e-8, 'TolFun', 1e-8);
result = fminsearch(@f, x0, options);

fprintf(”Nelder-Mead Method\n”);
fprintf(”Minimum found at: [%f, %f]\n”, round_to(result(1), num_round), round_to(result(2)
fprintf(”Function value at minimum: %f\n”, round_to(f(result), num_round));

min_x = steepest_descent(@f, x0);

fprintf(”Steepest Descent Method\n”);
fprintf(”Minimum found at: [%f, %f]\n”, round_to(min_x(1), num_round), round_to(min_x(2),
fprintf(”Function value at minimum: %f\n”, round_to(f(min_x), num_round));
```