В.М. Савченко, О.Б. Маций, О.В. Мнушка

СИСТЕМНИЙ АНАЛІЗ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ В GNU OCTAVE

Міністерство освіти і науки України

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ АВТОМОБІЛЬНО-ДОРОЖНІЙ УНІВЕРСИТЕТ

В. М. Савченко
О.Б. Маций
О.В. Мнушка

СИСТЕМНИЙ АНАЛІЗ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ В *GNU OCTAVE*

Навчальний посібник

Харків
ХНАДУ
2020

УДК 303.732.4
С 12

Рекомендовано до видання рішенням Вченої ради
Харківського національного автомобільно-дорожнього університету,
протокол № 21/19/7.4 від 04 жовтня 2019 р.

Рецензенти: ***Ніконов О.Я.*** – Лауреат Премії Президента України, доктор технічних наук, професор, завідувач кафедри комп'ютерних технологій і мехатроніки Харківського національного автомобільно-дорожнього університету.
Бетін О.В. – Лауреат Державної Премії України в галузі науки та техніки, доктор технічних наук, професор, завідувач кафедри хімії, екології та експертизних технологій Національного аерокосмічного університету ім. Н. Є. Жуковського «ХАІ»;
Нємченко К.Е. – доктор фіз.-мат. наук, професор, завідувач кафедри інформаційних технологій в фізико-енергетичних системах Харківського національного університету імені В. Н. Каразіна;
Назаров О.С. – кандидат технічних наук, доцент, доцент кафедри програмної інженерії Харківського національного університету радіоелектроніки.

Колектив авторів:
Савченко В. М., к.т.н.,
Маций О. Б., к.т.н.,
Мнушка О. В., асистент

Савченко В. М.
Системний аналіз та математичне моделювання у ***GNU Octave***: навчальний
С 12 посібник / В. М. Савченко, О. Б. Маций, О. В. Мнушка. – Харків: ХНАДУ, 2020. – 128 с.
ISBN 978-966-303-752-3

Розглянуто методологічні питання системного аналізу, розглядаються поняття і визначення складних систем і системного аналізу. Описано етапи і процедури проведення системних досліджень, сформульовані цілі та завдання системного аналізу. Розглянуто математичні моделі і методи системного аналізу. Розглянуто типові постановки задач. Приділена увага питанням побудови моделей складних систем. Проведено класифікацію моделей. Приділено увагу дослідженню практичного застосування теорії системного аналізу. Посібник може бути корисним не тільки для студентів, а й викладачів, науковців та аспірантів вищих навчальних закладів.

УДК 303.732.4

ISBN 978-966-303-752-3

ВСТУП

Системний аналіз та математичне моделювання є актуальними та важливими способами отримання нових знань, технологічних та конструкторських рішень. Імітаційне та математичне моделювання суттєво зменшують не тільки час впровадження нових наукових розробок у виробництво, а також їх вартість, що підвищує конкурентоздатність.

Сучасний фахівець в галузі комп'ютерних наук та програмної інженерії має володіти апаратом аналітичного та чисельного аналізу, мати навички програмної реалізації основних методів та алгоритмів, серед яких оптимізація функцій однієї та багатьох змінних, обробка експериментальних даних, апроксимація функціональних залежностей тощо. Це важливо з декількох причин:

- *по-перше*, програміст під час роботи над різними проектами має справу із задачами, де потрібні не тільки вміння «писати код», а й широка ерудиція;
- *по-друге*, розробка архітектури додатків базується на глибокому розумінні предметної галузі;
- *по-третє*, конкурентноспроможність програміста залежить в тому числі й від вміння реалізовувати стандартні чисельні методи та стандартні алгоритми на типових структурах даних, таких як графи та дерева.

Ці знання та навички допомагають інженерам-конструкторам аналізувати отримані за допомогою пакетів прикладних програм *CAD/CAM/CAE* результати моделювання процесів та пристроїв, приймати правильні конструкторські та технологічні рішення, обирати оптимальні алгоритми розв'язання різних класів задач.

Посібник містить необхідні теоретичні відомості про методи, що вивчаються, та алгоритми побудови обчислювального процесу за обраним методом [1–9]. Для програмної реалізації означених методів обрано мову програмування *GNU Octave* (*MATLAB*) [10-12] та інтегроване середовище (*IDE*) *Octave*. Стандартна бібліотека *Octave* надає перевірені реалізації поширених числових методів та засоби візуалізації результатів, а *IDE Octave*, починаючи з версії 4.0, має достатньо зручний графічний віконний інтерфейс однаковий для різних операційних систем.

GNU Octave є вільним програмним забезпеченням, тому всі результати розрахунків або імітаційного моделювання, що отримані з його допомогою, можна використовувати в інженерній практиці та науковій діяльності. Існують версії пакета для основних операційних систем – *Windows*, *Linux*, *Mac OS* та *BSD Unix*.

Посібник містить багато прикладів розв'язання задач та їх окремих фрагментів[1], що повинно допомогти студентам в процесі розробки власних обчислювальних алгоритмів. Для всіх тем наведено орієнтовний перелік стандартних функцій та операторів *Octave,* що надає необхідні інструменти для безпосереднього розв'язання задач та перевірки результатів, отриманих за допомогою розроблених програм, написаних мовою програмування *Octave* (*MATLAB*).

Бажано використовувати актуальні версії пакету *GNU Octave*. Інформація, що необхідна для початку роботи із пакетом, зосереджена у додатках А-В. У додатку Г наведені основні операції над матрицям у вигляді прикладів, які рекомендується виконати, якщо ви не знайомі із мовою програмування та структурами даних *GNU Octave*.

Матеріал посібника розділений на вісім тем, в кінці кожної теми є практичне завдання для закріплення вивченого матеріалу. Рекомендується послідовне вивчення тем. Приклади, що мітяться у кожній темі є ілюстративним матеріалом та не призначені для розв'язання будь яких реальних наукових або інженерних проблем із реального життя.

[1] Зовнішній вигляд результатів обчислень в прикладах подекуди змінено з міркувань економії місця

ТЕМА 1
РОЗВ'ЯЗАННЯ СИСТЕМ ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ

Метою є ознайомлення студентів з методами розв'язання систем лінійних алгебраїчних рівнянь (СЛАР) аналітичними та числовими методами із використанням *Octave*.

Постановка задачі. Задано систему з N лінійних алгебраїчних рівнянь (1.1)

$$\begin{cases} a_{11}x_1 & a_{12}x_2 & \dots & a_{1N}x_N & = & b_1, \\ a_{21}x_1 & a_{22}x_2 & \dots & a_{2N}x_N & = & b_1, \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{N1}x_1 & a_{N2}x_2 & \dots & a_{NN}x_N & = & b_N. \end{cases} \quad (1.1)$$

Знайти розв'язок цієї системи у вигляді (1.2)

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}, \quad (1.1)$$

де $\boldsymbol{A}$ – матриця коефіцієнтів розміром $N \times N$, $\boldsymbol{b}$ – вектор правих частин розміром N, $\boldsymbol{x}$ – шуканий вектор розміром N.

Розв'язання систем лінійних алгебраїчних рівнянь є однією з основних обчислювальних задач в різних прикладних областях – чисельні методи, методи наближення даних, спектральні задачі т. п. призводять до необхідності пошуку розв'язку систем лінійних алгебраїчних рівнянь. Частина відомих методів (Крамера, Гаусса) розв'язання СЛАР є зручними для «ручного» застосування, інша (ітераційні методи, розкладання матриць) – для реалізації у вигляді комп'ютерної програми [3–6]. За допомогою *Octave* СЛАР розв'язують як у напівавтоматичному режимі, так і використовуючи велику кількість вбудованих функцій [7, 8].

Метод Гауса складається з прямого та зворотного проходів. На етапі прямого проходу за допомогою елементарних перетворень послідовно виключають змінні з (1.1) та формують верхню трикутну матрицю (1.2), а під час зворотного – обчислюють корені $x_i, i \in [1, N]$, починаючи з x_N. Методу Гаусса властиво накопичувати похибки при використанні десяткових дробів на проміжних кроках методу. Реалі-

зація методу для роботи із простими дробами є нетривіальною задачею[2], оскільки зазвичай мови програмування не підтримують такий формат даних на внутрішньому рівні[3].

$$\begin{cases} c_{11}x_1 & c_{12}x_2 & \dots & c_{1N-1}x_{N-1} & c_{1N}x_N & = & d_1, \\ 0 & c_{22}x_2 & \dots & c_{2N-1}x_{N-1} & c_{2N}x_N & = & d_2, \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & c_{NN}x_N & = & d_N. \end{cases} \quad (1.3)$$

Наведений алгоритм реалізують на одній з мов програмування та застосовують у прикладних задачах. Основним недоліком є верифікація такої програми на різних наборах вхідних даних, тому даний підхід є гарним для навчання, але не забезпечує універсальним інструментом, що можна застосовувати для різних прикладних задач.

Перед початком роботи із пакетом опрацюйте приклади із додатка Г, це дасть розуміння роботи програм, що використовуються у якості прикладів.

З огляду на вище сказане, доцільним є використовувати прикладних бібліотек (*BLAS*[4], *LAPACK*[5], *Eigen*[6]) або пакетів спеціалізованих прикладних програм, наприклад, *Octave*, яка надає наступні можливості при роботі зі СЛАР:

- розв'язання систем з квадратними і прямокутними матрицями;
- розв'язання систем прямими і ітераційними методами;
- матричні розкладання;
- зберігання великих розріджених матриць в компактній формі і спеціальні алгоритми для розв'язання систем з такими матрицями.

Оператори та функції для роботи із матрицями

Група операторів «.+» ; «.-»; «.*» ; «./» ; «.^» використовується для виконання поелементних операцій над матрицями. У звичайній математиці додавання і віднімання матриць визначається поелементна операція. Оскільки використання операторів *Octave* без крапки

---

[2] Існують прикладні бібліотеки, наприклад, *GNU MPIR*, *gmpy* та ін.

[3] *C++*1x, *Python* наразі забезпечують підтримку на рівні стандартної бібліотеки

[4] http://www.netlib.org/blas/

[5] http://www.netlib.org/lapack/

[6] http://eigen.tuxfamily.org/index.php?title=Main_Page

означає «регулярне» використання, різниці між «+» і «.+» немає. Що стосується множення, ділення та піднесення до степеня, існує різниця між "регулярним" використанням та поелементним; отже, не використовуйте цих операторів без крапки перед ними.

Таблиця 1.1

Матричні оператори

Octave	Лінійна алгебра	Опис
C=A.'	$C = A^T$	Транспонування
C=A'	$C = A^*$	Ермітова матриця
C=A.^B	$C = A_{ij}{}^{B_{ij}}$	Піднесення до степеня
*C=A*B*	$C = AB$	Перемноження матричне
*C=A.*B*	$C_{ij} = A_{ij} * B_{ij}$	Перемноження поелементне
X=A\B	$AX = B \Leftrightarrow X = A^{-1}B$	Ліве матричне ділення
X=A/B	$XB = A \Leftrightarrow AB^{-1}$	Праве матричне ділення
X=A./B	$C_{ij} = A_{ij} * B_{ij}$	Ділення поелементне
C=A+B	$C = A + B$	Додавання матриць
C=A-B	$C = A - B$	Віднімання матриць

Функції *Octave* для роботи із матрицями:

- $[\mathrm{R}, \mathrm{p}, \mathrm{Q}] = chol(A)$ – розклад Холецького; $R^T R = A$; $R^T R = Q^T A\, Q$; $p = 0$, якщо матрицю визначено як додатну.
- $[L, U, P] = lu(A)$ – LU-розклад матриці на верхню (L) та нижню (U) трикутні та матрицю перестановок (P) такий, що $PA = LU$, отже $A = P^T LU$
- $[Q, R, P] = qr(A)$ – QR-розклад матриці на унітарну (Q), верхню трикутну (R) та матрицю перестановок (P) такий, що $AP = QR$, отже $A = QRP^T$.
- $inv(A)$ – обернена матриця до A: $AA^{-1} = A^{-1}A = E$, де E – одинична матриця.
- $det(A)$ – детермінант матриці.
- $norm(A, p)$ – норма матриці. Опція p={0,1,2,'*inf*','*fro*'} є відповідною метрикою. Без аргументів повертає евклідову норму $\|b\|_2 = \sqrt{\sum_i |b_i|^2}$, а у формі $norm(A, 'fro')$ повертає норму Фробеніуса $\|A\|_2 = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |a_{i,j}|^2}$;

• $[R, lambda] = eig(A)$ обчислює діагональну матрицю власних значень $lambda$ та матрицю правих власних векторів R, що задовольняють рівності $AR = Rlambda$. Ці вектори є нормованими таким чином, що норма кожного з них дорівнює одиниці. Для пошуку лівих власних векторів – $[L, lambda] = eig(A')$.

• $sum(A)$, $mean(A)$ – сума та середнє арифметичне елементів.

• $corr(x, y)$ – обчислення коефіцієнта кореляції двох векторів.

Функції *Octave* для розв'язання СЛАР

Розв'язання систем за допомогою знаку зворотної косої риски «\» – універсальний метод, який без додаткових налаштувань добре підходить для більшості простих СЛАР. Знайдемо розв'язок наступної системи рівнянь

$$\begin{cases} 2x - 3y + z = 2 \\ 2x + y - 4z = 9 \\ 6x - 5y + 27 = 17 \end{cases}. \qquad (1.4)$$

Розв'язком (1.4) є $x = 5,\ y = 3,\ z = 1$, в чому не важко переконатися знайшовши розв'язок вручну. Для спрощення перепишемо систему рівнянь у матричному вигляді $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$ та знайдемо її розв'язок за допомогою *Octave*.

Для цього потрібно заповнити матрицю коефіцієнтів при x і вектор-стовпець вільних членів у правій частині (права частина повинна бути саме стовпцем, інакше виведеться помилка про невідповідність розмірностей аргументів) і використати знак «\» (або функцію $mldivide\ (A, b)$). Для запобігання виведенню на екран значень функцій та змінних в кінці рядка слід поставити «;». Наприкінці потрібно обчислити нев'язку, яка повинна наближатися до нуля і переконатися, що розв'язок системи знайдено вірно.

Від вибору способу розв'язання системи рівнянь залежить обчислювальна складність алгоритму та його стійкість до наборів вхідних даних. При розв'язанні у *Octave* системи лінійних алгебраїчних рівнянь за допомогою знаку зворотної косої риски $A \setminus b$ працює алгоритм, який в залежності від типу матриці розв'язує систему за допомогою найбільш відповідного методу, реалізованого в одній з процедур пакету *LAPACK* або *UMFPACK*.

Приклад 1.1. Пошук розв'язку СЛАР.

Задача – розв'язати СЛАР за допомогою «\».

```
## Приклад 1.1 Розв'язок СЛАР "\"
display("Матриця коефіцієнтів")
A = [2 -3 1; 2 1 -4; 6 -5 2]
display("Стовпчик вільних членів")
b = [2; 9; 17]
display("Розв'язок x = A \ b")
x = A \ b
display("Перевірка norm(A*x-b)")
norm(A*x - b)
display("Перевірка")
b - A * x
```

Результат:

```
>> Матриця коефіцієнтів
A =
   2  -3   1
   2   1  -4
   6  -5   2
Стовпчик вільних членів
b =
    2
    9
   17
Розв'язок x = A  b
x =
   5.00000
   3.00000
   1.00000
Перевірка norm(A*x-b)
ans =    1.8310e-15
Перевірка
ans =
   4.4409e-16
  -1.7764e-15
   0.0000e+00
```

Приклад 1.2. Пошук розв'язку СЛАР.

Задача – найти розв'язок СЛАР у символьному вигляді.

```
## Приклад 1.2
pkg load symbolic
## Символьні змінні
syms x y z
display([x y z])
## Система рівнянь
display("Система рівнянь")
eqn1 = 2*x +   y +   z ==   2
eqn2 =  -x +   y -   z ==   3
eqn3 =   x + 2*y + 3*z == -10
## Розв'язок
display("Розв'язок")
[x y z] = solve(eqn1, eqn2, eqn3)
display("Перевірка, всі рядки мають бути True")
display("subs(eqn1)")
subs(eqn1)
display("subs(eqn2)")
subs(eqn2)
display("subs(eqn3)")
subs(eqn3)
```

Результат:

```
(sym) [x  y  z]  (1x3 matrix)
Система рівнянь
eqn1 = (sym) 2*x + y + z = 2
eqn2 = (sym) -x + y - z = 3
eqn3 = (sym) x + 2*y + 3*z = -10
Розв'язок
x = (sym) 3
y = (sym) 1
z = (sym) -5
Перевірка, всі рядки мають бути True
subs(eqn1)
ans = (sym) True
subs(eqn2)
ans = (sym) True
subs(eqn3)
ans = (sym) True
```

Матричні декомпозиції Холецького, LU и QR.

Матричні декомпозиції – це методи, за допомогою яких матрицю розкладають на складові частини спеціального вигляду, що полегшує обчислення більш складних матричних операцій. Методи матричної декомпозиції (матричної факторизації) є основою лінійної алгебри в комп'ютерних обчисленнях, навіть для базових операцій, таких як розв'язання систем лінійних рівнянь, обчислення зворотного і обчислення визначника матриці.

LU-декомпозиція призначена для квадратних матриць і розбиває матрицю на компоненти L и U.

QR- розкладання призначене для будь-яких m x n матриць (необмежена квадратними матрицями) і розбиває матрицю на Q і R компоненти.

Розкладання Холецького призначене для квадратних симетричних матриць, де всі значення більше нуля, так званих позитивно визначених матриць що розкладається як на нижні (L), так і верхні (U) трикутні матриці. Для пошуку декомпозицій найчастіше використовують метод Гаусса. Розглянемо як приклад розкладання Холецького матриці, зазначимо, що функція *chol* () не перевіряє матрицю на симетричність!

Приклад 1.3. Матрична декомпозиція.

Задача – знайти матричну декомпозицію Холецького.

```
## Приклад 1.3
## Декомпозиція Холецького
clc, clear
display("Вихідна матриця")
A = [2 1 1;
     1 2 1;
     1 1 2]
display("Пошук правої трикутної матриці; P=0 якщо матриця
позитивна:")
[U, P] = chol(A, "upper")
display("Пошук лівої трикутної матриці:")
[L, P] = chol(A, "lower")
display("Перевірка: B = U' * U")
B = U' * U
display("Перевірка: C = L * L'")
C = L * L'
```

Результат:

```
Вихідна матриця
A =
   2   1   1
   1   2   1
   1   1   2
Пошук правої трикутної матриці; P=0 якщо матриця позитивна:
U =
   1.41421   0.70711   0.70711
   0.00000   1.22474   0.40825
   0.00000   0.00000   1.15470
P = 0
Пошук лівої трикутної матриці:
L =
   1.41421   0.00000   0.00000
   0.70711   1.22474   0.00000
   0.70711   0.40825   1.15470
P = 0
Перевірка: B = U' * U
B =
   2.0000   1.0000   1.0000
   1.0000   2.0000   1.0000
   1.0000   1.0000   2.0000
Перевірка: C = L * L'
C =
   2.0000   1.0000   1.0000
   1.0000   2.0000   1.0000
   1.0000   1.0000   2.0000
```

LU-декомпозиція матриць може бути застосована для прямокутних і квадратних матриць, при розкладанні рядки можуть мінятися місцями, про що можна дізнатися по додатковому параметру *P*. Зверніть увагу на матрицю перестановок (P), без її урахування результат перевірки буде з точністю до перестановки рядків (див. приклад 4, матриці *C* та *B*), що може призвести до невірних результатів при використанні матричних декомпозицій для розв'язку СЛАР.

QR-декомпозиція відрізняється гарною обчислювальною стійкістю і застосовується, зокрема, при наближенні даних методом найменших квадратів.

Приклад 1.4. Матрична декомпозиція.

Задача – написати програму *LU*-декомпозиції.

```
## Приклад 1.4
## LU-декомпозиція
display("Вихідна матриця")
A = [2 5 2; 5 2 1; 2 1 2]
display("Пошук LU-декопозиції, P=0 якщо матриця позитивна:")
[L, U, P] = lu(A)
display("Перевірка: B = L * U"); B = L * U
display("Перевірка: C = P' * L * U"); C = P' * L * U
```

Результат:

```
Вихідна матриця
A =
   2   5   2
   5   2   1
   2   1   2
Пошук LU-декопозиції, P=0 якщо матриця позитивна:
L =
   1.00000   0.00000   0.00000
   0.40000   1.00000   0.00000
   0.40000   0.04762   1.00000
U =
   5.00000   2.00000   1.00000
   0.00000   4.20000   1.60000
   0.00000   0.00000   1.52381
P =
Permutation Matrix
   0   1   0
   1   0   0
   0   0   1
Перевірка: B = L * U
B =
   5   2   1
   2   5   2
   2   1   2
Перевірка: C = P' * L * U
C =
   2   5   2
   5   2   1
   2   1   2
```

Приклад 1.5. Матрична декомпозиція.

Задача – написати програму *QR*-декомпозиції.

```
## Приклад 1.5
## QR-декомпозиція
A = [ 2 1 2;2 6 2; 5 2 1]
display("Пошук QR-декопозиції, P=0 якщо матриця позитивна:")
[Q, R, P] = qr(A)
display("Перевірка: B = Q * R");        B = Q * R
display("Перевірка: C = Q' * R * P'"); C = Q * R * P'
```

Результат:

```
A =
   2   1   2
   2   6   2
   5   2   1
Пошук QR-декопозиції, P=0 якщо матриця позитивна:
Q =
  -0.156174   0.324957  -0.932745
  -0.937043  -0.347368   0.035875
  -0.312348   0.879624   0.358748
R =
  -6.40312  -3.74817  -2.49878
   0.00000   4.35330   0.83480
   0.00000   0.00000  -1.43499
P = Permutation Matrix
   0   1   0
   1   0   0
   0   0   1
Перевірка: B = Q * R
B =
   1.0000   2.0000   2.0000
   6.0000   2.0000   2.0000
   2.0000   5.0000   1.0000
Перевірка: C = Q * R * P'
C =
   2.0000   1.0000   2.0000
   2.0000   6.0000   2.0000
   5.0000   2.0000   1.0000
```

Приклад 1.6. Застосування матричної декомпозицій до розв'язання систем лінійних алгебраїчних рівнянь[7]

```
## Приклад 1.6, розв'язок системи рівнянь
clc, clear;
## 1 Backslash
display("Вихідна матриця:")
A = [ 8 4 8 0; -2 -9 -9 6; 1 4 0 9; 8 9 6 5 ]
display("Стовпчик вільних членів")
b = [12; -88; 89; -11]
display("1. Розв'язок x = A \\ b");   x = A\b
display("Перевірка norm(A*x-b)");     norm(A*x - b)
display("Похибка:");                  err = (A*x - b)
## 2. QR
## Ax=b, AP=QR, A=QRP' => Ry=Q'b, x = Py, y=R\(Q'b)
display("2. QR декомпозиція")
[Q R P] = qr(A)
display("Розв'язок x = R\\(Q'*b)");   x = P*(R \(Q' * b ))
display("Перевірка norm(A*x-b)");     norm(A*x - b)
display("Похибка:");                  err = Q*R*P'*x-b
## 3. LU
## Ax=b,PA=LU,A=P'LU=>Ly=Pb,Ux=y=>LUx=Pb;y=L\(P*B),x=U\y
display("3. LU декомпозиція");        [L U P] = lu(A)
display("Розв'язок x=U\\(L\\(P*b))"); x = U\(L\(P * b))
display("Перевірка norm(A*x-b)");     norm(A*x - b)
display("Похибка");                   err = P'*L*U*x-b
```

Результат (в скороченому вигляді, всі вектори представлено у транспонованому вигляді):

```
A =
   8   4   8   0
  -2  -9  -9   6
   1   4   0   9
   8   9   6   5
b =  12  -88   89  -11
1. Розв'язок x = A \ b
ans =  -29.2690   -7.1912   34.3647   16.3371
Перевірка norm(A*x-b)
ans =    4.2633e-14
```

[7] https://www.gnu.org/software/gsl/doc/html/linalg.html#qr-decomposition-with-column-pivoting

```
Похибка:
ans = 0.0000e+00  -2.8422e-14  -1.4211e-14  -2.8422e-14
2. QR декомпозиція
Розв'язок x = R \ (Q' * b)
ans =   -29.2690   -7.1912   34.3647   16.3371
Перевірка norm(A*x-b)
ans =    3.1776e-14
Похибка:
ans = -4.2633e-14  -4.2633e-14   4.2633e-14   2.8422e-14
3. LU декомпозиція
Розв'язок x = U \ (L \ (P * b))
ans = -29.2690   -7.1912   34.3647   16.3371
Перевірка norm(A*x-b)
ans =    4.2633e-14
Похибка:
ans =    0.0000e+00  -2.8422e-14  -1.4211e-14  -2.8422e-14
```

Як бачимо, для тестового прикладу всі три способи дали однаковий результат із заданою точністю. Зазначимо, що без використання матриць перестановок для методів, основаних на матричній декомпозиції можна отримати неправильний результат.

Функція linsolve для пошуку розв'язку систем лінійних алгебраїчних рівнянь в найпростішому варіанті викликається з двома вхідними аргументами і одним вихідним аргументом $x = linsolve(A, b)$ та розв'язує СЛАР (1.1) одним із способів, в залежності від того, квадратна матриця, чи ні:

- Якщо $\boldsymbol{A}$ - квадратна матриця, то попередньо обчислюється її *LU*-декомпозицію і потім розв'язується дві системи рівнянь з трикутними матрицями $\boldsymbol{L}$ і $\boldsymbol{U}$;
- Якщо $\boldsymbol{A}$ - прямокутна матриця, то попередньо обчислюється її *QR*-розкладання і потім розв'язується система з трикутними матрицями.

Для функції *linsolve*() за допомогою структури *opts* вказують на тип матриці (допустимі значення *true* або *false*):

- *SYM* – симетрична;
- *LT* – нижня трикутна;
- *UT* – верхня трикутна;
- *UHESS* – матриця Гессенберга;
- *POSDEF* – симетрична, позитивно визначена;

- *RECT* – прямокутна;
- *TRANSA* – використання транспонованої матриці.

Якщо матриця системи позитивно визначена, то це обов'язково врахувати при пошуку розв’язку СЛАР, оскільки для позитивно визначених матриць розв’язання засноване на декомпозиції Холецького, вимагає менше операцій, ніж *LU*-декопозиція, що використовується при пошуку розв’язку СЛАР з квадратними матрицями загального вигляду.

У разі розріджених матриць при їх конструюванні слід застосовувати функцію *sparse* ( ), далі з ними можна працювати за допомогою звичайного розв'язувача («\») або використовувати спеціальні функції.

Приклад 1.7. Розв’язання системи рівнянь.

Задача – розробити програму розв’язання розряджених СЛАР

```
## Приклад 1.7
## Розв'язок рівнянь
clc, clear;
display("Вихідна матриця:")
A = sparse([0 2 0 1 0; 4 -1 -1 0 0; 0 0 0 3 -6; -2 0 0 0 2; 0
0 4 2 0])
display("Стовпчик вільних членів")
b = sparse([8; -1; -18; 8; 20])
display("Розв'язок x = A \\ b");  x = A \ b
display("Перевірка norm(A*x-b)"); norm(A*x - b)
display("Похибка:"); err = A*x - b
display("Перегляд матриць у розгорнутому вигляді.")
display("full(A)"); display(full(A))
display("full(b)"); display(full(b))
display("full(x)"); display(full(x))
display("full(err)"); display(full(err))
display("Цю систему можна розвязати аналогічно прикладу 1.6,
додатково: help lu та help qr")
```

Результат:

```
Вихідна матриця:
A =
Compressed Column Sparse (rows = 5, cols = 5, nnz = 11 [44%])
  (2, 1) ->  4
```

```
  (4, 1) -> -2
  (1, 2) ->  2
  (2, 2) -> -1
  (2, 3) -> -1
  (5, 3) ->  4
  (1, 4) ->  1
  (3, 4) ->  3
  (5, 4) ->  2
  (3, 5) -> -6
  (4, 5) ->  2
Стовпчик вільних членів
b =
Compressed Column Sparse (rows = 5, cols = 1, nnz = 5 [100%])
  (1, 1) ->  8
  (2, 1) -> -1
  (3, 1) -> -18
  (4, 1) ->  8
  (5, 1) ->  20
Розв'язок x = A \ b
x =
Compressed Column Sparse (rows = 5, cols = 1, nnz = 5 [100%])
  (1, 1) ->  1.00000
  (2, 1) ->  2.0000
  (3, 1) ->  3.0000
  (4, 1) ->  4.0000
  (5, 1) ->  5
Перевірка norm(A*x-b)
ans =    3.5527e-15
Похибка:
err =
Compressed Column Sparse (rows = 5, cols = 1, nnz = 1 [20%])
  (3, 1) ->    3.5527e-15
Перегляд матриць у розгорнутому вигляді.
full(A)
   0   2   0   1   0
   4  -1  -1   0   0
   0   0   0   3  -6
  -2   0   0   0   2
   0   0   4   2   0
full(b')
    8   -1  -18    8   20
full(x')
```

```
   1.00000   2.00000   3.00000   4.00000   5.00000
full(err')
   0.0000e+00  0.0000e+00  3.5527e-15  0.0000e+00   0.0000e+00
Цю систему можна розвязати аналогічно прикладу 1.6, додатково:
help lu та help qr
```

Резюме

При вивченні даної теми ознайомилися із деякими можливостями пакету *Octave* для пошуку розв'язку систем лінійних алгебраїчних рівнянь. Також ми познайомились із різними методами матричної декомпозиції та способами пошуку розв'язку СЛАР за їх допомогою.

Розібрані в цьому розділі підходи до розв'язання СЛАР застосовуються і в інших пакетах на основі *MATLAB* (*SciLab*, *FreeMAT*), в *Python* (*ScyPy*), *C++* (*Eigen*) й т. п.

Функції *Octave*, які використовуються в роботі: $chol()$, $lu()$, $qr()$, $norm()$, $eig()$, оператор «\», додатково див. функції *Octave* $hess()$, $qz()$, $schur()$, $svd()$ та інші.

Питання для самоперевірки

1. Назвіть приклади застосування СЛАР.
2. Які недоліки у методу Гаусса? Як їх подолати?
3. Назвіть основні прийоми роботи з матрицями у *Octave*.
4. Задати матриці A=[2,3; 4,1], b=[13; 17] та знайти розв'язок СЛАР числовим і графічним методами за допомогою *Octave*.
5. Задати матриці A=[2,3; 4,1],b=[10; 8] та знайти розв'язок СЛАР за методом Крамера та графічним методом за допомогою *Octave*.

Практичне завдання

Розв'язати СЛАР методом Гаусса, Розв'язати СЛАР за допомогою вбудованих функцій *Octave*.

Порядок виконання завдання

1. Згідно з варіантом привести СЛАР до вигляду $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$.
2. Розв'язати СЛАР за методом Гаусса.
3. Знайти власні значення матриць $\boldsymbol{A}$ та $\boldsymbol{b}$.
4. Знайти розв'язок у вигляді $\boldsymbol{x} = \boldsymbol{A}\backslash\boldsymbol{b}$ та відхил $\boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x}$.

5. Розв'язати СЛАР за допомогою розкладання матриць Холецького, *LU*- та *QR*-розкладань.

6. Задати матриці *A*=(*rand*(5).*25).+10 та *b*=(*rand*(1,5).*20)'. Повторити п. 2. Прокоментувати результати.

Індивідуальні завдання

Знайти розв'язок системи рівнянь згідно з варіантом.

1. $\begin{cases} x_1 - 2x_2 + 3x_3 = 3 \\ 2x_1 + x_2 + 7x_3 = 1 \\ x_1 + 4x_2 + 2x_3 = 2 \end{cases}$

2. $\begin{cases} 2x_1 - 3x_2 + x_3 = 10 \\ 3x_1 - x_2 - x_3 = 3 \\ x_1 + x_2 - 7x_3 = 8 \end{cases}$

3. $\begin{cases} 5x_1 - 3x_2 + 9x_3 = 3 \\ 6x_1 + 2x_2 - x_3 = 2 \\ 3x_1 - x_2 + x_3 = -1 \end{cases}$

4. $\begin{cases} 4x_1 - x_2 + x_3 = 2 \\ 2x_1 + 2x_2 - x_3 = 8 \\ 3x_1 + 5x_3 - x_3 = 7 \end{cases}$

5. $\begin{cases} 7x_1 + x_2 - 5x_3 = 6 \\ 4x_1 - 4x_2 - 5x_3 = -12 \\ x_1 - 2x_2 + x_3 = -4 \end{cases}$

6. $\begin{cases} 4x_1 - 2x_2 + 3x_3 = 2 \\ 2x_2 - x_2 + 2x_3 = 8 \\ 3x_1 + 5x_2 - x_3 = 7 \end{cases}$

7. $\begin{cases} 7x_1 - x_2 + 2x_3 = -1 \\ x_1 + 2x_2 - 3x_3 = 5 \\ 5x_1 - 4x_2 - 2x_3 = -4 \end{cases}$

8. $\begin{cases} 9x_1 - 3x_2 + 11x_3 = -2 \\ 5x_1 + 2x_2 - 7x_3 = 6 \\ 3x_1 - 4x_2 + x_3 = -8 \end{cases}$

9. $\begin{cases} 2x_1 - 2x_2 + x_3 = 2 \\ 2x_2 + x_2 + x_3 = 5 \\ 7x_1 - x_2 + 2x_3 = 6 \end{cases}$

10. $\begin{cases} 3x_1 - x_2 - x_3 = -6 \\ 5x_1 - 2x_2 + x_3 = -2 \\ 6x_1 - x_2 + 2x_3 = 5 \end{cases}$

11. $\begin{cases} 4x_1 - x_2 + 2x_3 = 7 \\ x_1 + 4x_2 + x_3 = 6 \\ 2x_1 - x_2 + x_3 = 3 \end{cases}$

12. $\begin{cases} 4x_1 - x_2 + x_3 = -2 \\ 5x_1 + 2x_2 - x_3 = 6 \\ x_1 - 4x_2 + x_3 = -8 \end{cases}$

13. $\begin{cases} x_1 - 4x_2 + x_3 = 2 \\ 5x_1 + x_2 - 4x_3 = 5 \\ 2x_1 - x_2 + 2x_3 = 8 \end{cases}$

14. $\begin{cases} 7x_1 - x_2 + x_3 = 1 \\ 5x_1 + 2x_2 - x_3 = -2 \\ 3x_1 - 7x_2 + 2x_3 = 3 \end{cases}$

15. $\begin{cases} x_1 + 3x_2 + 7x_3 = 8 \\ 2x_1 - x_2 + 3x_3 = 10 \\ x_1 - 3x_2 + x_3 = 6 \end{cases}$

16. $\begin{cases} 6x_1 - x_2 + x_3 = 2 \\ 2x_1 - 4x_2 + x_3 = -3 \\ x_1 + x_2 - 2x_3 = 1 \end{cases}$

ТЕМА 2
ЛІНІЙНИЙ РЕГРЕСІЙНИЙ АНАЛІЗ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ

Метою є ознайомлення студентів з лінійним методом найменших квадратів (МНК) та аналіз експериментальних даних засобами *Octave* та її графічною підсистемою для візуалізації експериментальних даних.

Постановка задачі. В результаті експериментальних досліджень отримано набір точок (табл. 2.1), які пов'язані деякою функціональної залежністю $y = f(x)$.

Побудувати аналітичну залежність $y = f(x, a_1, a_2, \dots, a_n)$, яка найбільш точно описує результати експерименту.

Таблиця 2.1

Набір вхідних даних

x	x_1	x_2	x_3	…	x_n
y	y_1	y_2	y_3	…	y_n

Основи побудови двовимірних графіків у *Octave*

При роботі із експериментальними даними важливе місце під час їх аналізу займає візуалізація за допомогою якої можна визначити приблизний закон розподілу величин, а також приблизно оцінити адекватність отриманих даних. Розглянемо базові можливості пакету при роботі із двовимірними графіками, способи побудови тривимірних графіків ми розглянемо у наступних розділах, коли в них настане потреба.

- $plot(x_1, y_1, \dots)$ – побудова графіку функцій, x_i, y_i – одновимірні матриці, y – може бути функцією користувача
- $ezplot(f(x), [x_{min}\ x_{max}])$ – побудова графіка функції, яку задано символьним виразом, якщо не вказані $[x_{min} \dots x_{max}]$, то графік будується на інтервалі $[-2\pi \dots 2\pi]$. У якості заголовку графіку виводиться функція.
- $fplot(f(x), [x_{min}\ x_{max}])$ – побудова графіка функції, яку задано символьним виразом або m-файлом, якщо не вказані $[x_{min} \dots x_{max}]$, то графік будується на інтервалі $[-5 \dots 5]$.

Цей перелік функцій не є вичерпним. Для надання гарного

вигляду графіки потребують додаткових налаштувань:

• *стиль лінії.* Є чотири стилі ліній – суцільна ('-'), штрихова ('--'), пунктирна (': ') та штрих-пунктирна ('-.'). Тип лінії позначається відповідним символом та записується поряд з рядом даних, наприклад $plot(sin(x), x, '-.')$. Додатковий параметр *'linewidth'* задає товщину лінії;

• *символ точки*: '+' (плюс), '*o*' (коло), '*' (зірка), '.' (точка),'*x*' (хрест), '*s*'(квадрат), '*d*' (ромб) ,'^','*v*', '>','<' трикутники, '*p*' п'ятикутник, '*h*' шестикутник. Додатковий параметр *'markersize'* задає розмір символу;

• *колір лінії (або символу)*:'*k*' (*blacK*, чорний), '*r*' (***R**ed*, червоний), '*g*' (***G**reen*, зелений), '*b*' (***B**lue*, блакитний), '*y*' (***Y**ellow*, жовтий), '*c*' (***C**yan*, пурпурний), '*w*' (***W**hite*, білий);

• *границі, осі, заголовок.* Границі по осях абсцис та ординат задаються параметрами *xlim*(), *ylim*, наприклад, *xlim*([1,10]), *ylim*([-2,2]). Відповідно, підписи під осями задають за допомогою *xlabel*, *ylabel.* Підпис графіка задається в параметрі *title.* Параметр *grid on|off* вмикає або вимикає сітку на графіку. Керувати виглядом осей можна за допомогою спеціального об'єкту *gca*();

• *легенда* задається за допомогою параметра *legend*, також можна використовувати *Tex*-нотацію, тому вигляд легенди може бути наближеним до математичної нотації;

• *результат* у вигляді файлу з графіком може бути збережений автоматично у поточному каталозі за допомогою команди *print,* яка підтримує векторні та растрові формати файлів-зображень:, наприклад, $print\ figure1.pdf$, $print\ -djpg\ figure1$. В першому випадку на виході маємо векторний *pdf*-файл, в другому – растровий *jpeg*.

З усіма графічними об'єктами можна працювати через хендлери – змінні, що містять в собі посилання на графічний об'єкт, осі й т. п. (табл. 2.1). Через хендлери встановлюють відповідні властивості графічного об'єкта, що дозволяє отримати добре оформлений репрезентативний графік.

Для побудови графіків можна використовувати різний інструментарій (приклад 2.1). Від його вибору залежить функціональні можливості графіка та його зовнішній вигляд. У версії 5.x стандартним засобом побудови графіків є використання *qt-tooolkit*,

який підходить майже для усіх випадків.

Зазначимо, що зовнішній вигляд графіків у різних підсистемах буде різним, інколи при циклічному переключенні підсистем можливі непередбачувані артефакти на зображенні та тексті. Незважаючи на можливість переключати підсистеми, бажано їх тільки включати за необхідності.

Таблиця 2.1

Функції для роботи із графічними об'єктами

Функція	Опис
h = groot ()	Посилання на кореневий графічний об'єкта
h = gcf ()	Посилання на фігуру (графік)
h = gco ()	Посилання на поточний об'єкт фігури
h = gca ()	Посилання на об'єкт, що описує осі
h = figure(…)	Посилання на новий об'єкт-графік
h = legend("…")	Посилання на новий об'єкт-легенду
set(h, p)	Встановлює властивості (*p*) графічного об'єкта
get(h)	Повертає властивості (*p*) графічного об'єкта
clf	Очищення фігури
delete(h)	Видалення фігури (або іменованого файлу)

Примітка: для всіх функцій групи *gc**() можна передавати посилання на будь-яку складову графіка – фігуру, осі координат, легенду й т. д.

Для отримання довідки по командах у командному вікні *Octave* потрібно використовувати команди *help* або *doc*, наприклад, *help help plot* або *doc legend* (більш докладна довідка).

Приклад 2.1. Робота із підсистемами для побудови графіків

Задача – перегляд та вибір підсистемами для побудови графіків

```
## Приклад 2.1
## Вибір підсистеми побідови графіків
clc, clear all
display("Первірка доступних підсистем")
display(available_graphics_toolkits())
display("Первірка завнтажених підсистем")
display(loaded_graphics_toolkits())
display("Вибір: graphics_toolkit gnuplot")
graphics_toolkit gnuplot
display("Первірка завнтажених підсистем")
display(loaded_graphics_toolkits())
```

Результат:

```
Первірка доступних підсистем
{
  [1,1] = fltk
  [1,2] = gnuplot
  [1,3] = qt
}
Первірка завнтажених підсистем
{}(1x0)
Вибір: graphics_toolkit gnuplot
Первірка завнтажених підсистем
{[1,1] = gnuplot
```

Приклад 2.2. Побудова одновимірного графіка (рис. 2.1)

Задача – побудувати на графіку експериментальні точки.

Для розв'язання цієї задачі використаємо функцію *plot*( ) та хендлери на графічні об'єкти для налашування зовнішнього вигляду

```
## Приклад 2.2
## Побудова наборів даних на графіку
## очистка змінних та командного вікна
clc; clear
## набори даних
X=[0 4 10 15 21 29 36 51 68];
Y1=[63.7 71.0 70.3 80.6 88.7 99.9 92.4 123.6 145.1];
Y2=[37 10 30 60 78 82 99 119 151];
## побудова графіка
plot(X,Y1,'dk',"markersize", 18,"linewidth", 2.25,
     X,Y2,'pk',"markersize", 18,"linewidth", 2.25)
## підпис осей
xlabel('X'); ylabel('Y');
grid on;
## оформлення осей
set(gca, "linewidth", 2, "fontsize", 16,'gridalpha', 0.75)
h = legend ('data ({\itY}_{0}^{1})',
            'data ({\itZ}_{0}^{2})' );
legend (h, "location", "southeast");
set (h, "fontsize", 22);
## збереження результату у файлі формату png
print -dpng data_plot
```

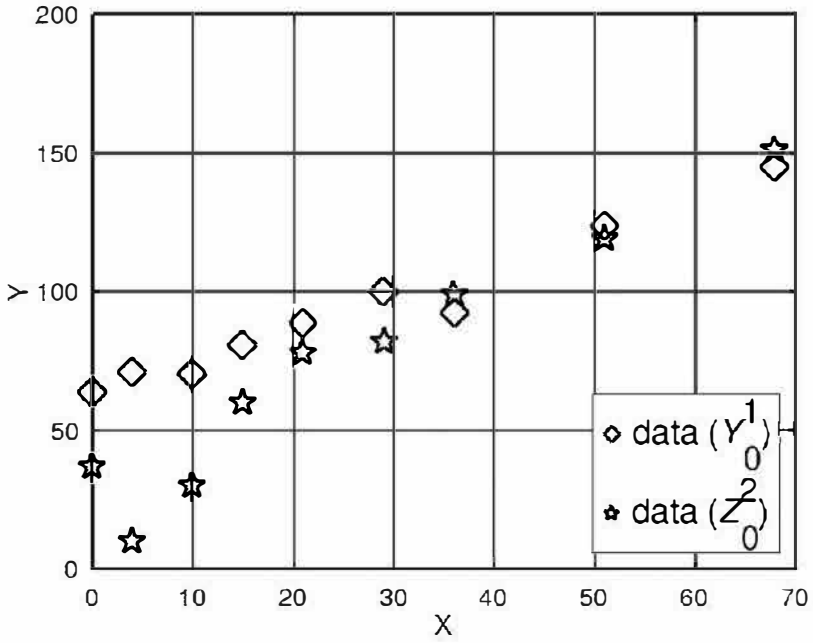


Рисунок 2.1

Приклад 2.3. Побудова одновірного графіка функції (рис. 2.2).

Задача – побудувати графік функції на інтервалі.

У цьому прикладі показано техніку побудови графіків у окремих графічних областях (*subplot*).

Також показано техніку побудови графіків, що мають асимптоти $y = a + \frac{b}{x} + \frac{x}{x^2}$ та $y = \frac{\sqrt{1+x^2}}{x-1}$.

В прикладі 2.3 ми використали відповідні їм анонімні функції $f = @(x)\ a + b\ ./\ x + c\ ./\ x.\hat{}2$ та $z = @(x)\ sqrt(1 + x.\hat{}2)\ ./\ (x\ . - 1)$. Анонімні функції схожі на звичайні функції *Octave*, але з більш жорсткими обмеженнями – не більше двох аргументів та одне значення, що повертається. Символ «@» також використовують, що отримати вказівних на існуючу функцію, наприклад: $pointer\_to\_f = @sin$;

```
## Приклад 2.3
## побудова графіка функції
clear all; delete(gco); clc
subplot(2, 1, 1); % перший графік
a=1; b=-0.35; c=0.05;
x1= -3: 0.1 : -0.1;
x2 = 0.1 : 0.1 : 3;
## анонімна функція
f = @(x)  a + b ./ x + c ./ x.^2;
```

```
hf = plot ( x1 , f(x1) , x2 , f(x2));
set(hf, {'LineWidth'}, {3.25});
set(hf, {'Color'}, {'black'});
title ( 'y = a + b/x + c/x^2' );
xlabel ( 'X' );
ylabel ( 'Y' );
set(gca, "linewidth", 1.5, "fontsize", 16,'gridalpha', 0.75)
grid on;
subplot(2, 1, 2); другий графік
## анонімна функція
z = @(x) sqrt(1 + x.^2) ./ ( x .- 1 ) ;
x1 = -3 : 0.1 : 0.9; x2 = 1.1 : 0.1: 3;
hf = plot ( x1 , z(x1) , '-k' , x2 , z(x2) , '-k' );
set(hf, {'LineWidth'}, {3.25});
set(hf, {'Color'}, {'black'});
title ( 'y=\surd(1+x^2)/(x-1))' );
xlabel ( 'X' );
ylabel ( 'Y' );
set(gca, "linewidth", 1.5, "fontsize", 16,'gridalpha', 0.75)
grid on;
```

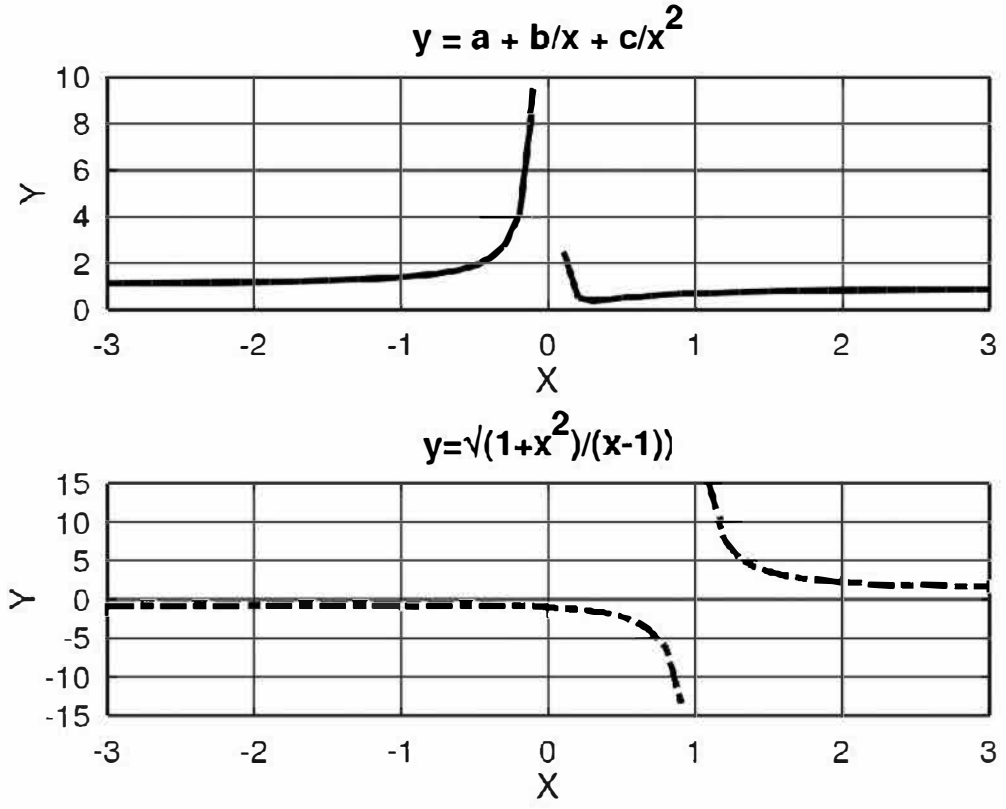


Рисунок 2.2

Приклад 2.4. Побудова графіка функції.

Задача – побудувати графік функції, яку задано аналітично.

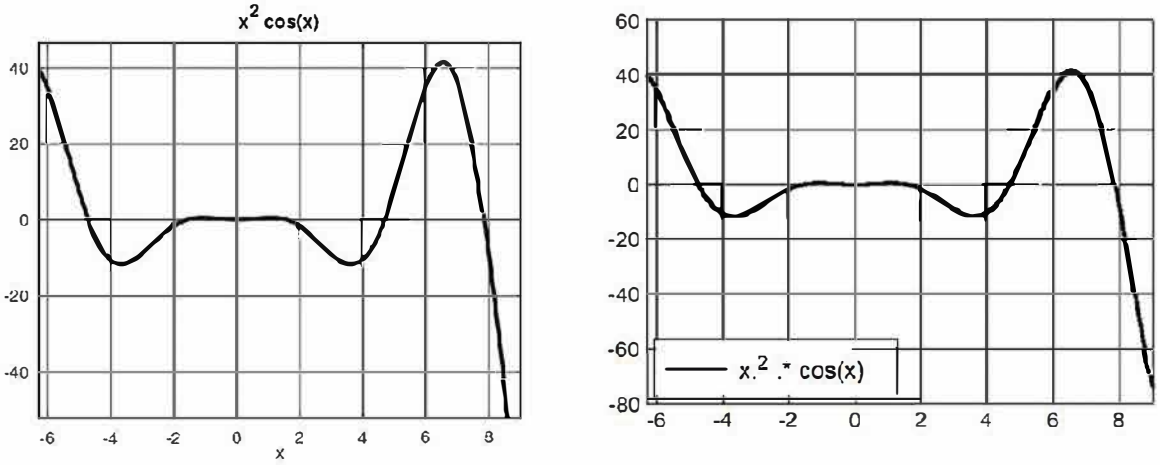
Покажемо використання функції *ezplot( )* для заданої функції (рис. 2.3а). На рис. 2.3б аналогічний результат отримано за допомогою функції *fplot( )*.

Для збереження результату у вигляді графічного файлу застосовано функцію *print( )* із параметрами[8]. Функція *print( )* є достатньо потужним інструментом та дозволяє використовувати великий набір графічних форматів[9].

```
## Приклад 2.4
## побудова графіка ezplot
pkg load symbolic
x = sym("x");
f = inline("x^2*cos(x)");
h = ezplot(f, [-2*pi,9]);
## налаштування осей координат
set(gca, "linewidth", 2, "fontsize", 16,'gridalpha', 0.75)
set(h, "linewidth", 4, "color", "black")
grid on
## збереження графіка у файл
print -dpng ezplot_ex
## побудова графіка fplot
clf;
clear all;
delete(gco);
fplot("x^2 * cos(x)", [-2*pi,9], '-k', 'linewidth', 3.75);
## налаштування осей координат
h = legend(gco);
set (h, "location", "southwest");
set (h, "fontsize", 18);
set(gca, "linewidth", 2, "fontsize", 16,'gridalpha', 0.75)
grid on
## збереження графіка у файл
print -dpng fplot_ex1
```

[8] У *Octave* деякі функції можна визивати у вигляді команди, як у даному прикладі

[9] https://octave.org/doc/interpreter/Printing-and-Saving-Plots.html



а) б)

Рисунок 2.3

Приклад 2.5. Побудова графіків із різними осями координат.

Задача – побудувати на одному графіку дві функції із різними масштаби осей координат.

В *Octave* є функція *plotyy*() для розв'язання цієї задачі (рис. 2.4). Зазначимо, що працює ця функція достатньо повільно, а у *MATLAB* замість неї рекомендують використовувати функцію *yyaxis*( ), яка не реализована в *Octave.*

```
## Приклад 2.5
## побудова двох графіків із різними осями
clf; clear all
x = 0:0.125:2*pi; y1 = sin (x); y2 = exp (x - 1);
## формування графіка
[ax h1 h2] = plotyy (x,   y1,
             x-1, y2,
             @plot, @semilogy);
## параметри осей координат
xlabel ("X"); ylabel (ax(1), "y_1"); ylabel (ax(2), "y_2");
set(ax,{'ycolor'},{'black';'black'})
set(ax, 'linewidth', 2, 'fontsize', 16,'gridalpha', 0.75)
## параметри ліній на графіку
set(h1,'linewidth', 5.75, 'color', 'black', 'linestyle', ':')
set(h2,'linewidth', 5.25, 'color', 'black', 'linestyle', '-')
grid on
print -dpng plotyy_example
```

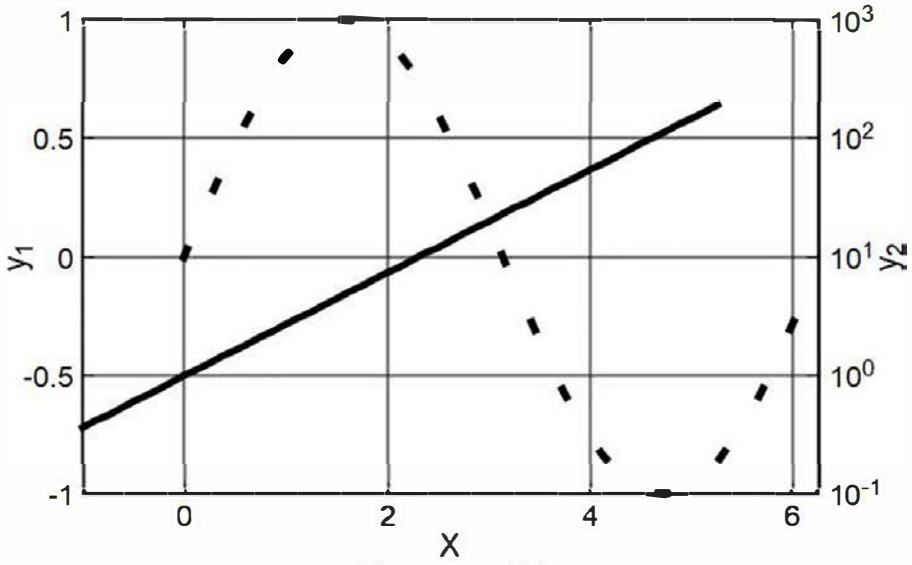


Рисунок 2.4

Приклад 2.6. Графік функції, що задана набором значень.

Задача – на основі файлу даних побудувати графік функції.

Основна умова – правильна структура даних. Найпростіший випадок, коли дані зберігаються у стовпчиках. Якщо дані зберігаються у рядках, або у вигляді якихось інших структур, потрібні додаткові перетворення даних, наприклад функція *reshape*( ) дозволяє змінити структуру масиву. Отриманий графік наведений на рис. 2.5.

```
## Приклад 2.6
## сворюємо свій набір даних
x = (-4*pi : 0.25 : 4*pi);
y =@(x) sin(x) ./ x; ## формуємо масив та заповнюємо його
даними
data = zeros(length(x), 2);
data(:,1) = x;
data(:,2) = y(x);
## зберігаємо дані у текстовий файл
save('-ascii','data.txt','data');
## читаємо дані
saved_data = load('data.txt');
## структура файлу
printf ("Структура файлу:\n")
printf ("x\t\t y\n", data(10,1), data(10,2))
printf ("%f\t %f\n", data(10,1), data(10,2))
printf ("%f\t %f\n", data(11,1), data(11,2))
## будуємо графік
```

```
hf = plot(saved_data(:,1),saved_data(:,2));
set(hf, {'LineWidth'}, {3.25});
set(hf, {'Color'}, {'black'});
title ( 'y = sin(x)/x' );
xlabel ( '\itx' );
ylabel ( '\ity' );
set(gca, "linewidth", 1.5, "fontsize", 16,'gridalpha', 0.75)
grid on
```

Результат:

```
>> Структура файлу:
x                  y
-10.066371         -0.059453
-9.816371          -0.038880
-9.566371          -0.014752
-9.316371          0.011613
```

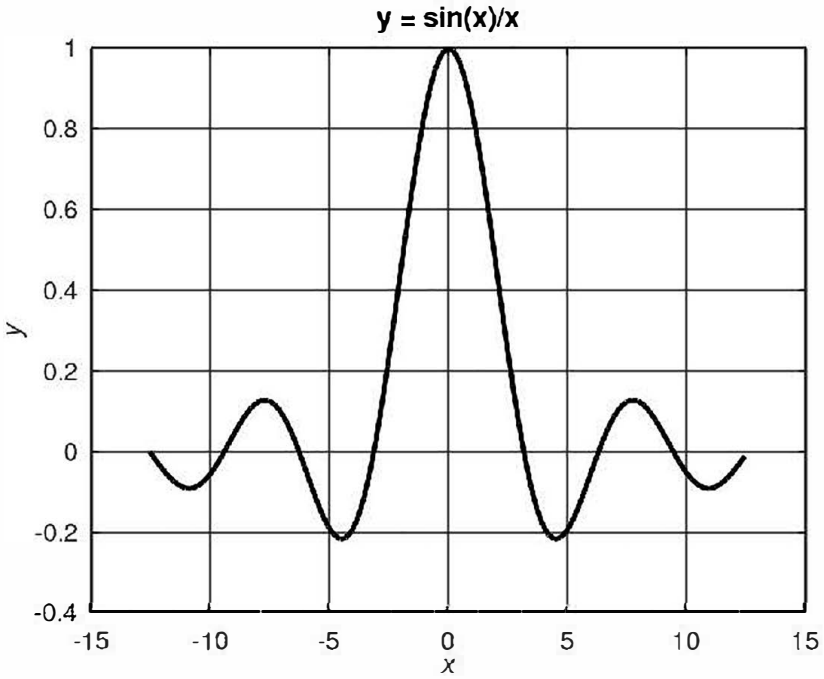


Рисунок 2.5

Ми розібрали основи візуалізації даних в пакеті *Octave* за допомогою побудови одновимірних графіків аналітичних функцій та експериментальних наборів даних. Як правило, його можливостей достатньо для повсякденних практичних задач. Якщо можливостей пакету недостатньо, потрібно формувати файл даних та обробляти його в інших пакетах (*gnuplot, matplotlib, plotty* та ін.)

Побудова аналітичної залежності експериментальних даних за допомогою МНК

Лінійний МНК. Для розв'язання цієї задачі обирають вид аналітичної залежності $y = f(x, a_1, a_2, \dots, a_n)$ та визначають коефіцієнти a_i . У найпростішому випадку шукають параметри прямої $y = a_2 x + a_1$. Відповідно до методу найменших квадратів [7, 8] у кожній точці мінімізують суму квадратів відхилів значень експериментальної (y_i) та розрахованої (Y_i) функцій

$$\varphi(a_1, a_2, \dots, a_n) = \sum_{i=1}^{n} |y_i - Y_i|^2 \to \min \qquad (2.1)$$

Необхідною умовою існування мінімуму функції (2.1) є $\partial\varphi/\partial a_i = 0$. Якщо параметри a_i входять у (2.1) лінійно, отримують систему з n лінійних рівнянь з n невідомими

$$\begin{cases} \sum_{i=1}^{n} \partial\varphi(a_1, a_2, \dots, a_n)/\partial a_1 = 0, \\ \dots \\ \sum_{i=1}^{n} \partial\varphi(a_1, a_2, \dots, a_n)/\partial a_n = 0. \end{cases} \qquad (2.2)$$

Лінійний випадок, $Y = a_1 + a_2 x$. Підстановкою Y в (2.1) та (2.2) отримують систему двох рівнянь з двома невідомими (2.3)

$$\begin{cases} a_1 n + a_2 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i, \\ a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i x_i, \end{cases} \qquad (2.3)$$

розв'язком якої є

$$a_2 = \left[n \sum_{i=1}^{n} y_i x_i - \sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i \right] \Big/ \left[n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2 \right] \qquad (2.4)$$

$$a_1 = \frac{1}{n} \left[\sum_{i=1}^{n} y_i - a_2 \sum_{i=1}^{n} x_i \right].$$

Коефіцієнт кореляції r (2.6) використовують для оцінки залежності між виміряними величинами. Якщо $|r| \to 1$, то існує лінійна залежність між даними, а якщо $|r| \to 0$, то між даними немає лінійного зв'язку.

$$r = \sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}) \Big/ \sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2} \quad (2.5)$$

де $\overline{x}$, $\overline{y}$ – середні арифметичні значення.

В разі коректного розв'язку задачі *t*-критерій Ст'юдента (2.6) має бути більше табличного значення розподілу Ст'юдента для заданого n [7]

$$t = r\sqrt{(n-2)/(1-r^2)}. \quad (2.6)$$

Приклад реалізації МНК в *Octave.*

Розглянемо приклад реалізації методу в *Octave.* Зазначимо, що у наведених нижче прикладах навмисно не використовуються стандартні засоби пакета, а використовується наведений вище алгоритм.

Приклад 2.7. Лінійний МНК

Задача - визначити параметри прямої $y = a_2x + a_1$

```
% Приклад 2.7. Побудова лінії регресії за допомогою МНК
clc; clear
n = 9 % Кількість експерементальних точок
% Вихідні дані
X=[0 4 10 15 21 29 36 51 68];
Y=[63.7 71.0 70.3 80.6 88.7 99.9 92.4 123.6 145.1];
% Визначення параметрів прямої y=a2*x+a1 у вигляді a*x=b
% Складемо СЛАР та визначемо невідомі a1 та a2
x = [n, sum(X); sum(X), sum(X.^2)];
b = [sum(Y); sum(X.*Y)];
% Знайдемо розв'язок системи рівняь
display(x); display("b'=" ); display(b')
a = x \ b; display("a'="); display(a')
```

Результат:

```
n =  9
x =
       9     234
     234   10144
b'=
     835.30000   26452.60000
a'=
   62.4897    1.1662
```

В результаті отримали рівняння прямої $y = 62{,}49x + 1{,}17$. Для перевірки отриманого результату побудуємо відповідні графічні залежності (рис. 2.6), з яких видно шо пряма достатньо добре описує експериментальні дані.

Приклад 2.8. Лінійний МНК

Задача – побудувати графік із результатом (рис. 2.6).

```
% Приклад 2.8
% Побудова графіка лінії y = a2 * x + a1,
% та експериментальних точок в одній графічній області
x=0 : 68;
y=a(2)*x + a(1);
plot(X,Y,'ok',"markersize", 16,"linewidth", 2.5,
     x,y,'--k','linewidth',2.75)
xlabel('X'); ylabel('Y'); grid on;
set(gca, "linewidth", 2, "fontsize", 16,'gridalpha', 0.75)
h = legend ('data', '{\ita}_{2}{\itx}+{\ita}_{1}');
legend (h, "location", "southeast");
set (h, "fontsize", 16);
```

Приклад 2.9. Лінійний МНК

Задача – обчислити коефіцієнт кореляції, середньоквадратичну помилку та критерій Ст'юдента.

```
% Приклад 2.9
% Обчислення коефіцієнта кореляції
% середні значення
mean_X = sum(X) / length(X)
mean_Y = sum(Y) / length(Y)
% чисельник
nom = sum((X.- mean_X).* (Y.- mean_Y));
```

```
% знаменник
denom = sum((X.- mean_X).^2) * sum((Y.- mean_Y).^2);
% коефіцієнт кореляції
r = nom / sqrt(denom)
% t-критерій Ст'юдента
t = r * sqrt(( n-2) / (1 - r^2))
```

Результат:

```
mean_X =  26
mean_Y =  92.811
r =  0.98179
t =  13.675
```

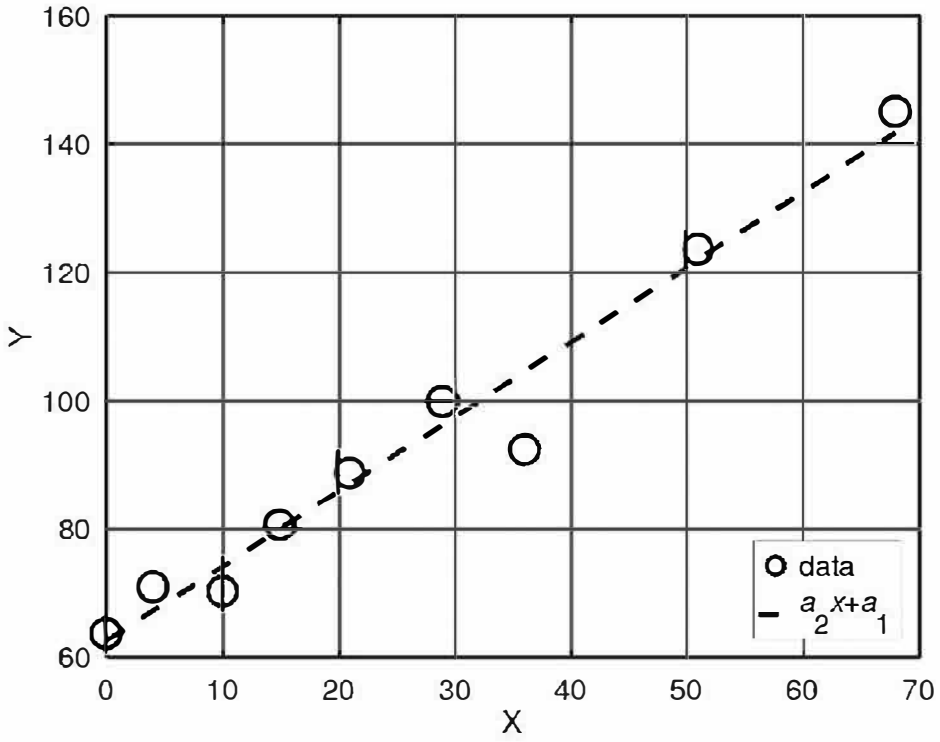


Рисунок 2.6

Таким чином, отримана пряма достатньо точно описує експериментальні дані (з урахуванням кількості точок), що і підтверджується отриманими значеннями коефіцієнтів кореляції та критерія Ст'юдента:

- коефіцієнт кореляції $r = 0{,}98179$;
- критерій Ст'юдента $t = 13{,}675 > (1{,}806 \ldots 5{,}041)$ для набору даних заданої величини.

Резюме

В даній темі ми вивчили МНК для лінійного регресійного аналізу даних. Розробили програму, що його реалізує та отримали параметри прямої $y = a_2 x + a_1$ для набору експериментальних точок. Вивчили можливості пакету *Octave* для побудови двовимірних графіків, навчилися прийомам оформлення зовнішнього вигляду ліній та точок на графіках. Навчилися роботі із компоновкою графіків за допомогою функції $subplot()$.

Функції *Octave*, що вивчаються: $plot()$, $fplot()$, $ezplot()$, $subplot()$, $sum()$, «\», $print()$

Питання для самоперевірки

1. Назвіть приклади використання МНК.
2. Що дає знання аналітичної залежності між даними, які були отримані під час експерименту?
3. Назвіть основні етапи методу найменших квадратів?
4. Для чого використовують коефіцієнт кореляції у МНК?
5. Як сформувати СЛАР для визначення коефіцієнтів регресії?

Практичне завдання

За допомогою МНК визначити параметри прямої, яка найкращим чином описує експериментальні дані.

Порядок виконання завдання

1. Визначити параметри лінії регресії за МНК (2.1–2.6):
   - задати вектори експериментальних значень *x* та *y*;
   - за допомогою функції $plot(x, y)$ побудувати графік виміряних значень;
   - сформувати систему рівнянь (2.3). Розв'язати отриману систему за допомогою команди «\» (див. тему 1);
   - розрахувати коефіцієнт кореляції (2.5);
   - побудувати на одному графіку експериментальні точки та розраховану лінію регресії.
2. Визначити коефіцієнти лінії регресії за допомогою стандартних функцій *Octave*. Порівняти результати, побудувавши на одному графіку експериментальні точки та розраховану лінію регресії.

Індивідуальні завдання

1.

x	1	2	3	4	5
y	2,2	2,8	3,2	3,4	3,8

2.

x	1	2	3	4	5
y	3,15	4,65	5,1	5,25	5,4

3.

x	2	3	4	5	6
y	11,25	9,3	8,25	5,25	4,5

4.

x	1	2	3	4	5
y	5,63	4,65	4,13	2,63	2,25

5.

x	1	2	3	4	5
y	8,2	5,9	4,9	4	3,2

6.

x	1	2	3	4	5
y	7,2	5,9	4,9	4	3,2

7.

x	1	2	3	4	5
y	4,81	5,76	6,91	8,31	9,95

8.

x	1	2	3	4	5
y	2	6,2	8,6	10,4	11,65

9.

x	1	2	3	4	5
y	2,2	0,4	0,22	0,16	0,12

10.

x	1	2	3	4	5
y	0,25	0,09	0,07	0,05	0,04

11.

x	1	2	3	4	5
y	3	3,5	3,67	3,75	3,8

12.

x	1	2	3	4	5
y	2,57	7,15	13	19	27,3

13.

x	1	2	3	4	5
y	3,83	2,55	2,4	2,03	1,91

14.

x	1	2	3	4	5
y	5,1	4,4	3,2	2,7	2,55

15.

x	1	2	3	4	5
y	1,1	1,55	1,9	2,25	2,5

16.

x	1	2	3	4	5
y	1,1	0,2	●,11	0,08	0,06

ТЕМА 3
НЕЛІНІЙНИЙ РЕГРЕСІЙНИЙ АНАЛІЗ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ

Метою є ознайомлення студентів з нелінійним методом найменших квадратів та аналізом експериментальних даних засобами *Octave*.

Постановка задачі. В результаті експериментальних досліджень отримано ряд точок (табл. 2.1) та за допомогою лінійного МНК з'ясували, що між точками не існує лінійної залежності. Побудувати аналітичну залежність $y = f(x, a_1, a_2, \dots, a_n)$, яка найбільш точно описує результати експерименту, а саме для набору даних $(x_i, y_i)_{i=1,2,\dots,N}$ знайти поліном степеня n:

$$p^{(n)}(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_{n-1} x + p_n + 1 = \sum_{i=1}^{k+1} p_i x^{i-1}, \quad (3.1)$$

коефіцієнти якого отримують після розв'язання задачі мінімізації

$$\min_{p_1, p_2, \dots, p_{n+1}} \sum_{i=1}^{N} (p^n(x_i) - y_i)^2 \quad (3.2)$$

Іншими словами, визначити поліном, який найменше відхиляється від експериментальних даних в тому сенсі, що сума квадратів відстаней від заданих точок (x_i, y_i) до розрахованих $(x_i, p^n(x_i))$ буде мінімальною.

Нелінійний МНК. В нелінійному регресійному аналізі перевіряють гіпотезу про те, що експериментальні дані пов'язані деякою нелінійною залежністю. У ряді випадків за допомогою додаткових елементарних перетворень нелінійну функцію заміняють її лінійним аналогом та застосовують лінійний МНК або підібрати функціональну залежність (3.1)–(3.3) та оцінити її адекватність (3.4).

Поліном k-го степеня $Y = \sum_{i=1}^{k+1} p_i x^{i-1}$. Коефіцієнти полінома p_i визначають в результаті розв'язання СЛАР (2.2), в яку підставляють

матриці коефіцієнтів (3.3)-(3.5).

$$\boldsymbol{C} \cdot \boldsymbol{p} = \boldsymbol{b}, \tag{3.3}$$

$$C_{ij} = \sum_{k=1}^{N} x_k^{i+j-2}, i = 1, \dots, k+1, j = 1, \dots, k+1, \tag{3.4}$$

$$b_i = \sum_{k=1}^{N} y_k x_k^{i-1}, i = 1, \dots, k+1. \tag{3.5}$$

Деякі функції є такими, що приводяться до лінійного вигляду за допомогою відповідних підстановок (табл. 3.1), після чого стає можливим застосовати лінійний МНК. Зауважимо, що для функцій, в яких виконується підстановка типу $Y = y^{-1}$, отримана модель признаку (y) є внутрішньо нелінійною.

Тип функціональної залежності між даними вибирають базуючись на фізичних, економічних й т. п. законах або в найпростішому випадку після аналізу графіка, на якому відображають експериментальні точки. Надалі функцію замінюють аналогічним лінійним рівнянням (табл. 3.1), коефіцієнти якого підставляють в (2.1) та розв'язують отриману СЛАР будь-якими методами та знаходять відповідні коефіцієнти прямої a_1, a_2. Після перевірки результатів повертаються до вихідної моделі та виконують зворотні підстановки (приклад 3.1), після чого перевіряють адекватність отриманих результатів, розраховуючи індекс кореляції та сумарну квадратичну помилку (3.6).

На жаль, не завжди можна функціональними перетвореннями від нелінійних моделей перейти до лінійних. Крім того, потрібно мати на увазі, що при обчисленні параметрів за МНК мінімізується сума квадратів відхилень перетворених, а не вихідних даних.

У разі адекватно підібраної функції сумарна квадратична помилка наближається до нуля, а індекс кореляції R близький до одиниці

$$\sum_{i=1}^{N} [y_i - Y_i]^2 \to 0, R = \sqrt{1 - \sum_{i=1}^{N} (y_i - Y_i)^2 \Big/ \sum_{i=1}^{N} (y_i - \overline{y})^2} \to 1. \tag{3.6}$$

Таблиця 3.1

Функції, які приводяться до лінійного вигляду[10]

Функція	Заміни	Лінійне рівняння
$y = ax^b e^{cx}$	$Y = \ln y$ $X = \ln x$ $A = \ln a$	$Y = A + bX + cx$
$y = ax^b$		$Y = A + bX$
$y = ae^{bx}$		$Y = A + bx$
$y = \frac{1}{ax + b}$	$Y = \frac{1}{y}$	$Y = ax + b$
$y = \frac{1}{\frac{a}{x} + b}$	$Y = \frac{1}{y}, X = \frac{1}{x}$	$Y = aX + b$
$y = \frac{1}{(ae^x + b)}$	$Y = \ln y, X = e^x$	$Y = aX + b$
$y = a \ln x + b$	$X = \ln x$	$y = aX + b$

Степінь полінома повинен бути меншим за кількість заданих точок для того, щоб такий поліном був єдиним. Наприклад, якщо задано три точки, то їх можна наблизити поліномом нульового (точка), першого (пряма) або другого (параболою) степенів.

Причому, парабола буде точно проходити через три задані точки (сума квадратів відстаней дорівнюватиме нулю), оскільки три коефіцієнти квадратичного полінома однозначно визначаються з трьох умов проходження через задані точки. В даному випадку, ми отримаємо вже не наближення даних, а їх інтерполяцію. На практиці, як правило, застосовують поліноми не надто високих ступенів[11].

Приклад 3.1. Перевірка гіпотези про закон розподілення експериментальних даних.

Задача – вважати, що дані описуються функціональною залежністю $y(x) = \frac{1}{a/x+b}$, визначити коефіцієнти a та b.

Замінимо функцію її лінійним аналогом $Y(X) = aX + b$ (див. табл. 3.1) та застосуємо лінійний МНК.

```
## Приклад 3.1
## Перевірка гіпотези за лінійним МНК
clear all; clc; delete(gco())
X0=[0.56;  0.8 ; 1.8; 3.2; 4.8; 5.05; 7.1; 8.25; 9.05; 9.55];
```

[10] За умови дотримання обмежень на область визначення функції

[11] Див. приклади далі в цій темі

```
Y0=[0.46; 1.05; 1.75; 2.65; 2.83; 2.91; 2.98; 3.0; 3.03;3.06];
## підготовка вхідних даних
Y = 1./Y0; X = 1./X0; n = length(Y);
x = [n,  sum(X); sum(X), sum(X.^2)];
b = [ sum(Y);  sum(X.*Y) ];
## Знайдемо розв'язок системи рівняь
a = x \ b
printf("Рівняння прямої: y=%f*x + %f\n", a(2), a(1))
## Побудова графіка лінії y = a2 * x + a1,
## та експериментальних точок в одній графічній області
subplot(2,1,1)
x=linspace(0.1 , max(X));
y=a(2)*x + a(1);
plot(X,Y,'ok',"markersize", 16,"linewidth", 2.5,
     x,y,'--k','linewidth',2.75)
xlabel('X'); ylabel('Y'); grid on;
set(gca, "linewidth", 2, "fontsize", 16,'gridalpha', 0.75)
h = legend ('data', '{\ita}_{2}{\itx}+{\ita}_{1}');
legend (h, "location", "southeast");
title ("Лінійний МНК"); set (h, "fontsize", 16);
## Обчислення коефіцієнта кореляції
mean_X = sum(X) / length(X);
mean_Y = sum(Y) / length(Y);
nom = sum((X.- mean_X).* (Y.- mean_Y));
denom = sum((X.- mean_X).^2) * sum((Y.- mean_Y).^2);
r = nom / sqrt(denom);
t = r * sqrt(( n-2) / (1 - r^2));
printf("Коефіцієнт кореляції r=%f, критерій Ст’юдента t=%f\n",
r, t)
#### перевірка результату
f = @(x, a, b) 1./(a ./ x + b);
x=linspace(0.1,10); y=f(x, a(2), a(1));
subplot(2,1,2)
plot(X0,Y0,'ok',"markersize", 16,"linewidth", 2.5,
     x,y,'-k','linewidth',2.75)
xlabel('X'); ylabel('Y'); grid on;
set(gca, "linewidth", 2, "fontsize", 16,'gridalpha', 0.75)
h = legend ('data', '1/({\ita}/{\itx}+{\itb})');
legend (h, "location", "southeast");
title ("Результат");set (h, "fontsize", 16);
## перевірка результату
y = Y0; Y = f(X0, a(2), a(1));
```

```
reshape(Y, 1, length(Y));
err = sum((y - Y).^2)
R = sqrt(1 - sum((y - Y) .^ 2) / sum((y - mean(y)) .^ 2))
```

Результат обчислень

```
a =
   0.14974
   0.96095
Рівняння прямої: y=0.960952*x + 0.149736
Коефіцієнт кореляції r=0.951514, критерій Ст'юдента t=8.749183
err =  2.8625
R =  0.80240
```

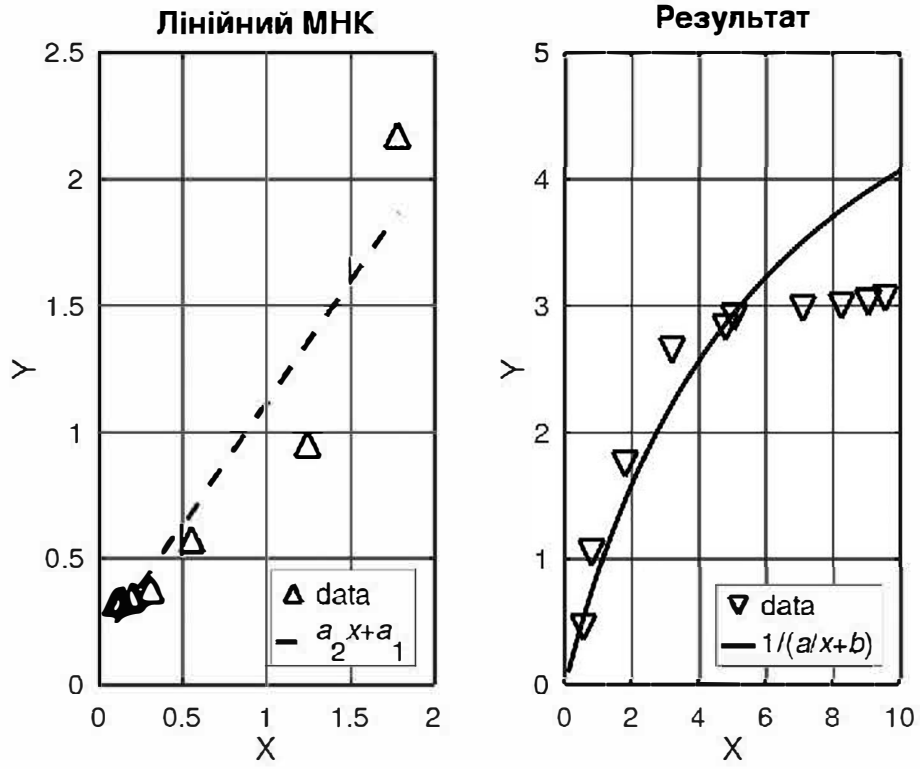


Рисунок 3.1

В результаті роботи програми отримали коефіцієнти $a = 0{,}96$ та $b = 0{,}14$. Аналіз результатів (рис. 3.1) показує, що пряма проходить доволі близько до набору даних $(X, Y_i)_{i=1,2,\ldots,N}$, коефіцієнт кореляції та критерій Ст'юдента в границях допустимих значень, при цьому аналітична крива $y(x)$ відповідає вхідним даним тільки в частині діапазону, що підтверджується розрахованими значеннями сумарної квадратичної помилки $err = 2{,}8625$, яка суттєво відрізняється від нуля та коефіцієнта кореляції $R = 0{,}8024$, який не наближається до одиниці.

Отже лінеаризація функції $y(x)$ в даному випадку дала незадовільний результат.

Функції *Octave* для наближення даних за МНК

• $polyfit(x, y, k)$. Параметри x, y – масиви вхідних даних, k – степінь полінома. Функція $polyfit$ повертає матрицю коефіцієнтів a, починаючи з найбільшого степеня;

• $polyval(p, x)$ – обчислення значення полінома p в точці x;

• $spline\ (x, y, breaks)$ – апроксимація сплайнами;

• $polyder$ (c) – обчислення похідної від полінома, заданого вектором коефіцієнтів c.

Для функції $polyfit()$ у вхідних аргументах передають вектори з даними, а на виході отримують вектор коефіцієнтів полінома, починаючи зі старшого степеня. Функція $polyfit()$ може бути викликана додатковими аргументами для поліпшення якості наближення та отримання деякої додаткової інформації про нього.

Поставимо задачу наблизити дані, які задані масивами X та Y поліномами другого-п'ятого степенів за допомогою стандартної функції $polyfit()$, вказавши в її вхідних аргументах вектор з даними і степені поліномів. Самі коефіцієнти визначимо як масиви $p2 - p5$, відповідно.

Приклад 3.2. Наближення даних поліномом.

Задача – для заданих точок знайти коефіцієнтів поліномів 2-5 степенів.

```
% Приклад 3.2
X = [2.65 2.53 2.4 1.65 1.12 0.72 0.5 0.42 0.28  0.12 ];
Y = [-2.75 -1.05 0.9  2.45 2.1 1.95 0.1 -1.2  -2.4 -2.6];
p2 = polyfit(X, Y, 2)
p3 = polyfit(X, Y, 3)
p4 = polyfit(X, Y, 4)
p5 = polyfit(X, Y, 5)
p2 = -3.4172   9.8999  -4.2471
p3 = -0.43212  -1.66388   8.06923  -3.83962
p4 = -1.1801    5.8320  -12.2175   14.1998   -4.7528
p5 = -2.3980   15.4349  -35.0142   30.0179   -2.6504   -2.9016
```

Для побудови графіків кривих, що описуються отриманими

поліномами, слід знайти їх значення в проміжних точках, що належать інтервалу, на якому задані дані, тобто між $x(1)$ і $x(end)$. Для цього потрібно згенерувати $N = 50$ точок, рівномірно розташованих в області визначення даних, за допомогою функції $linspace(\,)$ та обчислити в них значення поліномів $p^{(2)}, p^{(3)}, p^{(5)}$ за допомогою функції $polyval(\,)$. Результат зберегти у масивах $fn2$, $fn3$, $fn5$ відповідно.

Вхідними аргументами функції $polyval()$ в найпростішому випадку є вектор коефіцієнтів полінома і вектор значень незалежної змінної, для яких потрібно обчислити значення полінома, а вихідним – вектор значень полінома. Для перевірки побудувати вихідні точки і отримані криві на одному графіку (рис. 3.2).

Приклад 3.3. Наближення даних поліномом.

Задача – обчислити значення полінома за допомогою $polyval(\,)$.

```
% генеруємо 50 значень аргументу
x = linspace(X(1), X(end), 50);
% обчислення значення полінома
fn2 = polyval(p2, x);
fn3 = polyval(p3, x);
fn4 = polyval(p4, x);
fn5 = polyval(p5, x);
```

Приклад 3.4. Наближення даних поліномом.

Задача – побудувати графік кривої, що описується поліномом.

```
% побудова графіка
subplot(2,1,1)
plot(X, Y, 's',
      x, fn2,  "linewidth", 1.75, x, fn3, "linewidth", 1.75,
      x, fn4 , "linewidth", 1.75, x, fn5 , "linewidth", 1.75)
% налаштування виглдяу осей координат та легенди
xlabel('X'); ylabel('Y'); grid on
set(gca, "linewidth", 2, "fontsize", 16)
h = legend ('data','{\itp}^{(2)}({\itx})',
                   '{\itp}^{(3)}({\itx})';
legend (h, "location", "northeastoutside");
set (h, "fontsize", 16);
```

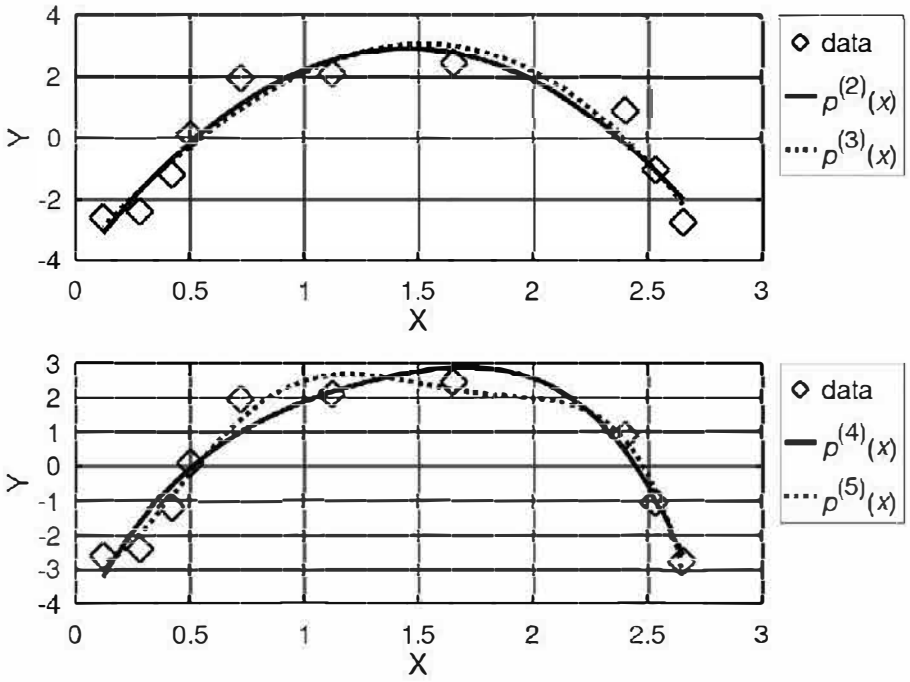


Рисунок 3.2

Для того, щоб дізнатися, наскільки далеко відстає графік кривої полінома від заданих точок, тобто яка була допущена помилка при наближенні даних, слід викликати функцію*polyfit*() з двома вихідними аргументами. У перший з них запишеться вектор коефіцієнтів побудованого полінома, а в другій структура з інформацією про наближення (норму), чим менший цей показник, тим краще.

Приклад 3.5. Наближення даних поліномом.

Задача – обчислити та надрукувати похибки обчислень.

```
## друк похибки
for i = [2:5]
  [p, S] = polyfit(X, Y, i);
   printf("Степінь: %d. Похибка: %f, df %d\n", i, S.normr,
S.df)
endfor
```

Результат:

```
Степінь: 2. Похибка: 1.931491, df 7
Степінь: 3. Похибка: 1.880512, df 6
Степінь: 4. Похибка: 1.672041, df 5
Степінь: 5. Похибка: 1.157744, df 4
```

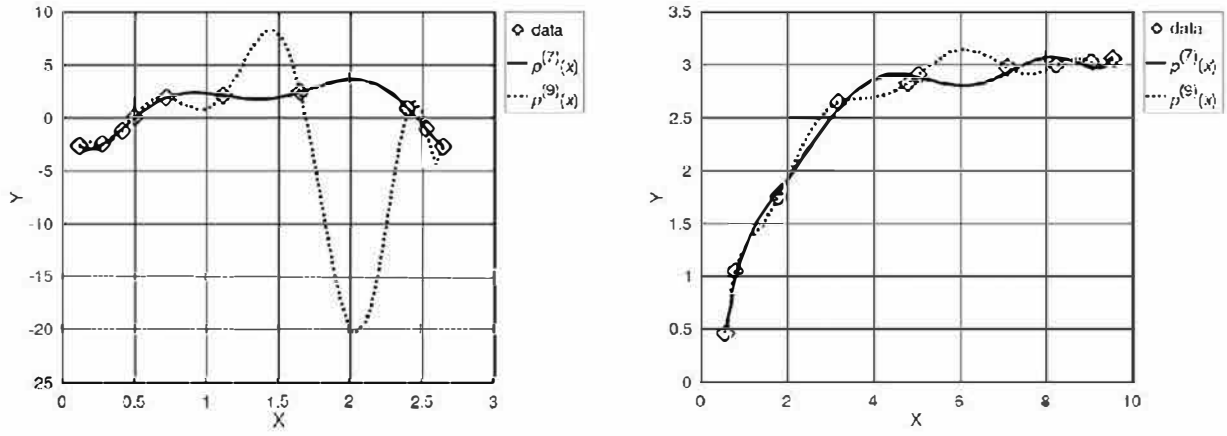
Приклад 3.6. Наближення даних поліномом.

Задача – обчислити значення поліномів 7 та 9 степенів.

```
## Приклад 3.6
p7 = polyfit(X, Y, 7), p9 = polyfit(X, Y, 9)
polyout(p7), polyout(p9)
## обчислення значення полінома
fn7 = polyval(p7, x); fn9 = polyval(p9, x);
for i = [7,9]
  [p, S] = polyfit(X, Y, i);
   printf("Степінь: %d. Похибка: %f, df %d\n", i, S.normr,
S.df)
endfor
```

Результат:

```
p7 =
0.03972*s^7 + 4.767*s^6 - 43.242*s^5 + 143.8*s^4 - 222.7*s^3 +
159.5*s^2 - 40.109*s^1 + 0.29243
p9 =
43.189*s^9 - 516.92*s^8 + 2574.6*s^7 - 6919.5*s^6 + 10894*s^5
- 10223*s^4 + 5578.7*s^3 - 1654.9*s^
2 + 240.71*s^1 - 15.427
Степінь: 7. Похибка: 0.275276, df 2
Степінь: 9. Похибка: 0.000000, df 0
```



а) б)

Рисунок 3.3

Не завжди підвищення степеня полінома призводить до кращого результату. У цьому можна переконається побудувавши поліноми 7, 8, 9, …, *n*-степенів. При тому, що помилка наближення буде

зменшуватися (до певної точки), якість наближення поза експериментальних точок може стати незадовільною (рис. 3.3а), але для першого прикладу результат буде іншим (рис. 3.3б).

Приклад 3.7. Наближення даних поліномом.

Задача – обчислити коефіцієнти полінома за наведеним алгоритмом МНК (3.3)-(3.6).

```
% Приклад 3.7
clear all, clc
N = 4; % степінь полінома + 1
X=[0.56;  0.8 ; 1.8; 3.2; 4.8; 5.05; 7.1; 8.25; 9.05; 9.55];
Y=[0.46; 1.05; 1.75; 2.65; 2.83; 2.91; 2.98; 3.0; 3.03; 3.06];
b = zeros(N, 1); C = zeros(N, N);
# визначення коефіцієнтів  для СЛАР
for i = 1:N
  for j = 1:N
    C(i,j) = sum(X.^(i+j-2));
  end
  b(i) = sum(Y.*X.^(i-1));
end
printf("Вихідні матриці:\n")
display('X ='); display(X');    display('Y ='); display(Y');
display(C);     display('b ='); display(b');
## Розв'язок
printf("Коефіцієнти полінома:\n"); p = C \ b; (flip(p))'
printf("Поліном 4 степеня (polyfit):\n");      polyout(p)
```

Результат:

```
Вихідні матриці:
X =  0.56 0.80 1.80 3.20 4.80 5.05 7.10 8.25  9.05 9.55
Y =  0.46 1.05 1.75 2.65 2.83 2.91 2.98 3.00 3.03 3.06
C =
        10.00000     50.16000       354.55360      2810.29537
        50.16000    354.55360      2810.29537     23496.66977
       354.55360   2810.29537     23496.66977    202591.20297
      2810.29537  23496.66977    202591.20297   1781341.44901
b =
     23.720    143.560   1054.691   8447.629
Коефіцієнти полінома:
ans = 0.010177  -0.208234   1.396564  -0.110295
```

```
Поліном 4 степеня (polyfit):
-0.1103*s^3 + 1.3966*s^2 - 0.20823*s^1 + 0.010177
```

Резюме

В даній теми ми навчилися використовувати нелінійний варіант МНК для аналізу експериментальних даних, а саме складати та розв'язувати СЛАР засобами *Octave* для розрахунку коефіцієнтів поліному n-го степеня.

Ознайомилися із деяким стандартними функціями, призначеними для роботи із поліномами. Розробили та протестували програми розрахунків коефіцієнтів поліномів на основі алгоритму нелінійного МНК та за допомогою функції $polyfit(\,)$. Переконалися, що заміна нелінійної функції її лінійним аналогом не завжди призводить до прийнятного результату. Навчилися прийомам складного форматування графіків засобами пакету.

Функції *Octave*, які вивчаються в темі: $polyfit(\,)$, $polyval(\,)$, $sum(\,)$, $mean(\,)$, $plot(\,)$, оператор «\».

Питання для самоперевірки

1. В яких задачах у вашій предметній галузі виникає необхідність застосування нелінійного МНК?
2. Опишіть алгоритм обчислення коефіцієнтів полінома k-го степеня.
3. Як оцінити адекватність обраної функціональної залежності?
4. Як нелінійну функцію привести до лінійного вигляду?
5. Як степень полінома впливає на криву регресії? Що відбувається за значного підвищення степеня полінома ($k > 7 \ldots 10$)?

Практичне завдання

За допомогою МНК та вбудованих функцій *Octave* визначити параметри кривої, яка найкращим чином описує експериментальні дані.

Порядок виконання завдання

1. Задати вектори x та y вихідних точок та побудувати їх графік.
2. За допомогою функції $polyfit(\,)$ обчислити коефіцієнти для декількох поліномів 3-7 степенів. Оцінити адекватність підібраної функції за допомогою (3.4). Побудувати графіки, порівняти результати з експериментальними даними, зробити висновки.

3. Обчислити коефіцієнти полінома, використовуючи (3.1)–(3.3) та вилучивши один-два коефіцієнти полінома, наприклад, два старші парні степеня. Оцінити адекватність підібраної функції, побудувати відповідні графіки, зробити висновки.

4. Визначити аналітичну функцію, що найкраще відповідає експериментальним даним (типи функцій див. рис. 3.7), застосувавши масштабний коефіцієнт до даних (за необхідності).

5. Використовуючи функцію *spline*( ), знайти сплайн, що найкращим чином описує вхідні дані. Порівняти із результатами апроксимації поліномами, побудувавши спільний графік. Зробити висновки.

Індивідуальні завдання

Варіант 1

X	0,45	0,61	1,68	2,75	3,82	4,89	5,96	7,05	8,11	9,77
Y	0,40	0,53	4,35	11,7	25,6	36,30	54,20	75,93	102	132

Варіант 2

X	0,37	1,44	2,51	3,58	4,65	5,72	6,79	7,86	8,93	10,0
Y	5,32	6,69	12,45	14,86	18,42	29,55	47,42	69,20	94,0	126

Варіант 3

X	0,27	1,35	2,43	3,51	4,59	5,67	6,75	7,83	8,92	10,0
Y	3,65	4,82	5,26	5,54	5,74	5,9	6,05	6,14	6,24	6,45

Варіант 4

X	0,24	1,32	2,40	3,48	4,56	5,64	6,72	7,80	8,88	9,97
Y	0,01	0,49	3,42	5,40	6,61	7,66	8,53	9,27	9,92	10,49

Варіант 5

X	4,60	5,71	6,86	7,97	9,0	10,2	11,3	12,38	13,49	14,60
Y	19,25	15,56	13,95	13,00	12,27	11,75	11,42	10,88	10,25	10,06

Варіант 6

X	4,70	5,8	6,9	8,1	9,15	10,45	11,45	12,78	13,35	14,71
Y	14,7	13,25	11,65	11,01	10,55	9,91	9,52	9,05	8,9	8,45

Варіант 7

X	4,5	5,7	6,85	8,2	9,04	10,1	11,45	12,3	13,85	15,25
Y	3,21	35,3	80,3	96,2	102	107	110	113	114	117

Варіант 8

X	4,8	5,9	7,0	8,5	9,1	10,6	11,5	12,6	13,93	14,8
Y	12,5	10,4	9,5	9,05	8,8	8,42	8,1	7,99	7,88	7,6

Варіант 9

X	1,21	2,0	2,88	3,55	4,54	5,37	6,2	7,05	7,88	8,71
Y	33,2	31,5	31,0	29,2	26,5	24,1	19,7	14,3	7,1	1,25

Варіант 10

X	1,2	2,12	3,0	3,75	4,85	5,74	6,75	7,59	8,5	9,4
Y	172	169	166	165,5	164,9	160,2	157,1	151	147	138

Варіант 11

X	1,44	2,3	3,26	4,1	5,09	5,78	6,95	7,8	8,85	9,99
Y	31,1	40	40,7	40,4	40,2	39,8	39,45	39,05	38,9	37,2

Варіант 12

X	1,36	2,27	3,22	4,2	5,02	5,88	6,99	7,74	8,65	9,33
Y	89,0	88,6	87,8	87,1	86,0	84	83,2	81,7	79,5	77,6

Варіант 13

X	1,5	2,39	3,2	4,2	5,4	6,0	6,9	7,85	8,9	9,9
Y	12,3	12,39	12,50	12,7	13,35	14,0	15,0	17,2	19,8	25

Варіант 14

X	1,3	2,2	3,1	4,2	4,9	5,9	6,7	7,7	8,6	9,75
Y	10,0	10,3	10,7	11,4	12,6	14,7	18,1	23,2	30,1	38,5

Варіант 15

X	0,79	1,8	2,7	4,2	4,79	5,82	6,79	7,8	8,4	9,9
Y	116	114	112	110	107,9	106	103	100,42	97,4	93

Варіант 16

X	0,8	1,78	2,8	3,81	4,82	5,9	6,8	7,8	8,8	9,8
Y	1,75	5	9,2	14,3	19,7	25,7	33	39,8	47,2	56

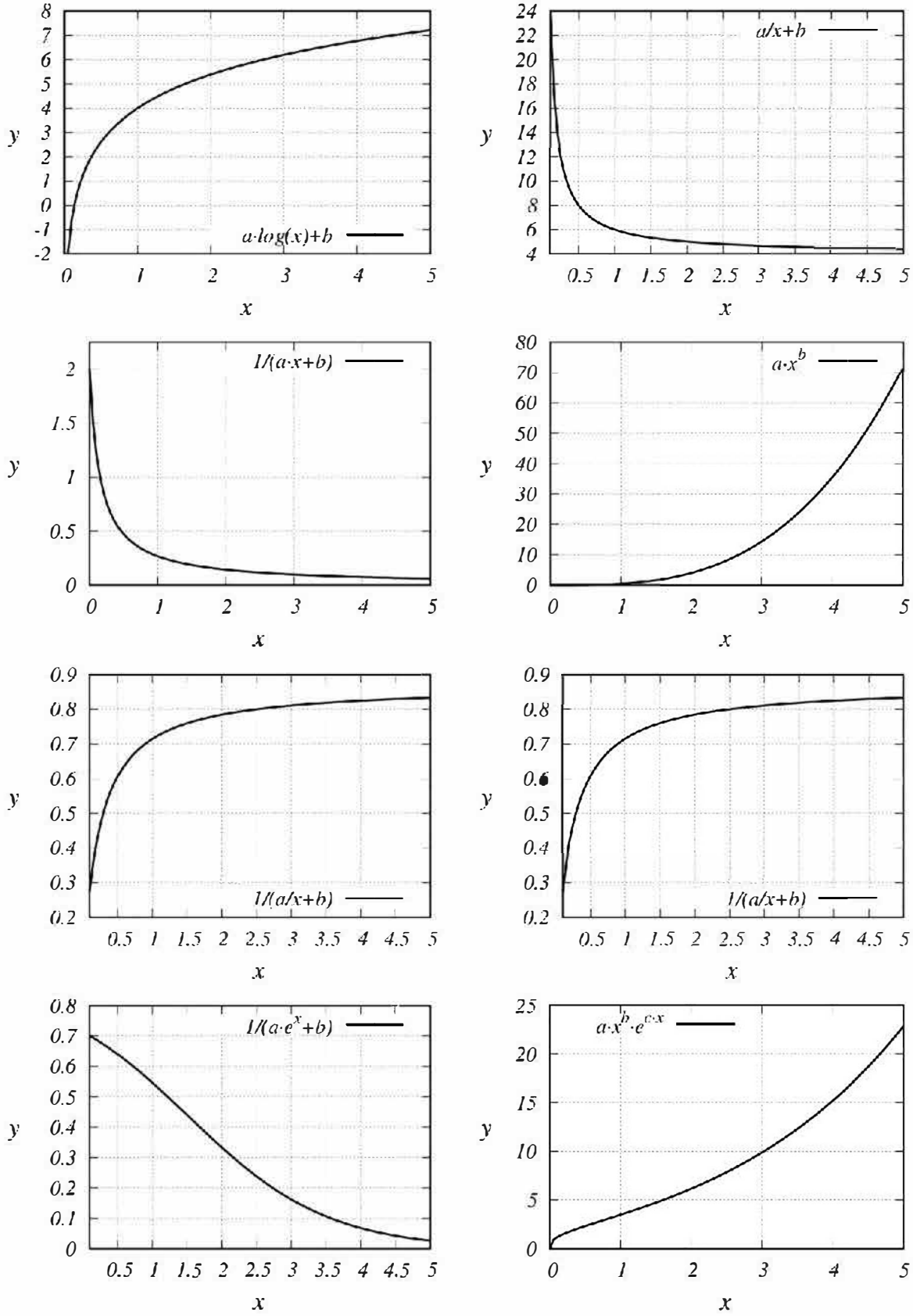


Рисунок 3.7

ТЕМА 4
ПРЯМІ МЕТОДИ ПОШУКУ МІНІМУМУ ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ

Метою є ознайомлення з прямими методами пошуку мінімуму однієї змінної, основам програмування алгоритмів основних методів оптимізації та застосуванням стандартних функцій *Octave*.

Постановка задачі. Нехай цільова функція $y = f(x)$ є функцією однієї змінної x та має мінімум на заданому інтервалі $[a, b]$. Визначити мінімум цільової функції із заданою похибкою ε.

Задачі на пошук мінімуму (максимуму) функції зустрічаються в багатьох галузях науки та техніки та відомі як мінімум з античної Греції. Теорія екстремальних задач почала розвиток з початку 17-го сторіччя. У [17] наведено історію розвитку даного напрямку математики, чисельних методів, а тепер й програмування. Класичні методи оптимізації базуються на оцінках інтервалу існування мінімуму та поступового його звуження. В даній темі ми зосередимося на розгляді та реалізації методів прямого пошуку. Вміння реалізовувати класичні алгоритми є корисною вправою для програміста, розширюють його світогляд та підвищують його конкурентоздатність. Пригадаємо з курсу математики:

- точку x_0 називають точкою локального максимуму функції $f: D \to R$, якщо $\exists U_D(x_0): \forall x \in U_D(x_0) f(x) \geq f(x_0)$;
- точку x_0 називають точкою локального мінімуму функції $f: D \to R$, якщо $\exists U_D(x_0): \forall x \in U_D(x_0) f(x) \leq f(x_0)$.

Якщо знак нерівності строгий, то отримаємо строгий локальний мінімум або максимум. Значення функції $f(x)$ в точці максимуму називається локальним максимумом, значення функції в точці мінімуму – локальним мінімумом даної функції. Локальні максимум і мінімум функції називаються локальними екстремумами.

Всі методи пошуку мінімуму функції, що будуть розглядатися в темах 4-5, не використовують інформацію про похідну функції, тому не вимагають складних обчислювальних процедур.

Загальна схема прямих методів пошуку мінімуму на інтервалі

Функцію $f(x)$, $x \in D = [a, b]$ називають унімодальною на D, якщо існує така точка $x^* \in D$, що

$$\begin{aligned} &f(x_1) > f(x_2), \qquad x^* > x_2 > x_1, \\ &f(x_1) < f(x_2), \; x^* < x_1 < x_2, \;\; x_1, x_2 \in D. \end{aligned} \tag{4.1}$$

Якщо унімодальна функція є неперервною, то вона має єдину точку мінімуму на D, яка співпадає з x^*(рис.4.1 *а*).

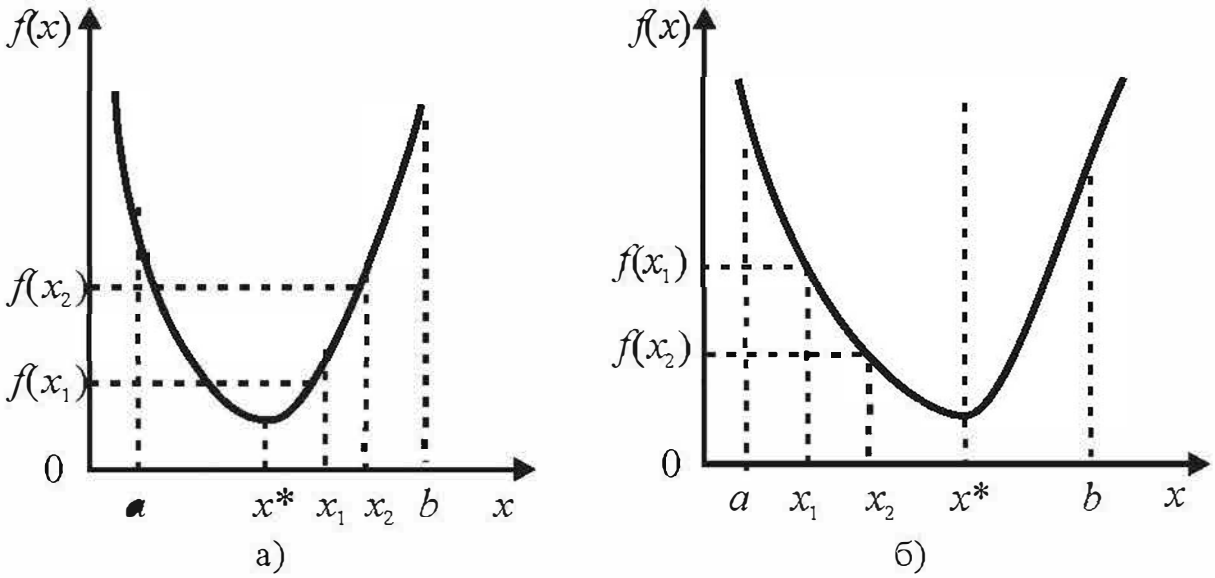


Рисунок 4.1 – Локалізація інтервалу пошуку мінімуму

Для *локалізації інтервалу, що містить мінімум,* обчислюють значення функції у двох точках x_1 та x_2 на інтервалі $[a, b]$ $(a < x_1 < x_2 < b)$. Із властивості унімодальності (4.1) можна зробити висновок, що мінімум розташовано або на $[a, x_2]$, або на $[x_1, b]$.

Якщо $f(x_1) < f(x_2)$, то мінімум не може знаходитися на відрізку $[x_2, b]$, для його виключення приймають $b = x_2$ (рис. 4.1 *а*).

Якщо $f(x_1) > f(x_2)$, то мінімум не може знаходитися на відрізку $[a, x_1]$, для його виключення приймають $a = x_1$ (рис. 4.2 *б*).

У випадку $f(x_1) = f(x_2)$ мінімум буде на $[x_1, x_2]$, тому $a = x_1$, $b = x_2$.

Коли величина інтервалу, що містить мінімум, стає меншою за задану похибку ε, обчислення припиняють.

Алгоритм локалізації інтервалу, що містить мінімум.

Задати точку x_0 і деяку додатну величину Δ.

Крок 1. Якщо $f(x_0) > f(x_0 + \Delta)$, то $k = 1$, $x_1 = x_0 + \Delta$, $h = \Delta$. Інакше, якщо $f(x_0) > f(x_0 - \Delta)$, то $x_1 = x_0 - \Delta$, $h = -\Delta$.

Крок 2. $h = 2h$, $x_{k+1} = x_k + h$.

Крок 3. Якщо $f(x_k) > f(x_{k+1})$, то $k = k + 1$ та повертаються до кроку 2, інакше відрізок $[x_{k-1}, x_{k+1}]$ містить точку мінімуму.

Методи одновимірного пошуку відрізняються способом вибору (обчислення) точок x_1, x_2.на кожній ітерації[12]. Ефективність відповідних алгоритмів оцінюють за кількістю обчислень функції, що необхідні для досягнення заданої точності.

Метод рівномірного пошуку. Інтервал $[a, b]$, де знаходиться мінімум, ділять на однакові відрізки розміром $\Delta x = \varepsilon$. Для N-точок $(N = (b - a)/\Delta x)$ обчислюють значення функції та знаходять найменше з них, аргумент якого і є оптимальним значенням x. Це достатньо неефективний метод, що вимагає великої кількості обчислень.

Метод дихотомії (перший варіант). Задають $\varepsilon > 0$, визначають початкове наближення $x_0 = (a + b)/2$.

Крок 1. Якщо $b - a < \varepsilon$, то $x^* = (a + b)/2$ – точка мінімуму, закінчують обчислення, інакше $x_1 = (x_0 + a)/2$ та $x_2 = (x_0 + b)/2$.

Крок 2. Якщо вірною є умова$f(x_1) < f(x_0)$, то виконують присвоєння $b = x_0$ та $x_0 = x_1$, щоб виключити інтервал $[x_0, b]$, та повертаються на на крок 1, інакше – крок 3.

Крок 3. Якщо вірною є умова$f(x_2) < f(x_0)$, то виконують присвоєння $a = x_0$ та $x_0 = x_2$, щоб виключити інтервал $[a, x_0]$ та повертаються на крок 1, інакше $a = x_1$, $b = x_2$ та повертаються на крок 1.

Метод дихотомії (другий варіант). Точки x_1, x_2 розташовано на відстані $\Delta < \varepsilon$ від середини $[a_i, b_i]$

$$x_1 = \frac{a_i + b_i - \Delta}{2}, x_2 = \frac{a_i + b_i + \Delta}{2}. \qquad (4.2)$$

За одну ітерацію нтервал невизначеності зменшується вдвічі (4.2). За n-ітерацій його величина буде дорівнювати $(b - a)/2^n$.

Задану точність отримують за $n \geq \ln((b - a)/\varepsilon)/\ln 2$ ітерацій. На кожній ітерації цільову функцію обчислюють двічі.

Алгоритм *методу дихотомії* (*другий варіант*).

Задати $\varepsilon > 0$, $\Delta < \varepsilon$, визначити відрізок локалізації мінімуму $[a, b]$.

---

[12] лат. *iteratio* «повторення», один прохід циклічного алгоритму

Крок 1. Обчислюють точку $x_0 = (a + b)/2$. За формулою (4.2) обчислюють x_1, x_2. Якщо $f(x_1) > f(x_2)$, то $a = x_0$, інакше $b = x_0$, що призводить до виключення відповідного інтервалу.

Крок 2. Якщо $b - a < \varepsilon$, то $x^* = (a + b)/2$ – точка мінімуму, закінчують обчислення, інакше – крок 1.

Метод золотого перетину.

Точки x_1 та x_2 ділять інтервал $[a, b]$ за пропорцією *золотого перетину* (4.3)

$$x_1 = a_i + \frac{(3-\sqrt{5})}{2}(b_i - a_i),$$
$$x_2 = b_i - \frac{(3-\sqrt{5})}{2}(b_i - a_i). \qquad (4.3)$$

За одну ітерацію інтервал невизначеності зменшується приблизно в 1.618… разів. На наступній ітерації функцію обчислюють один раз. Кількість ітерацій $n \geq \frac{\ln((b-a)/\varepsilon)}{\ln((\sqrt{5}-1)/2)}$ забезпечує задану точність результату.

Метод чисел Фібоначчі.

Числа Фібоначчі визначають за допомогою рекурентних співвідношень: $F_{n+2} = F_{n+1} + F_n$, $n = 1,2,3\ldots$, $F_1 = F_2 = 1$.

За методом чисел Фібоначчі на початковому інтервалі $[a_0, b_0]$ обчислюють точки

$$x_1 = a_0 + \frac{F_{n-2}}{F_n}(b_0 - a_0),\ x_2 = a_0 + \frac{F_{n-1}}{F_n}(b_0 - a_0) \qquad (4.4)$$

де n обчислюють з $(b_0 - a_0)/\varepsilon < F_n$.

Для запобігання рекурсії[13] числа Фібоначчі розраховують за формулою Біне $F_n \cong \left[\left((1+\sqrt{5})/2\right)^n - \left((1-\sqrt{5})/2\right)^n\right]/\sqrt{5}$.

На k-му кроці методу точки x_1, x_2 розташовано симетрично відносно середини відрізка $[a_k, b_k]$

[13] З міркувань швидкості та передбачуваності процесу обчислень

$$x_1 = a_k + \frac{F_{n-k-1}}{F_{n-k+1}}(b-a), \qquad x_2 = a_k + \frac{F_{n-k}}{F_{n-k+1}}(b-a). \qquad (4.5)$$

Наприкінці процесу обчислень $k = n$, а точки x_1 та x_2 співпадають та ділять відрізок $[a_k, b_k]$ навпіл. Зі збільшенням n, з огляду на те, що відношення F_n/F_{n+2} є нескінченним десятковим дробом, симетрія інтервалів порушується, що призводить до додаткових похибок методу. Для методів золотого перетину і чисел Фібоначчі дотримуються тієї ж стратегії виключення інтервалів, що й в методі дихотомії.

Далі розглянемо як побудувати однотипний обчислювальний процес знаходження мінімуму функції на інтервалі з урахуванням співвідношень (4.1)-(4.5).

Реалізація алгоритмів пошуку мінімуму у *Octave*

Метою є ознайомлення із можливостями програмування подібних задач у *Octave* із використанням стандартних алгоритмічних конструкцій. Також розглянемо деякі універсальні функції пакету для перевірки наших програм.

З урахуванням того, що ми працюємо з математичними функціями, природнім є також використання функцій користувача в програмах, що розроблюються. На мові програмування *Octave* є можливість реалізації простих анонімних функцій та "повноцінних" функцій користувача у вигляді *m*-файлів. В попередніх темах ми вже бачили приклади використання функцій, тепер розглянемо це питання більш ретельно.

Анонімні функції користувача визначають за допомогою символу @:

@(список аргументів) вираз.

Анонімну функцію можна також присвоїти деякій змінній й використовувати звичним чином, як будь-яку іншу функцію.

Символ @ означає посилання на деяку стандартну або користувальницьку функцію, наприклад, $ref\_sin = @sin$, тепер при використанні змінної-посилання буде визвано оригінальну функцію, наприклад, $ref\_sin(0{,}5) \equiv \sin(x)$. Такі функціональні вказівники використовують також для адаптації функцій при використані деяких алгоритмів (докладніше див. довідку *Octave*).

Зазначимо, що вираз, який міститься в тілі анонімної функції має бути простим, тобто заборонено використовувати умови, цикли й т. п.

Приклад 4.1. Анонімні функції.

Задача – визначити анонімні функції користувача та перевірити роботу адаптерів функції для алгоритмів.

```
## 1 - функціональний адаптер
a = 1; b = 2; quad (@(x) betainc (x, a, b), 0, 0.4)
## 2 - анонімна функція
f = @(x) 2 - log(x) - x;
xx = linspace(1,10);
plot(xx, f(xx), '-k', 'linewidth', 3.25)
grid on
## 3 - анонімна функція 2-х змінних
xy = @(x,y) sin(x .^ 2+y .^ 2)
x=0.5; y=0.5;
printf("Значення функції xy(%2.2f,%2.2f) = %f\n", x, y,
xy(x,y))
```

Результат:

```
ans =  0.13867
xy = @(x, y) sin (x .^ 2 + y .^ 2)
Значення функції xy(0.50,0.50) = 0.479426
```

У першому випадку функція $quad()$ приймає на вході тільки функції однієї змінної, тому створюємо анонімну функцію, а параметри a та b є сталими та визначаються у області пам'яті основної програми. У другому – було створено анонімну функцію однієї змінної, у третьому – двох змінних.

Зазначимо, що з міркувань продуктивності краще використовувати вказівники для існуючих функцій *Octave*, а не визначати анонімні функції, які обгортають існуючу функцію[14].

Іменовані функції користувача, скрипти та файли-функції.

Для визначення функції у *Octave* використовують дві основні форми

[14] Функція $f = @(x)\ sin(x)$ буде обчислюватися довше ніж ref_sin

function name (*params_list*) тіло функції; *endfunction*	*function* [*return_value_list*] = *name* (*params_list*) тіло функції; *endfunction*

Примітка: *params_list* – список параметрів, що передаються у функцію, розділених комою; *return_value_list* – список імен змінних, розділених комою, в які будуть записуватися значення, що повертаються з даної функції. Цей список має містити щонайменше один елемент, а його особливістю є те, що значення, що повертаються *можуть мати різні типи та розмір*.

За необхідності можна отримати доступ до конкретного вихідного параметру ($nthargout(n, func)$)та контролювати кількість вхідних та вихідних параметрів. У Octave не потрібно вказувати тип параметрів. Тип параметрів може змінюватися в процесі виконання програми, тому визначені спеціальні функції групи предикатів ($isnumeric(\,)$, $isbool(\,)$, $iscomplex(\,)$, $ismatrix(\,)$ й т.п.) за допомогою яких можна перевірити тип параметру в процесі виконання функції. Допускається використання рекурсії, але з обмеженнями для деяких вбудованих функцій.

Важливо запам'ятати, що у *Octave* аргументи функції *передаються за значенням*, тобто кожен аргумент замінюється локальною копією перед тим, як передаватися функції. В даний час немає способу вказати, що параметр функції повинен передаватися як посилання, а це означає, що неможливо безпосередньо змінити параметр функції у функції, він може змінити лише локальну копію в тілі функції.

Якщо файл починається з визначення функції, то такий файл вважається файлом-функцією й на нього накладаються додаткові обмеження – ім'я файлу дорівнює імені функції та не повинне співпадати зі стандартними функціями *Octave*. Для того, щоб пакет сприймав файл як *скрипт*[15] – набір команд, у файлі перед першим визначенням функції, мають бути ще якісь вирази (наприклад, 1;).

Більш докладну інформацію про особливості використання функцій містить онлайн документація *Octave*[16].

[15] *Octave* та *MATLAB* мають різні обмеження на структуру скрипта

[16] https://octave.org/doc/v5.1.0/Functions-and-Scripts.html

Приклад 4.2. Визначення та робота із функціями.

Задача – написати функцію для обрахунку параметрів одновимірного масиву, а саме – середнього значення елементів.

```
## Приклад 4.2
## function example
clear all; clc
function  [a b c] = m_avg (vector)
  a = sum(vector) / length(vector);
  b = length(vector);
  c = vector(1);
endfunction
v = linspace (1,10,10)
a = m_avg(v)
```

Результат:

```
>> my_avg
v = 1    2    3    4    5    6    7    8    9   10
Виклик із одним вихідним параметром:
avg =  5.5000
Виклик із двома вихідними параметрами:
avg =  5.5000 len =  10
Виклик із трьома вихідними параметрами:
avg =  5.5000 len =  10 val =  1
```

Керування потоком виконання команд під час роботи програми здійснюють за допомогою стандартних для мов програмування алгоритмічних конструкцій та операторів (табл. 4.1). Приклади їх використання будуть наведені нижче, при реалізації алгоритмів пошуку мінімумів.

Таблиця 4.1

Оператори для керування потоком виконання

Конструкція	**Приклад**	**Коментар**
Умови та вибір		

if (умова)
  then-оператори
elseif (умова)
  elseif-оператори
else
  else-оператори
endif
	if (rem (x, 2) == 0)
	  printf ('парне\n');
	elseif (rem (x, 5) == 0)
	  printf ('кратне 5\n');
	else
	  printf ('непаране\n');
	endif
		гілки *elseif* та *else* можуть бути відсутніми.

Продовження табл. 4.1

Конструкція	Приклад	Коментар

switch (X)
  case мітка_1
  операторі;
  case мітка_2
  операторі;
  otherwise
  дія;
endswitch
	A = rem(x,2);
	switch (A)
	  case 0
	  printf (‘парне\n’);
	  otherwise
	  printf
	(‘непарне\n’);
	endswitch
		на відміну від умовного оператору, мітки можуть бути рядками

Цикли		

while (умова)
  операторі
endwhile
	fibo = ones (1, 20);
	i = 3;
	while (i <= 20)
	  fibo (i) = fibo (i-1)
	+ fibo (i-2);
	  i++;
	  display(fibo(i))
	endwhile
		цикл виконується поки умова вірна

do
  операторі
until (умова)
	fibo = ones (1, 20);
	i = 2;
	do
	  i++;
	  fibo (i) = fibo (i-1)
	+ fibo (i-2);
	  display(fibo(i))
	until (i == 20)
		цикл виконується поки умова не стане вірною

for змінна = вираз
  операторі
endfor
	fibo = ones (1, 20);
	for i = 3:20
	  fibo(i) = fibo(i-1) +
	fibo(i-2);
	  display(fibo(i))
	endfor
		у якості виразу використовують діапазони значень, списки значень. Є особливості, якщо змінна - вектор-стовпець, то вона буде вектором стовпцем на кожній ітерації, для інших (діапазон, вектор-рядок, скаляр) змінна буде скаляром.

Продовження табл. 4.1

Конструкція	Приклад	Коментар

`break`	`if (a == 0)`
	`  break;`
	`endif`
		перериває виконання циклу

`continue`	`if (a == 0)`
	`  continue;;`
	`endif`
		відразу переходить до наступної ітерації, пропускаючи всі оператори нижче умови

Приклад 4.3. Пошук мінімуму функції однієї змінної.

Задача – написати функцію, що реалізує *алгоритм дихотомії*, на вході функція повинна приймати ім'я функції, відрізок локалізації мінімум та бажану точність. За основу візьмемо перший алгоритм.

В цьому й наступних прикладах ми ставили за мету показати безпосередню реалізацію наведеного алгоритму, можливо, нехтуючи його ефективністю. Результат роботи програми відображено у вигляді графіка (рис. 4.2).

```
## Приклад 4.3
## реалізація алгоритму методу дихотомії
clc, clear all
function [ res, x, err ] = bisect(f, a, b, tol)
   while (b - a > tol)
    x0 = (a + b) / 2; x1 = (x0 + a) / 2;  x2 = (x0 + b) / 2;
    if (f(x1) < f(x0))
      b = x0; x0 = x1;
    else
      a = x1; b = x2;
    endif
   endwhile
   x = (a+b)/2; res = f(x); err = b - a;
endfunction
## перевірка
f = @(x)  x.^4 - 32 * x .^ 2 + x + 4;
[res, X, err ] = bisect(f, -8, 0, 1e-3)
```

Результат:

```
>> bisect_ex2_min
res = -256.0038986168802
x = -4.007812500000000
err =  9.765625000000000e-04
```

Приклад 4.4. Пошук мінімуму функції однієї змінної.

Задача – написати функцію, що реалізує алгоритм на основі *методу чисел Фібоначчі*, на вході функція повинна приймати ім'я функції, відрізок локалізації мінімум та бажану точність.

В прикладі використовується функція *fibo*(), для пошуку заданого числа Фібоначі, яку пропонується розробити читачеві самостійно.

```
function [y, x, err, iter] = fibonacci_opt(f, a, b, tol)
  n = 1;
  while ( (b - a) / tol >  fibo(n))
    n += 1;
 endwhile
  x1 = a + fibo(n-2) / fibo(n)*(b-a);
  x2 = a + fibo(n-1) / fibo(n)*(b-a);
  for k = 1 : n
    if (f(x1) < f(x2))
      b = x2; x2 = x1;
      x1 = a + fibo(n-k-1) / fibo(n-k+1)*(b - a);
    else
      a = x1; x1 = x2;
      x2 = a + fibo(n-k) / fibo(n-k+1)*(b - a);
    endif
  endfor
  x = (b + a) / 2;  y = f(x); iter = n; err = b - a;
endfunction
```

Результат:

```
>> fibo_main
y = -256.0038978583761
x = -4.007678818032224
err =    4.516995161409909e-04
iter =  21
```

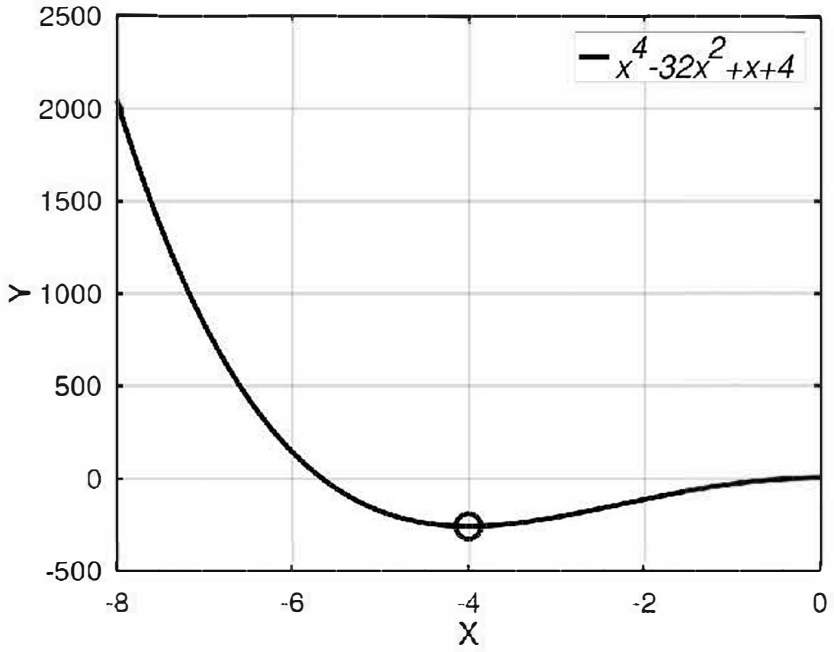


Рисунок 4.2

Функції мінімізації Octave реалізують основні алгоритми пошуку та характеризуються різною обчислювальною складністю. Нижче наведено перелік функцій, їх коротку характеристику, а також приклад 4.4, в якому показане їх застосування.

- [x, y, cvg, info] = *fminbnd* (fun, a, b, control) – пошук мінімуму унімодальної функції на заданому інтервалі методом золотого перетину і параболічної інтерполяції, *control* визначає додаткові параметри оптимізації.
- [x, y] = *fminsearch* (*fun*, *x*0, *control*) – метод безумовної оптимізації нульового порядку Нелдера-Міда (*Nelder-Mead*).
- [*x*, *fval*] = *fminunc* ('*myfun*', *x*0, *control*) – безумовна мінімізація функції. Для використання методу першого порядку задають опцію *optimset*('*GradObj*', '*on*') (див. приклад 4.2.), параметр *control* визначає додаткові параметри оптимізації;
- [x, y, info, iter, nf, lambda] = *sqp*(x0, fun) – повертає результат розв'язання задачі квадратичного програмування. У загальному випадку – це багатовимірна оптимізація з обмеженнями (g, h, lb, ub), які задають додатковими параметрами функції *sqp*( ).
- [*x*, *y*, *info*, *output*] = *fminbnd*(*f*, *a*, *b*) – пошук мінімуму унімодальної функції.

Приклад 4.5. Функції мінімізації *Octave*.

Задача – знайти мінімум функції на інтервалі за допомогою стандартних функцій *Octave*.

```
## оптимізація різними методами
display('Функція fminbnd')
[X, Y, INFO, OUTPUT]=fminbnd(f, -8, 0)
display('Функція fminsearch')
[X, Y]=fminsearch(f, -6)
display('Функція fminunc')
[X, Y, INFO, OUTPUT, GRAD, HESS] = fminunc(f, -6)
display('Функція sqp')
[X, Y, INFO, ITER]=sqp(-6, f)
```

Результат:

```
>> Функція fminbnd
X = -4.007790061006447
Y = -256.0038986502489
INFO =  1
OUTPUT = scalar structure containing the fields:
    iterations =  11
    funcCount =  12
    algorithm = golden section search, parabolic interpolation
    bracket =  -4.007823394339782  -4.007756727673112
>> Функція fminsearch
X = -4.007812500000000
Y = -256.0038986168802
>> Функція fminunc
X = -4.007789718579470
Y = -256.0038986502559
INFO =  3
OUTPUT = scalar structure containing the fields:
    iterations =  8
    successful =  7
    funcCount =  14
GRAD =   -1.903641419156349e-06
HESS =  128.7543429497616
>> Функція sqp
X = -4.007789753970393
Y = -256.0038986502558
INFO =  101
ITER =  7
```

Проблема пошуку глобального мінімуму

Приклад 4.6. Багатомодальні функції

Задача – на заданому інтервалі $[0;30]$ знайти мінімум функції $|2 \cdot (x-24) + (x-24) \cdot \sin(x-24)|$.

Основною проблемою цієї функції є її багатомодальність, тобто на заданому інтервалі вони мають декілька мінімумів, серед яких може бути глобальний мінімум, в чому можна переконатися побудувавши графік функції(рис. 4.3).

```
# Приклад 4.6, глобальний мінімум
y = @(x) abs(2*(x-24)+(x-24).*sin(x-24))
x = linspace(-20, 60, 160);
plot(x,y(x),'k','LineWidth', 3.25)
title('Функція із багатьма локальними мінімумами')
legend('|2{\cdot}(x-24)+(x-24){\cdot}sin(x-24)|')
grid on; print('theme_2_global.png', '-r600', '-dpng');
```

Рисунок 4.3

Методи, які ми вивчили в даній темі, *не можна застосовувати до таких функцій*, в чому ми пропонуємо переконатись читачеві самостійно.

Також цей приклад підтверджує тезу про те, що роботу із функціями, заданим аналітично або таблицями своїх значень, потрібно починати з побудови їх графіка.

Резюме

В даній темі ми вивчили методи оптимізації нульового порядку, навчилися локалізувати інтервал пошуку мінімуму унімодальної функції та на прикладі реалізації методів дихотомії(бісекції), золотого перетину та чисел Фібоначчі засвоїли роботу із функціями користувача та операторами керування потоком виконання програми *Octave*. Розібрали на прикладах використання стандартних функцій мінімізації *Octave*, побачили, що розроблені нами функції, дають приблизно такий же результат обчислень, що й стандартні, що є непрямим підтвердженням правильності реалізації алгоритмів.

В темах 1-4 ми присвятили багато часу основам програмування на мові *Octave*. Тепер ви маєте уяву про особливості цієї мови програмування та вмієте реалізовувати алгоритми для розв'язання різних задач. Надалі ми більше зосередимося на дослідженні методів та алгоритмів, при цьому не забуваючи про нові можливості мови *Octave*.

Функції та оператори *Octave*, що вивчаються: $if..else..endif$, $while..endwhile$, $do..until$, $for..endfor$, $fminbnd()$, $fminunc()$, $fminsearch()$, $sqp()$, $plot()$, анонімні функції, функціональні вказівники, звичайні функції користувача.

Питання для самоперевірки

1. Сформулюйте задачі одновимірної оптимізації, які виникають в процесі моделювання процесів та пристроїв.
2. Дайте визначення унімодальної функції на інтервалі.
3. Опишіть загальний алгоритм зменшення інтервалу невизначеності.
4. Опишіть метод дихотомії, доведіть правильність реалізації алгоритму в прикладі 4.3.
5. Опишіть метод золотого перетину, доведіть правильність реалізації алгоритму в прикладі 4.4.
6. Опишіть метод чисел Фібоначчі, доведіть правильність реалізації алгоритму в прикладі 4.5.
7. Назвіть характерні недоліки методів нульового порядку?

Практичне завдання

Дослідити функцію на заданому інтервалі, знайти мінімум функції згідно з варіантом: а) за допомогою програми на мові *Octave*; б) за допомогою вбудованих функцій.

Порядок виконання завдання

1. Для заданого варіанту визначити функцію користувача та побудувати її графік засобами *Octave*.

2. Визначити інтервал локалізації мінімуму.

3. Уточнити значення мінімуму за допомогою заданого методу. Додати на графік проміжні точки мінімуму і границі інтервалів для перших 5 кроків алгоритму.

4. Порівняти результати за п. п. 3–4 з аналогічними результатами, отриманими за допомогою вбудованих функцій *Octave*.

5. Повторити п. п. 1-4, розширивши інтервал пошуку мінімуму вдвічі, симетрично в обидві сторони. Прокоментувати результати.

Індивідуальні завдання

1. $f(x) = x^4 - 48x^2 + 22x + 5,\ x \in [-8; 0]$.
2. $f(x) = -x^4 + 48x^2 - 22x + 1,\ x \in [-5; 5]$.
3. $f(x) = x^5 + 10x^2 - 10x + 5,\ x \in [-1{,}5; 2]$.
4. $f(x) = 0{,}75x^3 + 2{,}5x^2 - 8{,}25x + 0{,}5, x \in [-2{,}5; 7]$.
5. $f(x) = 0{,}25x^3 - 5x^2 - 8{,}25x + 3,\ x \in [-3; 25]$.
6. $f(x) = 3x^4 - 0{,}8x^3 - 1{,}2x - 15x, x \in [-4; 4]$.
7. $f(x) = 0{,}01x^6 + 2x^4 - 55x^2,\ x \in [-7{,}5; -2]$.
8. $f(x) = x^3 - 5x^2 + x,\ x \in [1; 5]$.
9. $f(x) = -x^3 + 7x^2 + x,\ x \in [-6; 3]$.
10. $f(x) = x^5 + 25x^4 + 7x^3,\ x \in [-11; 10]$.
11. $f(x) = x^2 \ln 0{,}25x + 5,\ x \in [1; 6]$.
12. $f(x) = \sin 0{,}5x^2 \log 0{,}5x,\ x \in [2{,}5; 3.6]$.
13. $f(x) = \operatorname{arctg} x \cos 0{,}25x,\ x \in [-7; 1]$.
14. $f(x) = e^x \sin\cos 0{,}75x,\ x \in [-5; 0]$.
15. $f(x) = 3x \sin e^{x/3},\ x \in [-10; 2]$.
16. $f(x) = 0{,}5/x - x^2 + 5x,\ x \in [0; 2]$.

ТЕМА 5
ПРЯМІ МЕТОДИ ПОШУКУ МІНІМУМУ ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ. МЕТОДИ ТОЧКОВОГО ОЦІНЮВАННЯ

Метою є ознайомлення студентів з методами точкового оцінювання для пошуку мінімуму однієї змінної засобами *Octave*.

Постановка задачі. Унімодальна цільова функція $y = f(x)$ є функцією однієї змінної x та має мінімум на заданому інтервалі. Визначити мінімум цільової функції із заданою похибкою ε.

Методи точкового оцінювання. Розглянуті в попередній темі методи дозволяють знайти малий інтервал ($\Delta x \leq \varepsilon$), в якому знаходиться мінімум функції, властивості якої взагалі не враховують. За іншим підходом використовують кілька значень функції в певних точках для її апроксимації звичайним поліномом в обмеженій області. Основна ідея методу – можливість апроксимації гладкої функції та оцінювання точки екстремуму поліномом високого степеня. Якість цієї оцінки можна покращити двома способами – *збільшенням степеня* полінома або *зменшенням інтервалу* апроксимації. При цьому, характер поведінки функції, яку мінімізують, враховується при виборі вигляду (степеня) полінома.

Метод квадратичної апроксимації (*метод Пауелла*) базується на припущенні, що в обмеженому околі значень x функцію $f(x)$ можна апроксимувати квадратичним поліномом.

Якщо для заданої множини точок $X = \{x_1, x_2, x_3\}$ відомі значення функції в цих точках $f(x_1) = f_1$, $f(x_2) = f_2$, $f(x_3) = f_3$, тоді коефіцієнти полінома a_0, a_1 та a_2 цих точках визначають із рівності

$$\varphi(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2). \qquad (5.1)$$

Напишемо розв'язок (5.1) у вигляді (5.2)

$$a_0 = f_1, a_1 = \frac{f_2 - f_1}{x_2 - x_1}, a_2 = \frac{1}{x_3 - x_2}\left(\frac{f_3 - f_1}{x_3 - x_1} - \frac{f_2 - f_1}{x_2 - x_1}\right). \qquad (5.2)$$

Точність апроксимації з допомогою квадратичного полінома є

досить високою, тому враховуючи унімодальність функції $f(x)$ та застосувавши необхідну умову існування екстремуму $\partial\varphi(x)/\partial x = 0$ до (5.1), отримують оцінку мінімуму

$$x^* = \frac{x_2 + x_1}{2a_2} - \frac{a_1}{2a_2}. \tag{5.3}$$

Під час реалізації алгоритму будемо обчислювати

$$\tilde{x}^* = \frac{A + B + C}{2(A_1 + B_1 + C_1)}, \tag{5.4}$$

де

$$A = (x_2^2 - x_3^2)f(x_1),\ A_1 = (x_2 - x_3)f(x_1),$$
$$B = (x_3^2 - x_1^2)f(x_2),\ B_1 = (x_3 - x_1)f(x_2),$$
$$C = (x_1^2 - x_2^2)f(x_2),\ C_1 = (x_1 - x_2)f(x_3).$$

Алгоритм методу Пауелла заснований на послідовному наближенні до мінімуму за допомогою (5.1)–(5.3):

Крок 1. Задають x_1, похибку $\varepsilon > 0$ та крок $h > 0$.

Крок 2. $x_2 = x_1 + h$.

Крок 3. Якщо $f(x_1) > f(x_2)$, то $x_3 = x_1 + 2h$,
інакше $x_3 = x_1 - h$.

Крок 4. Обчислюють значення $F_{min} = \min\{f(x_1), f(x_2), f(x_3)\}$, яке відповідає проміжній точці мінімуму x_{min}

Крок 5. За формулою (5.4) визначають $\tilde{x}^*$, Якщо $|\tilde{x}^* - x_{min}| < \varepsilon$, то пошук закінчують, інакше переходять до кроку 6.

Крок 6. Якщо $\tilde{x}^* \in [\min\{X\}, \max\{X\}]$, то найбільше значення аргументу в даній точці $x = \arg\max_{x \in \{x_1, x_2, x_3\}} f(x)$ заміняють на $\tilde{x}^*$; якщо $\tilde{x}^* \notin [\min\{X\}, \max\{X\}]$, то приймають $x_1 = \tilde{x}^*$. Після зроблених замін перевіряють, що не було втрачено інтервал з поточною точкою мінімуму $x_1 < x_2 < x_3$, $f(x_1) < f(x_2) < f(x_3)$ та повертаються на крок 4. Кроки 4-6 є основними для методу Пауелла.

Метод квадратичної апроксимації зазвичай застосовують після локалізації точки мінімуму одним з методів, розглянутих у темі 4, так як для функції, що двічі диференціюється, поліном другого порядку достатньо добре апроксимує функцію в околі точки мінімуму.

Алгоритм інтерполяційного методу Девіса – Свена – Кемпі знаходження мінімуму функції на інтервалі:

Задають довільну точку x_0, похибку $\varepsilon > 0$ та крок $h > 0$.

Крок 1. Встановлюють напрямок убування цільової функції. Для цього беруть $x_1 = x_0 + h$. Якщо $f(x_1) \geq f(x_0)$, то $h = -h$, $x_1 = x_0 + h$, $h = 2h$.

Крок 2. Повторюють:

- $x_3 = x_1 + h$;
- якщо $f(x_3) \leq f(x_1)$, то $h = 2h$, $x_0 = x_1$, $x_1 = x_3$, інакше $h = h/2$, $x_2 = x_1 + h$ та переходять на крок 3.

Крок 3. Вилучають найвіддаленішу від мінімуму точку, а з решти трьох b є центральною точкою, $a = b - h$, $c = b + h$.

Крок 4. Проводять квадратичну інтерполяцію для визначення точки мінімуму

$$x^* = b + \frac{h}{2} \cdot \frac{f(a) - f(c)}{f(a) - 2f(b) + f(c)}. \qquad (5.5)$$

Якщо $f(c) < f(x^*)$, то $x_0 = c$, $h = h/2$ і повертаються на крок 1, інакше $x_0 = x^*$ та переходять на крок 1. Обчислення повторюють до тих пір, поки різниця двох послідовних оцінок $|x^*_{k-1} - x_k{}^*| > \varepsilon$.

Приклад 5.1. Метод оптимізації Пауелла.

Задача – розробити функцію користувача, що реалізує метод Пауелла.

```
## Приклад 5.1
clc, clear all
function [y, xa, err, iter] = powell_opt(f, l, r, tol)
  iter = 0; h = 0.5; x = zeros(1, 3);
  x(1) = (r + l) / 1.25;
  x(2) = x(1) + h;
  if (f(x(1)) > f(x(2)))
    x(3) = x(1) + 2 * h;
  else
    x(3) = x(1) - h;
  endif
  do
    iter += 1;
```

```
    A = (x(2)^2 - x(3) ^ 2)*f(x(1)); B = (x(3)^2-
x(1)^2)*f(x(2));
    C = (x(1)^2 - x(2) ^ 2)*f(x(3));
    A1 = (x(2) - x(3)) * f(x(1)); B1 = (x(3) - x(1)) *
f(x(2));
    C1 = (x(1) - x(2)) *(x(3));
    xa = (A + B + C) / (2 * (A1 + B1 + C1));
    x = sort(x);
    if (abs(f(xa) - f(x(1))) < tol)
      y = f(x(1)); err = [ abs(f(x(1)) - f(xa)), abs(x(1) -
xa)];
      break;
    endif
    if (xa <= x(3))
      x(3) = xa;  x = sort(x);
    elseif (xa >= x(1))
      x(1) = xa; x = sort(x);
    else
      x(1) = xa;
    endif
  until (iter > int32(1 / tol))
endfunction
```

Результат:

```
f = @(x) x.^4 - 32 * x .^ 2 + x + 4;
[y, x, e, c] = powell_opt(f, -8, 0, 1e-3)
>>
y = -256.0029740617852
x = -4.007789899710401
e = 9.245884689335071e-04   3.787859424764228e-03
c =  10
```

Функції оптимізації з пакету *optim* (*Octave*)

В попередній темі ми познайомились зі стандартними функціями оптимізації *Octave*, що включені в ядро (*core*) пакету. Додатково є окремий пакет *optim*, в якому реалізовані популярні алгоритми, нижче наведено деякі з них. Зазначимо, що деякі з наведених методів можуть використовуватися для мінімізації функції декількох змінних та необов'язково належать до методів точкового оцінювання.

• [x, y, cvg, iters, nevs] = *powell* (fun, x0, control) – метод Пауелла (багатовимірний випадок), Структура *control* приймає "*TolX*", "*MaxFunEvals*", "*MaxIter*", "*SearchDirections*".

• $[x, y, cvg, info] = nonlin\_min\,(fun, x0, control)$ – інтерфейс до методів $lm\_feasible\,(Levenberg - Marquardt)$, $octave\_sqp$, $siman$, $d2\_min$.

• $[x, v, nev, h, args] = d2\_min(fun, d2f, args, control, code)$ – відноситься до групи ньютонівських методів, метод функції $nonlin\_min$.

• $[x, y, cvg, iters] = bfgsmin(fun, args, control)$ – метод Брой-дена-Флетчера-Гольдфарба-Шано (*BFGS*).

• $x = nrm\,(f, x0)$ – метод Ньютона-Рафсона.

Приклад 5.2. Мінімізація за методом Пауелла.

Задача – визначити точку мінімум за допомогою вбудованої функції $powell()$.

```
## Приклад 5.2
clc, clear all
f=@(x) -sin(x) % функція
X=0:0.1:3;
plot(X, myf(X))
hold on
x=1.0; % початкова точка
o = optimset('MaxIter', 100, 'TolFun', 1E-10); % параметри
[x,y,convergence,iters,nevs] = powell (f,x,o)
plot(x, y, 'dk')
```

Результат:

```
>>
myf = @(x) -sin (x)
x =  1.570796326794911
y = -1
convergence =  1
iters =  2
nevs =  32
(arg: 0)
```

Резюме

В даній темі розібрали роботу із методами оптимізації на основі точкового оцінювання, дослідили на прикладі реалізацію метода Пауелла, що дозволяє достатньо швидко знаходити мінімум функції. Продовжили вивчати функції *Octave*: *powell*( ), *minimize*() , *nonlin_min*(), *plot*() та ін. Більш докладно із особливостями методів цих методів оптимізації, ми познайомимся у наступних темах.

Питання для самоперевірки

1. У яких випадках застосовують методи точкового оцінювання?
2. Отримайте розрахункові співвідношення (5.2) і (5.3).
3. Чому бажано визначити початковий інтервал локалізації мінімуму?
4. Опишіть метод оптимізації Пауелла. Які недоліки та переваги цього методу ви можете назвати
5. Опишіть метод оптимізації Девіса-Свена-Кемпі.

Практичне завдання

Знайти мінімум функції згідно з варіантом: а) за допомогою розробленої програми на мові *Octave*; б) за допомогою вбудованих функцій.

Порядок виконання завдання

1. Для заданої функції (див. тему 4) визначити функцію користувача та побудувати її графік засобами *Octave*.
2. Реалізувати обидва методи точкового оцінювання, пояснити особливість роботи обох алгоритмів. Скласти порівняльну таблицю вивчених методів оптимізації, вибрати критерії для оцінки та оцінити результати для розроблених функцій у темах 4-5.
3. Показати на графіку перші кроки роботи алгоритму за допомогою *subplot*() із 6 графіками.
4. Порівняти результат за п. 3 з аналогічними результатами, отриманими за допомогою вбудованих функцій *Octave*.
5. Дослідити роботу функцій мінімізації із пакету *optim*.

ТЕМА 6
МЕТОДИ ПЕРШОГО ТА ДРУГОГО ПОРЯДКІВ ДЛЯ ПОШУКУ МІНІМУМУ ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ

Метою є ознайомлення студентів з методами першого та другого порядків для пошуку мінімуму функції однієї змінної засобами *Octave*.

Постановка задачі. Нехай цільова функція $y = f(x)$ є унімодальною, безперервною та такою, що диференціюється (або двічі диференціюється) на інтервалі пошуку мінімуму. Визначити мінімум цільової функції із заданою похибкою ε.

Методи оптимізації, що вивчаються в даній темі використовують інформацію про першу (першого порядку) та другу (другого порядку) похідні функції. Пригадаємо із курсу математичного аналізу необхідні та достатні умови існування екстремуму.

Необхідна умова існування екстремуму: якщо функція $f(x)$ визначена в околі точки x_0 за виключенням може самої точки x_0, має в точці x_0 екстремум, то в цій точці $f'(x) = 0$ або $f'(x) = \infty$.

Достатні умови екстремуму функції однієї змінної:

Якщо функція $f(x)$ неперервна в т. x_0 і деякому δ-околі цієї точки, має похідну, крім, може, в самій т. x_0 , тоді, якщо $f'(x)$ при переході через т. x_0

- змінює знак з + на - , то x_0 є точкою максимуму;
- змінює знак з - на + , то x_0 є точкою мінімуму;
- знак не змінює, то x_0 не є точкою екстремуму.

Якщо функція $f(x)$ визначена і двічі диференційована в деякому околі т. x_0, причому $f'(x) = 0$, а $f''(x_0) \neq 0$, то в т. x_0 функція має (рис. 6.1)

- максимум, якщо $f''(x_0) < 0$;
- мінімум, якщо $f''(x_0) > 0$.

Таким чином, методи оптимізації

- нульового порядку не використовують похідну;
- першого порядку використовують першу похідну;
- другого порядку використовую другу похідну.

Методи чисельного диференціювання основані на використанні визначення похідної та апроксимаціях функції поліномами в шуканій точці

$$f'(x_0) = \lim_{\Delta x_0 \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}. \qquad (6.1)$$

Поширеним є спосіб апроксимації функції поліномами Ньютона або Лагранжа (R – похибка (залишковий член))

$$\begin{aligned} f^{(k)}(x) &\approx P^{(k)}(x), 0 \le k \le N, \\ f^{(k)}(x) &= P^{(k)}(x) + R, 0 \le k \le N. \end{aligned} \qquad (6.2)$$

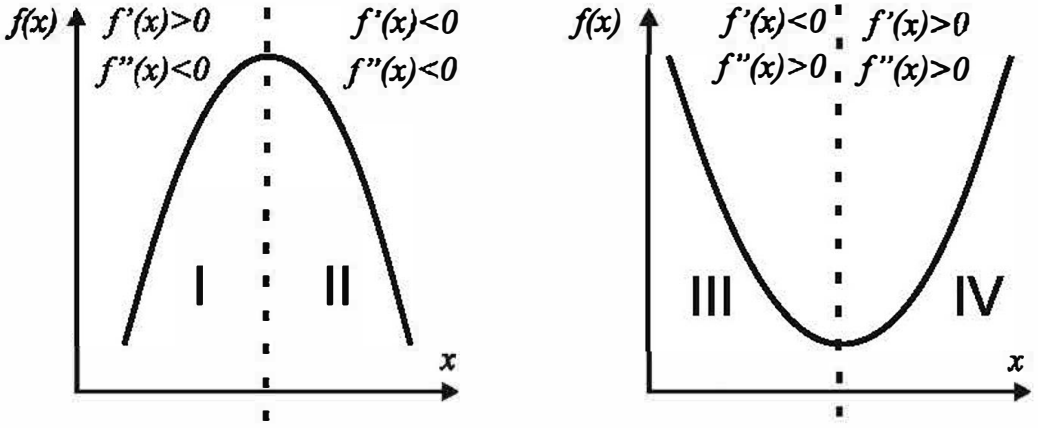


Рисунок 6.1

Таблиця 6.1

Формули для обчислення похідних першого порядку

Тип	Формула
Несиметрична обернена ($R(h)$)	$f'(x_i) = \frac{f_i - f_{i-1}}{h}$
Несиметрична обернена ($R(h^2)$)	$f'(x_i) = \frac{3f_i - 4f_{i-1} + f_{i-2}}{2h}$
Несиметрична пряма ($R(h)$)	$f'(x_i) = \frac{f_{i+1} - f_i}{h}$
Несиметрична пряма ($R(h^2)$)	$f'(x_i) = \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2h}$
Симетрична ($R(h^2)$)	$f'(x_i) = \frac{f_{i+1} - f_{i-1}}{2h}$
Симетрична ($R(h^4)$)	$f'(x_i) = \frac{-f_{i+2} + 8f_{i+1} - 8f_{i+1} + f_{i-2}}{12h}$

На практиці використовують отримані із (6.1)-(6.2) прямі (вперед), обернені (назад) та симетричні оцінки похідних (табл. 6.1-

6.2), симетричні схеми мають меншу похибку у порівнянні із несиметричними.

Таблиця 6.2

Формули для обчислення похідних другого порядку

Тип	Формула
Несиметрична обернена $(R(h))$	$f''(x_i) = \frac{f_i - 2f_{i-1} + f_{i-2}}{h^2}$
Несиметрична обернена $(R(h^2))$	$f''(x_i) = \frac{2f_i - 5f_{i-1} + 4f_{i-2} - f_{i-3}}{h^2}$
Несиметрична пряма $(R(h))$	$f''(x_i) = \frac{f_{i+2} + 2f_{i+1} - f_i}{h^2}$
Несиметрична пряма $(R(h^2))$	$f''(x_i) = \frac{2f_i - 5f_{i+1} + 4f_{i+2} - f_{i+3}}{h^2}$
Симетрична $(O(h^4))$	$f''(x_i) = \frac{f_{i+2} + 2f_i - f_{i+1}}{2h}$
Симетрична $(O(h^4))$	$f'(x_i) = \frac{-f_{i+2} + 16f_{i+1} - 30f_i + 16f_{i+1} + f_{i-2}}{12h^2}$

З практичної точки зору, під час обчислень кожну точку можна вважати за x_0, тому всі формули можна переписати для цієї точки (6.3), але за виключенням першої та останньої, в яких використовують відповідні формули вперед та назад.

$$
\begin{aligned}
f'(x_0) &= \frac{1}{2h}(-3f_0 + 4f_1 - f_2) + \frac{1}{3}h^2 f'', \\
f'(x_1) &= \frac{1}{2h}(f_2 - f_0) - \frac{1}{6}h^2 f'', \\
f'(x_2) &= \frac{1}{2h}(f_0 - 4f_1 + 3f_2) + \frac{1}{3}h^2 f''.
\end{aligned}
\qquad (6.3)
$$

Зазначимо, що існують інші способи побудови формул (на основі поліномів Лагранжа та Стірлінга) та різні схеми для чисельного знаходження похідної на рівномірних та нерівномірні сітках, при цьому основною проблемою є те, що зі зменшенням кроку h похибка диференціювання може суттєво зростати, що потрібно завжди мати на увазі.

Приклад 6.1. Знаходження похідної в чисельному вигляді.

Задача – написати програму знаходження похідної функції $\sin x$ на 3-х вузлах

На рис. 6.2а наведено графік функції та її похідної, а на 6.2б – похибка обчислень як різниця між точним та обчисленим значеннями похідної.

```
## Приклад 6.1
## Чисельне диференціювання
clear all; clc;
##pkg load symbolic;
## диференціювання по 3-х точках
function [dy] = diff_3p (y, h)
## формули
N = length(y);
if (N < 3)
  error("Помилка! Розмір масиві менше 3-х елементів")
  return
endif
direct = @(y, i) -3*y(i)+4*y(i+1)-y(i+2);
back   = @(y, i)  3*y(i)-4*y(i-1)+y(i-2);
## основний цикл
dy = zeros(1, N);
for i = 1 : N
  if (i < N-2)
    dy(i) = direct(y,i);
  else
    dy(i) = back(y,i);
  endif
endfor
dy /= 2 * h
endfunction
```

Результат:

```
## перевірка
f  = @(x) sin(x); ## функція
df = @(x) cos(x); ## похідна
N = 25; a = -2*pi/2; b = 2*pi/2; ## діапазон
x = linspace (a, b, N); h = (b-a) / N; ## крок
y = f(x); y1 = df(x); ## значення функції та точної похідної
dy = diff_3p(y, h); R = (y1 - dy); ## чис. похідна та помилка
```

Симетричні формули для оцінок других похідних використовують під час розв'язання крайових задач для звичайних диференціальних рівнянь і диференціальних рівнянь із частинними похідними. Несиметричні обернені формули застосовують для апроксимації перших похідних у граничних умовах для крайових задач і розв'язання жорстких звичайних диференціальних рівнянь із початковими умовами й т. п.

У *Octave* є декілька функцій для обчислення похідної чисельними методами та в аналітичному вигляді:

- $dx = deriv\ (f, x0, h, O, N)$ числове диференціювання (*optim*), f – ім'я функції (або вказівник), h – крок, x_0 – точка, $O = 2(4)$, $N = 1 \ldots 4$ – порядок похідної.
- $dx = diff(x)$ – обчислення кінцевих різниць. Якщо x – одновимірний масив вигляду, то $diff(x)$ – вектор різниць сусідніх елементів. Апроксимацією похідної n-го порядку є відношення $diff(y, n)./diff(x, n)$.
- $df = diff(f(x, y, \ldots.), n)$ – символьне диференціювання (*symbolic*).
- $[fx, fy, fz, \ldots] = gradient(f, h)$ – обчислення градієнта. Для обчислення похідної в точці x_0: $gradient(@f, x_0, h)$, де h – точність.

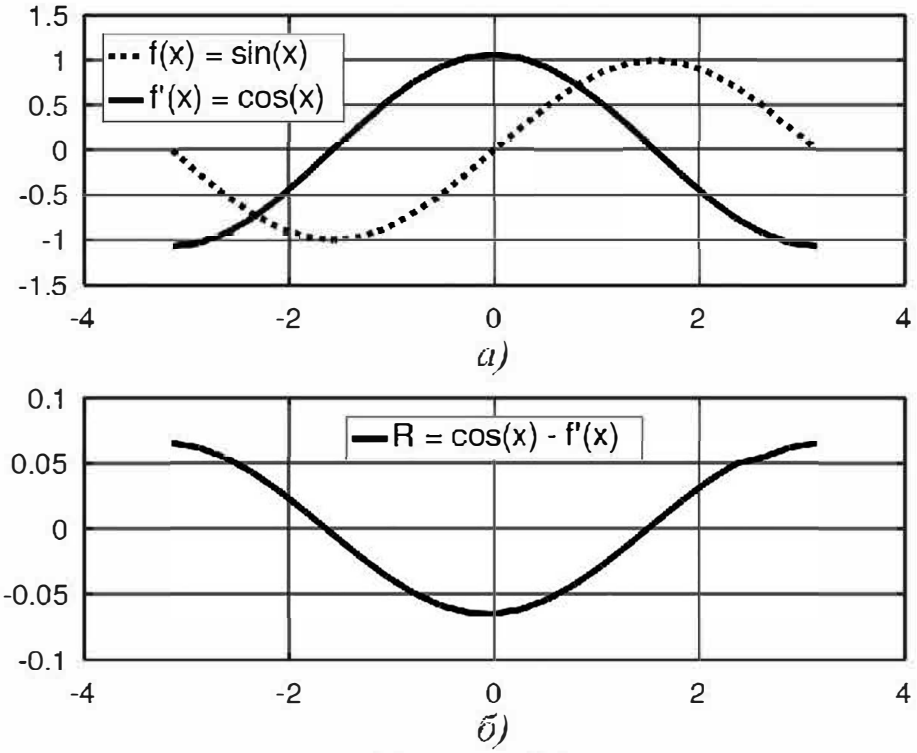


Рисунок 6.2

Приклад 6.2. Обчислення похідної в точці

Задача – написати програму обчислення похідної у символьному та в числовому вигляді.

Зверніть увагу на суттєві похибки чисельних функцій у випадку використання стандартних налаштувань.

```
## Приклад 6.2
pkg load symbolic; pkg load optim;
format long;
function f = myf(x)
  f=sin(x); % похідна -> cos(x)
endfunction
syms s;
dx=diff(sin(s),s)       % похідна в символьному вигляді
vpa(subs(dx, s, 1))     % розраховане значення
% розраховані значення у числовому вигляді
deriv(@myf,1, 1e-10)
gradient(@myf, 1, 0.00001)
gradient(@sin, 1)
```

Результат:

```
>> theme_6_ex2
dx = (sym) cos(s)
ans = (sym) 0.54030230586813971740093660744298
ans =    5.403022473871033e-01
ans =    5.403023058569989e-01
ans =    4.546487134128409e-01
```

Приклад 6.3. Обчислення похідної в точці.

Задача – написати програму обчислення похідної із використанням функцій *deriv*() та *gradient*(). Порівняти результати.

Результат представлено у графічному вигляді (рис. 6.3), зверніть увагу на поводження функції *derive*() на границі інтервалу.

```
## Приклад 6.3
clc;
pkg load optim;
function [dy] = f2(x) dy = sin(x); endfunction
f=@(x) sin(x);
a=-pi; b=pi; h=0.1; x=a:h:b;
```

```
dy = cos(x);
df1=gradient(f(x), h);      % похідна
df2=deriv(@f2, x, h);
subplot(2,1,1)
hf1 = plot(x,f(x),":", x, df1)
hl = legend("f(x) = sin(x)", "f'(x) = cos(x)");
hx = xlabel('a)')
set(hl, 'fontsize', 18, 'location','northwest')
set(gca, 'linewidth', 1.75, 'fontsize', 16, 'gridalpha',0.75);
set(hf1, 'linewidth', 3.5, 'color', 'black');
set(hx, 'fontsize', 22, 'fontangle', 'Italic', 'fontname',
'Times New Roman');
grid on
subplot(2,1,2)
err = ((df1-df2) ./ df1)*100;
hf2= plot (x, err)
hl = legend("Відносна похибка,{\\Delta}df/df1,%");
hx = xlabel('6)')
set(hl, 'fontsize', 18, 'location','southwest')
set(gca, 'linewidth', 1.75, 'fontsize', 16, 'gridalpha',0.75);
set(hf2, 'linewidth', 3.5, 'color', 'black');
set(hx, 'fontsize', 22, 'fontangle', 'Italic', 'fontname',
'Times New Roman');
grid on
```

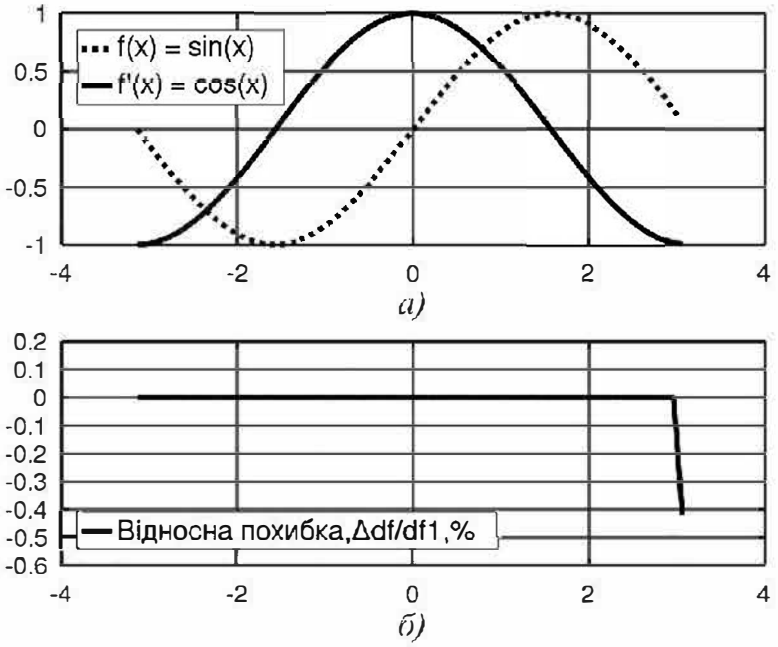


Рисунок 6.3

Перейдемо до розгляду *методів оптимізації першого та другого порядків*. Методам нульового порядку властиво накопичувати помилки, що призводить до виникнення додаткових похибок. Якщо функція на всьому інтервалі $[a, b]$ є унімодальною та безперервно диференційованою, то точка мінімуму x^* є коренем рівняння $f'(x) = 0$. Для відшукання x^* використовують методи розв'язання нелінійних рівнянь, а найбільш критичною операцією, що призводить до додаткових похибок та грубих помилок, є числове диференціювання.

Метод середньої точки побудований аналогічно прямим методам виключення інтервалу, тільки на кожній ітерації перевіряють виконання умов $f'(x) = 0$ та (або) $f'(x) < \varepsilon$.

Крок 1. Задають $\varepsilon > 0$ та дві точки a_1 і b_1 такі, що $f'(a_1) < 0$ та $f'(b_1) > 0$. Приймають $x_1 = (a_1 + b_1)/2$, $k = 1$.

Крок 2. Якщо $f'(x_k) = 0$ або $f'(x_k) < \varepsilon$, то $x^* = (a_k + b_k)/2$ є шуканою точкою мінімуму, обчислення закінчують.

Крок 3. Якщо $f'(x_k) < 0$, то $a_{k+1} = x_k$, $b_{k+1} = b_k$, інакше $a_{k+1} = a_k$, $b_{k+1} = x_k$, $x_k = (a_k + b_k)/2$ та повертаються на крок 2.

Даний метод швидше за прямі збігається до значення x^*.

Приклад 6.4. Метод середньої точки (рис. 6.4)

Задача – написати програму в якій реалізувати алгоритм методу середньої точки.

```
## Приклад 6.4
clear all;
format long g %  15 знаків після крапки
function [y] = f(x)
  y = - sin(x);
endfunction
a = 0; b = 2; h = 0.25;
x=a-h:h:b+h;   % дані для побудови графіку
plot(x, @f(x), ':k', 'linewidth', 2.75);
hold on; grid on;
eps=0.001; k=1;
a0 = a; b0 = b;
do
  ya=deriv(@f, a, 1);
  yb=deriv(@f, b, 1);
  plot(a0, ya,"kv", 'markersize',14,'linewidth', 1.5,
       b0, yb,"kd", 'markersize',14, 'linewidth', 1.5 )
```

```
  x1=(b+a)/2.
  df1=deriv(@f, x1, 1)
  plot(x1, df1,  "k.", 'markersize',36,
       a, @f(a), "k>", 'markersize',16,'linewidth', 1.5,
       b, @f(b), "k<", 'markersize',16,'linewidth', 1.5)
  if (df1 < 0)
    a=x1;
  else
    b=x1;
  endif;
  k++;
until (abs(df1)<eps)
hl = legend('-sin(x)','y_{a}', 'y_{b}',
"f'(x_{1})",'\itf(a)','\itf(b)');
set(hl, 'fontsize', 20, 'location', 'eastoutside')
set(gca, 'linewidth', 1.75, 'fontsize', 20, 'gridalpha',
0.65);
hold off
```

Результат:

```
>>
x1 = 1.5
dx1 = -0.05952330274987677
x1 = 2.25
dx1 = 0.5285898769428473
x1 = 1.875
dx1 = 0.2520487544363031
x1 = 1.6875
dx1 = 0.09797999045831812
x1 = 1.59375
dx1 = 0.01931315396878247
x1 = 1.546875
dx1 = -0.02012718272531056
x1 = 1.5703125
dx1 = -0.0004071261936942072
```

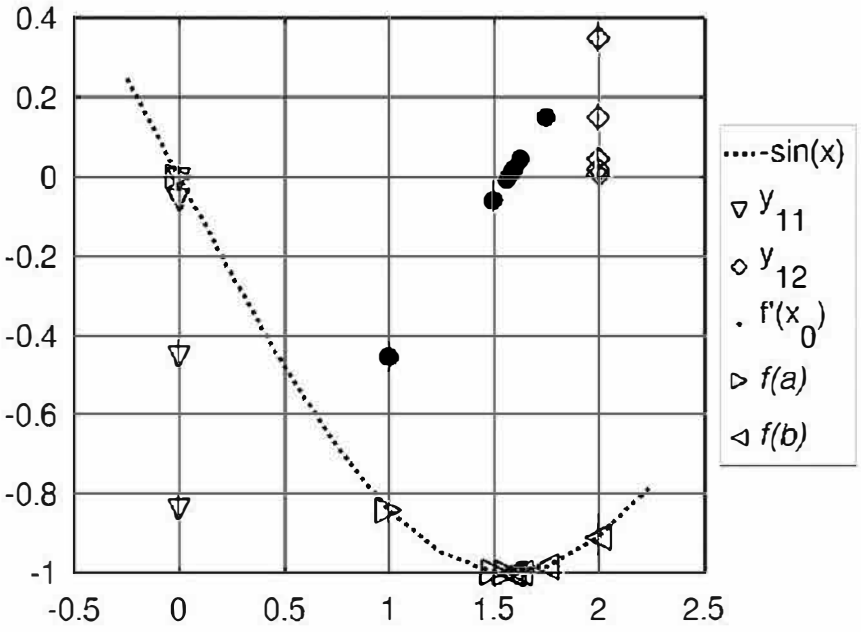


Рисунок 6.4

Приклад 6.5. Особливості роботи із функціями чисельного диференціювання.

Задача – дослідити вплив на результати обчислень додаткових параметрів функцій чисельного диференціювання.

Результат, що отримано, показує необхідність в додаткових налаштуваннях бібліотечних функцій та тестуванні програмного коду. Перші два результати значно відрізняються від таких же, отриманих методами нульового порядку, це сталось завдяки неправильному налаштуванню функції *Octave*. Висновок, або використовуйте стандартні методи, або уважно читайте опис функції у довідковій системі *Octave*. Цікавий також третій результат, який отримано всього за 2 кроки роботи алгоритму, а похибка з'явилась тільки в третьому розряді результату.

```
## Приклад 6.5
format long g %  15 знаків після крапки
function [y] = f(x)
  y = x.^4 - 32 * x .^ 2 + x + 4;
endfunction
function [x, iter] = mid_point_deriv(f, a, b, h, eps)
  iter = 1;
  do
    ya=deriv(@f, a, h);
    yb=deriv(@f, b, h);
```

```
    x=(b+a)/2.;
    df1=deriv(@f, x, h) ;
    if(df1 < 0) a=x; else b=x; endif;
    ++iter;
  until (abs(df1) < eps || iter > 1e3)
endfunction
a = -8; b = 0; eps=1e-3; h = 1;
[x, iter] = mid_point_deriv(@f, a, b, h, eps);
display([iter, x, f(x)])
h = 0.5; [x, iter] = mid_point_deriv(@f, a, b, h, eps);
display([iter, x, f(x)])
h = 0.25; [x, iter] = mid_point_deriv(@f, a, b, h, eps);
display([iter, x, f(x)])
h = 0.1; [x, iter] = mid_point_deriv(@f, a, b, h, eps);
display([iter, x, f(x)])
h = 0.01; [x, iter] = mid_point_deriv(@f, a, b, h, eps);
display([iter, x, f(x)])
```

Результат:

```
18   -3.88128662109375   -255.0059127554973     ! УВАГА
19   -3.976531982421875   -255.9414906161229    ! УВАГА
2  -4  -256
20   -4.006546020507812   -256.0037991060724
19   -4.007781982421875   -256.0038986463916
```

Метод хорд. За цим методом функцію замінюють хордою на інтервалі $[a, b]$. Нехай значення першої похідної на кінцях інтервалу є різними за знаком $f'(a) < 0$ та $f'(b) > 0$, тобто функція є опуклою на інтервалі (рис. 6.1, рис. 6.2). За початкове наближення x_0 обирають той кінець інтервалу $[a, b]$, де $f(x_0) \cdot f''(x_0) < 0$. Послідовність наближень до мінімуму визначається рекурентною формулою

$$x_k = b - f'(b)/(f'(b) - f'(a)) \cdot (b - a), k = 0,1,2, ... \qquad (6.4)$$

Алгоритм звуження інтервалу невизначеності аналогічний до попереднього. Якщо $f'(x_k) = 0$ або $f'(x_k) \leq \varepsilon$ обчислення закінчують, $x^* = x_k$ є точкою мінімуму.

Метод дотичних (*Ньютона*) – метод другого порядку. За цим методом дугу кривої $y = f(x)$ на інтервалі $[a, b]$ замінюють її дотичною

$f'(x) \cong f'(x_0) - f''(x_0)(x - x_0)$. За початкове наближення x_0 обирають той кінець інтервалу $[a, b]$, де $f(x_0) \cdot f''(x_0) > 0$. Послідовність наближень до мінімуму визначається рекурентною формулою

$$x_{k+1} = x_k - f'(x_k)/f''(x_k), k = 0,1,2, \ldots . \qquad (6.5)$$

Основна проблема – правильне обрання точки x_0. Умовою зупинення обчислень є $|x_{k+1} - x_k| < \varepsilon$.

Метод квадратичної апроксимації. Нехай для унімодальної опуклої та двічі безперервно диференційованої на відрізку $[a, b]$ функції $f(x)$ відомими є $y_1 = f(a)$, $y_2 = f(b)$, $y_{11} = f'(a)$, $y_{12} = f'(b)$. Якщо $y_{11} \cdot y_{12} < 0$, можна побудувати єдиний інтерполяційний поліном третього степеня (кубічний інтерполяційний поліном Ерміта)

$$P_3(x) = c_0 + c_1(x - a) + c_2(x - a)(x - b) + c_3(x - a)^2(x - b) \qquad (6.6)$$

де $c_0 = y_1$; $c_1 = (y_2 - y_1)/(b - a)$; $c_2 = (y_2 - y_1)(b - a)^{-2} - y_{11}/(b - a)$; $c_3 = (y_{12} - y_{11})(b - a)^{-2} - 2(y_2 - y_1)(b - a)^{-3}$.Застосувавши до (6.4) необхідні умови існування екстремуму $\partial P_3(x)/\partial x = 0$, отримують рівняння
$c_3(x - a)^2 - 2(y_{11} + y_{12} - 3c_1)/(b - a) + y_{11} = 0$,
розв'язком якого є

$$\begin{gathered} x_0 = a + \mu(b - a), \\ \mu = (\omega + z - y_{11})/(2\omega - y_{11} + y_{12}), \\ z = y_{11} + y_{12} - 3(y_2 - y_1)/(b - a), \\ \omega = \sqrt{z^2 - y_{11}y_{12}}, \mu \in (0,1), x_0 \in (a, b). \end{gathered} \qquad (6.7)$$

Алгоритм методу квадратичної апроксимації (6.6)-(6.7).

Задають $\varepsilon > 0$, за умови $f'(a) \cdot f'(b) < 0$ знаходять дві точки a та b.

Крок 1. Обчислюють функцію та її перші похідні в точках a та b. За формулою (6.1) обчислюють коефіцієнти полінома P_3.

Крок 2. Обчислюють x_0 та $f'(x_0)$. Якщо $|f'(x_0)| < \varepsilon$, то обчислення закінчують, а точка $x^* = x_0$. Якщо $f'(x_0) < 0$, то $a = x_0$, інакше $b = x_0$ та повторюють з кроку 1.

Резюме

Даною темою ми закінчили розгляд методів оптимізації функцій однієї змінної. Методи першого та другого порядків забезпечують достатньо швидкій пошук мінімуму функції, але при цьому беруть до уваги та використовують значення першої та (або) другої похідної функції на кожному кроці роботи алгоритму, що може призвести до серйозних похибок результатів обчислень.

Для функцій однієї змінної, як правило, не дуже складно знайти аналітичний вираз для похідних першого та другого порядків. Для функцій заданих таблицями свої значень та для наборів експериментальних даних використовують методи обчислень на сітках. В даній темі ми продиференціювали кілька функцій за допомогою формул, отриманих із многочлена Ньютона та побачили, що похибка обчислень достатньо велика. Також ми побачили, що при недостатній увазі до параметрів функцій *Octave* ми отримали велику похибку обчислень.

Реалізували метод середньої точки та візуалізували процес наближення до шуканої точки, а порівнявши результати обчислень узнали, що для досягнення тієї ж самої точності, методам вищих порядків треба менше ітерацій.

Функції *Octave*, які використовуються в темі: $fminunc()$, $diff()$, $deriv()$, $polyder()$, $gradient()$, $nrm()$, $nonlin\_min()$, $plot()$.

Питання для самоперевірки

1. Які недоліки прямих методів усувають методи вищих порядків?
2. Яким критеріям повинна задовольняти функція, щоб можна було застосувати методи першого і другого порядків?
3. Дайте порівняльну характеристику методів хорд і дотичних.
4. Основний недолік методу Ньютона?
5. Переваги та недоліки методу квадратичної апроксимації в порівнянні з іншими методами?
6. Поясніть формулу (6.4) із методу хорд.
7. Поясніть формулу (6.5) із методу Ньютона.

Практичне завдання

Знайти мінімум функції згідно з варіантом: а) за допомогою програми на мові *Octave*; б) за допомогою вбудованих функцій. При розробці алгоритмів для диференціювання функцій використовувати формули із табл. 6.1, 6.2.

Порядок виконання завдання

1. Для функції визначити функцію користувача та побудувати її графік засобами *Octave*.
2. Проаналізувати приклад реалізації методу середньої точки та аналогічно реалізувати метод хорд (6.4).
3. Проаналізувати приклад реалізації методу середньої точки та аналогічно реалізувати метод Ньютона (6.5).
4. Показати на графіку перші кроки роботи алгоритму.
5. Порівняти результати за п. п. 2-3 із аналогічними, отриманими за допомогою $fminunc(\ )$.
6. Побудувати графік залежності кількості ітерацій від заданої точності. Для побудови графіка провести 10 вимірювань, початкова похибка $\varepsilon = 0.1$.

ТЕМА 7
ПРЯМІ МЕТОДИ ПОШУКУ МІНІМУМУ ФУНКЦІЇ БАГАТЬОХ ЗМІННИХ

Метою є ознайомлення студентів з прямими методами безумовної оптимізації функції багатьох змінних засобами *Octave*.

Постановка задачі. Нехай задано функцію $f(\vec{x})$: $\mathbb{R}^n \to \mathbb{R}$, де $\vec{x}$ – вектор $(x_1, \ldots, x_n)$. Знайти мінімум цієї функції $f(\vec{x}) \to \min\limits_{x \in \mathbb{R}^n}$ з заданою похибкою $\varepsilon > 0$.

Загальна характеристика методів прямого пошуку. У методах прямого пошуку для визначення напрямку спуску до точки мінімуму не використовують інформація про градієнт функції або її матрицю Гессе[17]. Напрямок мінімізації повністю визначається послідовністю обчислених значень функції. Як правило, безумовна мінімізація за методами першого і другого порядків забезпечує вищу швидкість збіжності, але, на практиці, аналітичне визначення похідних функції багатьох змінних не завжди можливе, а застосування числових методів веде до похибок і помилок. В інших випадках, критерій оптимальності може бути заданий не в явному вигляді, а системою рівнянь, що ускладнює або унеможливлює визначення похідних.

Перевагою методів прямого пошуку є простота реалізації у вигляді комп'ютерної програми, при цьому не потрібно знати цільову функцію в явному вигляді та легко врахувати обмеження, як на окремі змінні, так і на область пошуку.

У прямих методах будують послідовності векторів $\vec{x}^0$, $\vec{x}^1$, ..., $\vec{x}^n$, таких, що $f(\vec{x}^0) > f(\vec{x}^1) > \cdots > f(\vec{x}^n)$. Початкову точку $\vec{x}^0$ обирають або довільною, або таку, що розташована найближче до очікуваної точки мінімуму. Перехід (ітерація) від точки $\vec{x}^k$ до точки $\vec{x}^{k+1}$, $k = 0, 1, 2, \ldots, n$, складається з двох етапів – вибору напрямку руху з точки $\vec{x}_k$ та визначення кроку вздовж цього напрямку.

Дослідження функцій двох змінних зручно розпочати з побудови графіків. Функції двох змінних зручно представити графіком у вигляді ліній рівня. Лінією рівня функції $f(\vec{x}), \vec{x} = (x_1, x_2)$ називають геометричне місце точок площини, в яких функція $f(\vec{x}) =$ const. Через звичайну точку площини проходить тільки одна лінія рівня. Точки

[17] Матриця, елементами якої є другі частинні похідні

максимуму і мінімуму можуть бути оточені замкнутими лініями рівня, в сідлових точках перетинаються дві або більше лінії рівня.

Графіки функцій двох змінних

У пакет *Octave* включено велику кількість функцій для візуалізації функцій двох змінних. Ми розглянемо частину цих функцій, інші рекомендується читачеві спробувати самостійно, користуючись довідковою системою пакету.

• $surf(X,Y,Z)$, $mesh(X,Y,Z)$ – побудова графіку поверхні $Z=f(X,Y)$, а $[X,Y] = meshgrid(x,y)$, x, y – вектори значень.

• $contour\ (X,Y,Z)$ – побудова ліній рівня.

Для побудови графіків двох змінних потрібна підготовча робота, а саме створення спеціальної структури *meshgrid* – сітки, на якій будуть будуватись майбутній графік, наприклад, послідовність команд

$x = y = linspace\ (-10, 10, 25);\ [x,y] = meshgrid\ (x,y)$

створює сітку 25*x*25 елементів в діапазоні значень [-10;10] по обох координатах. Можна створювати різ різну кількість точок по координатах, звісно діапазони значень також можуть буди різними.

Приклад 7.1. Побудова графіків поверхні.

Задача – побудувати графік поверхні для функції, яку задано в аналітичному вигляді

```
## Приклад 7.1
clear all; clf; delete(gco())
function plot_setup(plot_handler)
  xlabel('X'); ylabel("Y"); zlabel('Z');
  set(gca, 'linewidth', 1.75, 'fontsize', 20,
           'gridalpha', 0.75, 'FontName', 'Times New Roman',
           'FontAngle', 'Italic');
  set(plot_handler, 'linewidth',2.5)
  set(get(gca,'zlabel'),'rotation',0)
endfunction
## підготовка графічної структури
x = y = linspace (-10, 10, 25)';
[x, y] = meshgrid (x, y);
## function to plot
f = @(x,y)sqrt (x .^ 2 + y .^ 2);
z = f(x,y);
##1
```

```
subplot(2,2,1)
  hsp = surf(x,y,z); colormap("gray"); view(3);
  title('Графік f(x,y)) - \rm{{surf}()}', 'fontsize', 16);
  plot_setup(hsp)
##2
subplot(2,2,2)
  hsp = surfc(x,y,z); colormap("gray"); view(3);
  title('Графік f(x,y)) - \rm{{surfc}()}', 'fontsize', 16);
  plot_setup(hsp)
##3
subplot(2,2,3)
  hsp = mesh(x,y,z); colormap("gray"); view(3);
  title('Графік f(x,y)) - \rm{{mech}()}', 'fontsize', 16);
  plot_setup(hsp)
##4
subplot(2,2,4)
  hsp = waterfall(x,y,z); colormap("gray"); view(3);
  title('Графік f(x,y)) - \rm{{waterfall}()}','fontsize', 16);
 plot_setup(hsp)
```

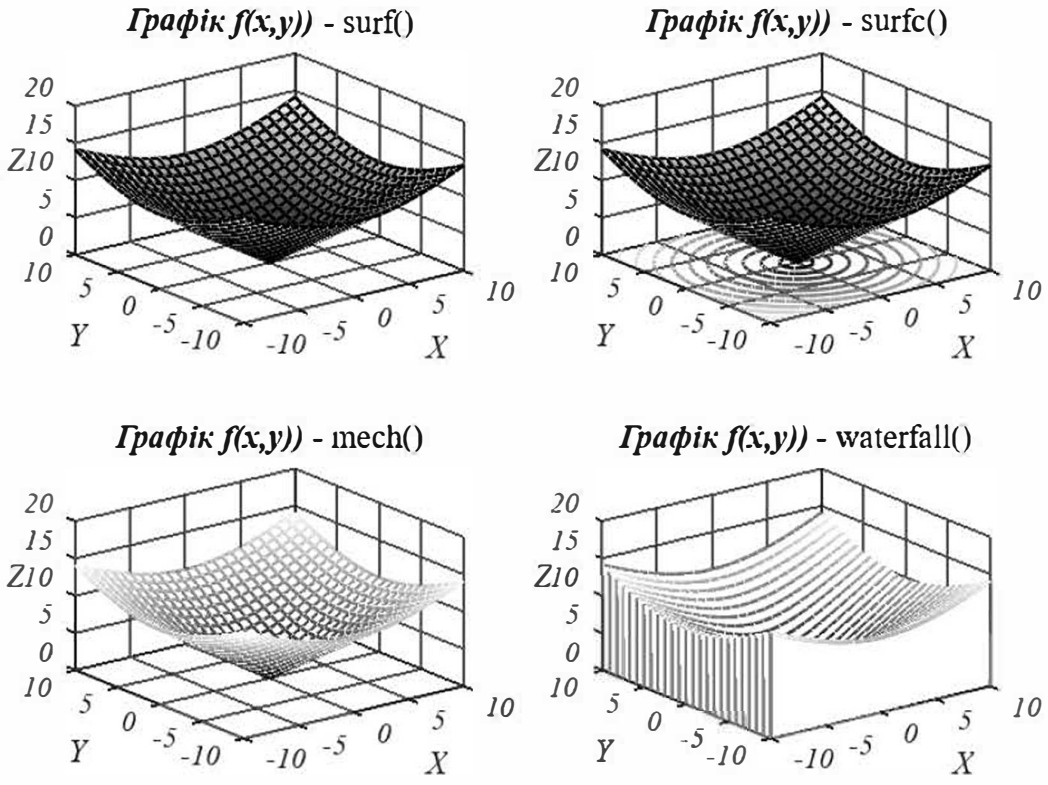


Рисунок 7.1

В наступному прикладі 7.2 показано використання функції *peaks*(), що будує на визначеній сітці значення змінної *Z*. Для підпису ліній рівня використовують функцію *clabel*(*c*, *hsp*), в якій c – мітки, *hsp* – вказівник на графік, побудований за допомогою *contour*(). Для побудови графіків (рис. 7.1) необхідно замінити *surf*() та *mesh*() на *contour*().

Приклад 7.2. Графік ліній рівні для функцій двох змінних

Задача – побудувати графік функції двох змінних для функції, заданої аналітично.

```
## Приклад 7.2
x = y = linspace (-10, 10, 50)';
[x, y] = meshgrid (x, y);
z = peaks(50);
  [c, hsp] = contour(x,y,z);
  clabel(c, hsp, 'fontsize', 16);
title('Графік f(x,y) - \rm{{contour}()}','fontsize', 16);
```

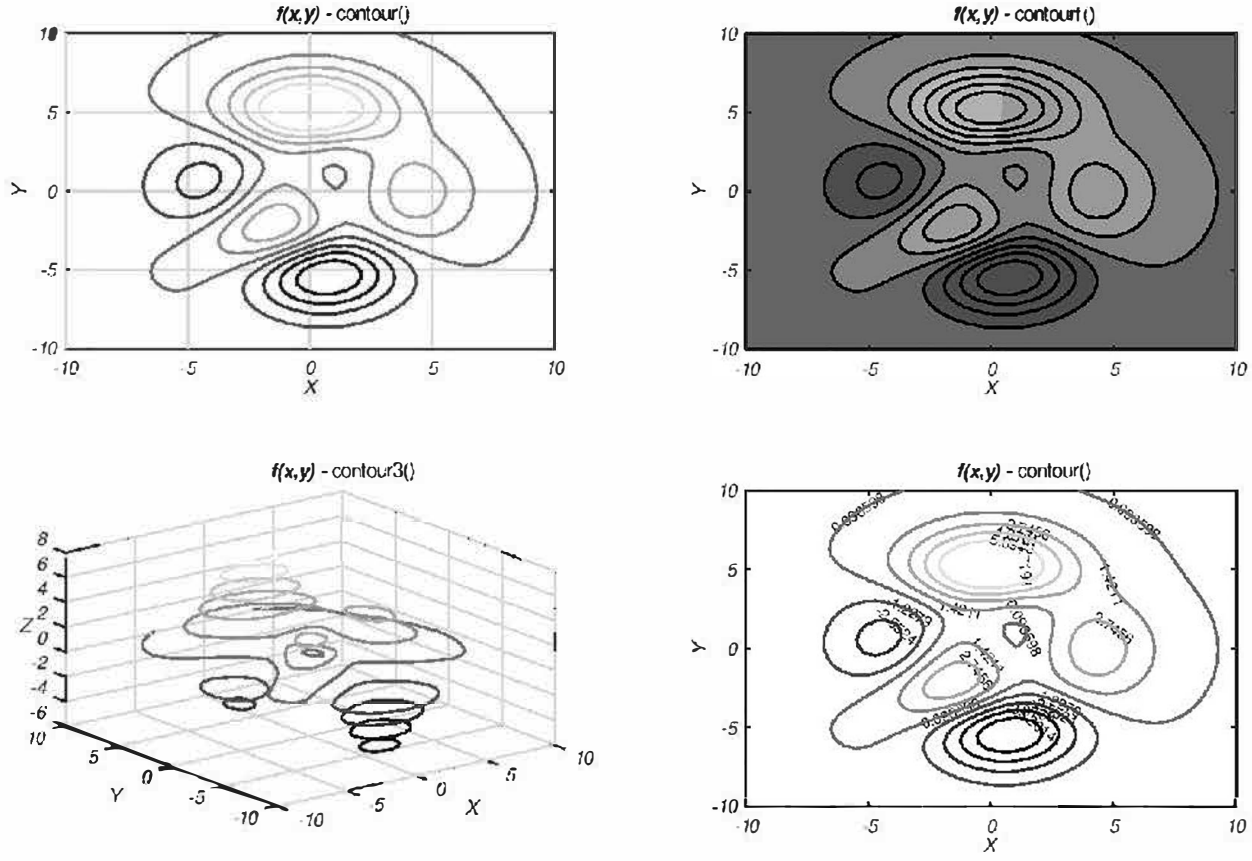


Рисунок 7.2

Приклад 7.3. Візуалізація матриці

Задача – відобразити матрицю у вигляді кольорових пікселів. Для цього використаємо псевдокольори та функцію $pcolor()$, що залежать від вмісту комірок[18].

```
## Приклад 7.3
z=-peaks(30) + 6;
colormap('gray')
pcolor(z)
```

Приклад 7.4. Побудова графіків на основі даних із файлу.

Задача – задано файл із даними, структура якого відповідає вимогам до *meshgrid*, побудувати графік поверхні

Подібну задачу ми рішали для одновимірних графіків (приклад 2.6), нижче наведено тільки фрагмент відповідної програми.

```
## Приклад 7.4
## перші два стовпці (x,y) повинні мати унікальні значення
## третій z = f(x,y)
## 10 0.1 200
## 15 0.2 300
## 20 0.3 250
...
m=2; n=2;
data = load('data.txt')
X = reshape(data(:,1),m,n)
Y = reshape(data(:,2),m,n)
Z = reshape(data(:,3),m,n)
mesh(X,Y,Z);
```

Метод координатного спуску (*підйому*) застосовують у разі, якщо досліджувана функція є унімодальною.

Алгоритм методу.

Задати вектор $\vec{x}^0 \in \mathbb{R}^n$ та похибку $\varepsilon > 0$. В якості критерію зупинення обчислень вибирати $\left\|\vec{x}^{k+1} - \vec{x}^k\right\|_2 \le \varepsilon$ або $\left|f(\vec{x}^{k+1}) - f(\vec{x}^k)\right| \le \varepsilon$.

Крок 1. Повторюють для $i = 1 \ldots n$, поки не виконаний критерій зупинення:

[18] Див. також функції $image(\,), imagesc(\,)$

- фіксують значення всіх змінних, крім x_i;
- одновимірна оптимізація отриманої функції.

Результат $\vec{x}^k = (x_1, x_2, \dots, x_n)$, де k – крок оптимізації.

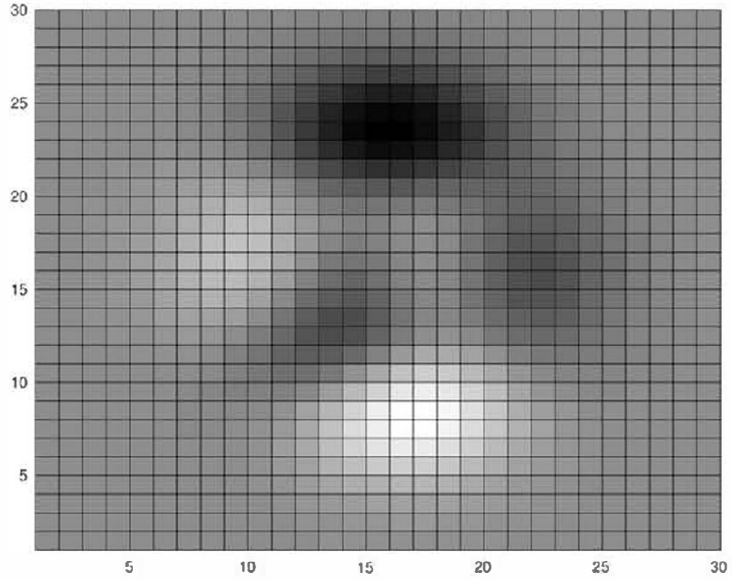


Рисунок 7.3

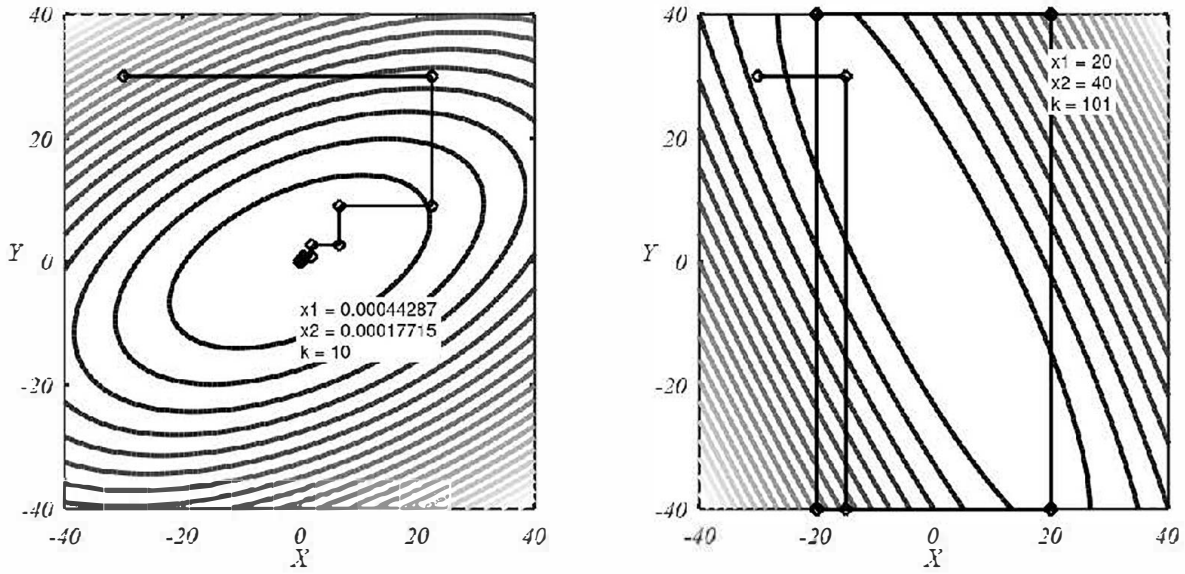
Приклад 7.5. Метод координатного спуску.

Задача – реалізувати метод координатного спуску із візуалізацією кроків роботи алгоритму на графіку.

Результат наведено на рис. 7.4а – мінімум знайдено, 7.4б – мінімум не знайдено.

```
## Приклад 7.5
clear all; more off;
function [z]=func(x1, x2, c)
  z=-(c(1)*x1.^2 + c(2)*x2.^2 + c(3)*x1.*x2);
endfunction
k = 1; kmax = 100; % лічильник ітерацій
eps = 0.001; % похибка
c=[-2, -5, 3]; % pass, також див. приклад 8.1
## c=[-3 -1 -3]; % fail
x=[-30, 30]; % початкові значення
xt = [x(1), x(2)]; % трасировка зміни x
% основний цикл методу
do
  x(1) = fminbnd(@(xx)func(xx, x(2), c), -40, 40); % спуск за
x(1)
```

```
  xt = [xt; x]; % додаємо точку
  x(2) = fminbnd(@(xx)func(x(2), xx, c), -40, 40) ; % спуск за
x(2)
  xt = [xt; x]; % % додаємо точку
until (norm(x,'fro') <= eps || ++k > kmax)
## Побудова графіку
xx = linspace(-40,40); yy = linspace(-40,40);
[X, Y] = meshgrid(xx, yy); Z=func(X, Y, c);
colormap (gray); contour(X, Y, Z, 20, 'fill',
'off','linewidth',5);
hold on;
## нанесення точок
plot(xt(:,1), xt(:,2), '.d-k', 'LineWidth', 3.25,
'markersize',12);
text(x(1) + 0, x(2)- 11, strvcat(['x1 = ' num2str(x(1))],
['x2 = ' num2str(x(2))], ['k = ' num2str(k)]),
'FontSize', 18, 'backgroundcolor', 'white');
xlabel('X'); ylabel("Y")
set(get(gca,'ylabel'),'rotation',0)
set(gca, 'linewidth', 2.5, 'fontsize', 24, 'gridalpha', 0.75,
'FontName', 'Times New Roman', 'FontAngle', 'Italic');
hold off
## зберігання файлу
print('theme_7_ex1_coord_good.png', '-r600', '-dpng', '-
S600,600');
```



а) б)

Рисунок 7.4

За методом Хука-Дживса напрямок спуску вибирається шляхом дослідження околу точки, знайденої на попередньому кроці. Алгоритм методу складається з послідовності кроків дослідницького пошуку (вибору точки) навколо базисної точки та пошуку за зразком. Результуючий напрямок на k-й ітерації є вектором $\vec{x}^k - \vec{x}^{k-1}$. На етапі дослідження використовують вектор переміщень, множник прискорення і масштабний коефіцієнт, на який ділять координати вектора в тому випадку, якщо в результаті дослідницького пошуку алгоритм не зміг зрушити з попередньої точки. Під час пошуку за зразком використовують дані, що були отримані в процесі дослідження, а мінімізація функції завершується пошуком у напрямку, заданому зразком.

За методом деформованого багатогранника (*метод Нелдера – Міда*) для мінімізації функції $f(\vec{x})$ в n-вимірному просторі будують багатогранник, що містить $(n + 1)$ вершину, яка відповідає деякому вектору. Обчислюють значення цільової функції $f(\vec{x})$ в кожній з вершин багатогранника, визначають максимальне з цих значень та відповідну йому вершину $\vec{x}_h$. Через цю вершину і центр ваги інших вершин проводять пряму, на якій знаходиться точка $\vec{x}_q$ з меншим значенням цільової функції, ніж у вершині $\vec{x}_h$. Надалі вершину $\vec{x}_h$ вилучають, а з решти вершин і точки $\vec{x}_q$ будують новий багатогранник, з яким повторюють описану процедура. У процесі виконання даних операцій багатогранник змінює свої розміри, що й зумовило назву методу.

За методом Розенброка пошук оптимальної точки здійснюють в кожному напрямку. Величина кроку в процесі пошуку безперервно змінюється залежно від рельєфу поверхні рівня. Поєднання обертання координат з регулюванням кроку робить метод Розенброка ефективним для розв'язання складних завдань оптимізації.

За методом Пауелла використовують властивість квадратичної функції таку, що будь-яка пряма, яка проходить через точку мінімуму функції, перетинає під рівними кутами дотичні до поверхонь рівного рівня функції в точках перетину.

Приклад 7.6. Мінімізація функції двох змінних.

Задача – написати програму мінімізації функції двох змінних за допомогою функції *fminsearch*( ).

```
## Приклад 7.6
clear all; clc; pkg load optim;
function cost = foo (x) % min=(1,1,1), f=3.0
  x--; cost = sum (-cos(x) + x.^2/9);
endfunction
x0 = [-15; -15; -15]; % початкова точка
format long
x = fmins ("foo", x0);
display ('fmins (за замовчанням)')
display(x')
## для розуміння значень у векторі параметрів
## optim_doc('fmins') (вихід ізі режим читання - натиснути
'q')
x = fmins ("foo", x0, [0, 1e-3,0,0,1,0,0,0,0,1000]); % Nelder
& Mead simplex
display("Nelder & Mead simplex");
display(x'); display(foo(x));
x = fmins ("foo", x0, [0, 1e-3,0,0,1,1,0,0,0,1000]);
display("Multidirectional search Method");
display(x'); display(foo(x));
x = fmins ("foo", x0, [0, 1e-3,0,0,1,2,0,0,0,1000]);
display("Alternating Directions search");
display(x'); display(foo(x))
x = nrm(@foo, x0);
display("Newton-Raphson Method");
display(x');display(foo(x))
opt=optimset('TolX', 1e-15,'MaxIter', 1000);
[x, fval]=fminsearch (@foo, x0, opt);
display("fminsearch");
display(x');display(foo(x))
```

Результат:

```
>>
fmins (за замовчанням) 0.99962   0.99954   0.99989
-3.0000
Nelder & Mead simplex 0.99962   0.99954   0.99989
-3.0000
Multidirectional search Method  0.99976   1.00159   0.99976
```

```
-3.0000
Alternating Directions search   0.99602   0.99602   0.99602
-3.0000
Newton-Raphson Method    1.0000   1.0000   1.0000
-3
fminsearch    1.00000   1.00000   1.00000
-3
```

Резюме

Тема мінімізації функцій багатьох змінних складна та багатогранна. Прямі методи пошуку не використовують дані про похідні, їх відносно просто реалізувати, не потрібно знати цільову функцію в явному вигляді та легко врахувати обмеження, як на окремі змінні, так і на область пошуку.

Безумовна мінімізація за методами першого і другого порядків[19] забезпечує вищу швидкість збіжності, але, на практиці, аналітичне визначення похідних функції багатьох змінних не завжди можливе, а застосування числових методів веде до похибок і помилок.

Ми познайомились із стандартними функціями - $fminsearch()$ та $powell()$, в яких реалізовані прямі методи пошуку, а також розробили програму для методу координатного спуску.

Функції для індивідуальних завдань – спеціальні функції з різними особливостями, тому їх використовують для тестування алгоритмів оптимізації.

Також ми навчилися будувати графіки поверхонь та графіки у вигляді ліній рівня, що забезпечує гарну візуалізацію та інформацію, достатню для аналізу поводження функції на заданому інтервалі. Функції *Octave*, які використовуються в роботі: $fminsearch()$, $powell()$, $nonlin\_opt()$, $surf()$, $contour()$.

Питання для самоперевірки

1. Опишіть алгоритм методу координатного спуску. Як буде себе вести цій метод для функцій з малою кривизною в області мінімуму?
2. Опишіть алгоритм методу Хука-Дживса.
3. Опишіть алгоритм методу Нелдера-Міда.

[19] Див. наступну тему

4. Сформулюйте переваги і недоліки методів прямого пошуку.

5. Яким чином впливає вибір $\vec{x}^0$ на результат та кількість обчислень?

6. Чому при побудові обчислювального процесу оптимізації функції зазвичай обмежують кількість ітерацій?

7. Побудуйте графік залежності $n = f(\varepsilon)$. Як пов'язані похибка та кількість обчислень?

Практичне завдання

Знайти мінімум функції $f(x, y)$ на визначеному інтервалі. Для функцій, що приймають більше двох аргументів, рекомендується під час виконання роботи вважати їх за функції двох змінних [11, 12]. Варіанти див. у попередній темі.

Порядок виконання завдання

1. Побудувати графіки досліджуваної функції за допомогою функцій $surf(\,)$ та $contour(\,)$.

2. Написати програму оптимізації за методом координатного спуску.

3. Реалізувати та дослідити метод Хука-Дживса [10].

4. Порівняти результати із п. п. 2 і 3 з результатами роботи функції *Octave* $fminsearch(\,)$,в якій реалізований метод деформованого багатогранника. Змініть вхідні параметри - $\vec{x}^0$, ε, Δx та порівняйте результати.

5. Порівняти результати за п. п. 2 і 3 з результатами роботи функції *Octave* $powell()$,в якій реалізований метод апроксимації Пауелла. Змініть вхідні параметри - $\vec{x}^0$, ε, Δx та порівняйте результати.

Індивідуальні завдання

Знайти мінімум функції $f(x, y)$ на визначеному інтервалі. Для функцій, що приймають більше двох аргументів, рекомендується під час виконання роботи вважати їх за функції двох змінних [11, 12].

1. Функція *Розенброка* $f(x) = \sum_{i=1}^{n-1}\left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$, $\min|_{n=2} = f(1; 1) = 0, -\infty \le x_i \le \infty, 1 \le i \le n.$

2. Функція *De Jong* (*сфера*) $f(x) = \sum_{i=1}^{n} x_i^2$,
$f(x_1; \ldots; x_n) = f(0; \ldots; 0) = 0, -\infty \le x_i \le \infty, 1 \le i \le n.$

3. Функція *Beale* $f(x,y) = (1{,}5 - x + xy)^2 + (2{,}25 - x + xy^2)^2 +$ $+(2{,}625 - x + xy^3)^2$, $f(3;\,0{,}5) = 0$, $-4{,}5 \le x, y \le 4{,}5$.

4. *Функція Goldstein-Price*
$f(x,y) = [1 + (x + y + 1)^2(19 - 14x + +3x^2 - 14y + 6xy + 3y^2)] \times$
$\times\,[30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]$,
$f(0;-1) = 3$, $-2 \le x, y \le 2$.

5. Функція *Booth* $f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$,
$f(1;3) = 0$, $-10 \le x, y \le 10$

6. Функція *Matyas* $f(x,y) = 0{,}26(x^2 + y^2) - 0{,}48xy$, $f(0;0) =$ 0, $-10 \le x, y \le 10$.

7. Функція *McCormick* $f(x,y) = \sin(x + y) + (x - y)^2 - 1{,}5x +$ $2{,}5y + 1$, $f(-0{,}54719; -1{,}54719) = -1{,}9133$, $-1{,}5 \le x \le 4, -3 \le$ $y \le 4$.

8. Функція *Powell* $f(\vec{x}) = \sum_{i=1}^{4}[(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} -$ $x_{4i})^2 + (x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4]$, $x_i \in [-4; 5], i = 1 \ldots d$, $f(\vec{x}^*) = 0, x^* = (0, \ldots, 0)$. Прийняти $i = 1$, графіки побудувати для фіксованих значеннях $x_1 = x_2 = \text{const}$ або $x_3 = x_4 = \text{const}$.

9. Функція *Styblinski-Tang* $f(\vec{x}) = \frac{1}{2}\sum_{i=1}^{d}(x_i^4 - 16x_i^2 + 5x_i)$, $f(\vec{x}^*) = -39{,}16599d$, $x^* = (-2{,}90534; \ldots; -2{,}90534)$.

10. Функція *Himmelblau* $f(x,y) = (x^2 + y - 11)^2 + (x + y^2 +$ $7)^2$, $x \in [-4; 4], y \in [-4; 4]$, $f(3{,}2) = 0$, $f(-2{,}8051; 3{,}1313) = 0$, $f(-3{,}7793; -3{,}2832) = 0$, $f(3.5844; 1.8481) = 0$.

11. Функція з двома глобальними мінімумами (*two–hump camel*) $f(x,y) = 1{,}0316285 + 4x^2 - 2{,}1x^4 + x^6/3 + xy - 4y^2 + 4y^4$, $\vec{x}^* = (0{,}898; -0.7126), (-0{,}0898;\ 0{,}7126)$.

12. Функція *Branin-Hoo*:
$f(\vec{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1 - t)\cos x_1 + s + 5x_1$,
$a = 1, b = 5.1\ /\ (4\pi^2), c = 5\ /\ \pi, r = 6, s = 10, t = 1/(8\pi)$,
$x_1 \in [-5, 10]$, $x_2 \in [0, 15]$,
$\vec{x}^* = \big((8{,}8893; 1{,}8893), (-3{,}6891; 13{,}6296), (\,2{,}5939;\ 2{,}7409)\,\big)$.

13. Функція з трьома глобальними мінімумами (*three–hump camel*):

$$f(x,y) = 2x^2 - 1.05x^4 + x^6/6 + xy + y^2,\ x \in [-5,5],$$

$y \in [-5;\ 5]$, $f(0;0) = 0$, $f(-1{,}74753;\ 0{,}87381) = 0{,}29864$, $f(1{,}74753;\ -0{,}87381) = 0{,}29864$.

14. Функція суми різних степенів $f(\vec{x}) = \sum_{i=1}^{n} |x_i|^{i+1}$, $x_i \in [-1;\ 1], i = 1 \ldots n,$, $f(0;0) = 0$.

15. Функція *Eason*: $f(\vec{x}) = -\cos x_1 \cos x_2 \mathrm{e}^{-(x_1-\pi)^2-(x_2-\pi)^2}$, $x_i \in [-100;100]$, $i = 1{,}2$, $f(0;0) = 0$.

Для роботи всіх стандартних мінімізаторів функції декількох змінних повинні приймати на вході вектор змінних x, тому функція , яку мінімізують має бути приведеною до відповідної форми.Нижче наведено приклад реалізації на мові *Octave* функції Розенброка, яка приймає вектор $\vec{x}$ та повертає значення, градієнт і матрицю Гессе[20]. (). Задані згідно з варіантом функції рекомендується реалізувати аналогічно.

Особливістю такої реалізації є те, що її результат залежить від кількості вхідних та (або) вихідних параметрів, для цього використовують стандартні *nargin* для аналізу вхідних, та *nargout* – вихідних параметрів.

Для спрощення приймаємо, що всі функції – функції двох змінних, що дозволить побудувати шрафік та дослідити роботу алгоритмів.

Можливі й інші варіанти, наприклад, для функції *sqp* похідні потрібно передавати у вигляді масиву (*cell array*), див. приклад в документації *GNU Octave*.

Приклад 7.7. Функція користувача для методів оптимізації.

Задача – написати функцію, що повертає значення функції, її першої похідної та матрицю Гессе.

```
## Приклад 7.7
function [f g H] = rosen2d(x)
  if nargin == 0;
```

[20] *https://github.com/probml/pmtk3/blob/master/demos/rosen2d.m*

```
     [f g H] = rosen2d(randn(100, 1));  return;  endl;
  if isvector(x) % функція
    f = 100*(x(2) - x(1)^2)^2 + (1-x(1))^2;
  else
    f = 100*(x(:,2) - x(:,1).^2).^2 + (1-x(:,1)).^2;
  if nargout > 1 % градієнт
    g = [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1)); 200*(x(2)-
x(1)^2)]; end
  if nargout > 2 % матриця Гессе
    H = [1200*x(1)^2-400*x(2)+2, -400*x(1); -400*x(1), 200];
  end
end
```

ТЕМА 8
ГРАДІЄНТНІ МЕТОДИ ПОШУКУ МІНІМУМУ ФУНКЦІЇ БАГАТЬОХ ЗМІННИХ

Метою є ознайомлення студентів градієнтними методами безумовної оптимізації функції багатьох змінних засобами *Octave*.

Постановка задачі. Нехай задано функцію $f(\vec{x})$: $\mathbb{R}^n \to \mathbb{R}$, де $\vec{x}$ – вектор $(x_1, \dots, x_n)$. Функція $f(\vec{x})$ повинна відноситься до класу безперервно диференційованих (або двічі безперервно диференційованих) функцій. Знайти мінімум цієї функції $f(\vec{x}) \to \min_{x \in \mathbb{R}^n}$ із заданою похибкою $\varepsilon > 0$.

Загальні відомості про градієнтні методи. Градієнтом функції $f(\vec{x})$ є вектор $(\partial f(x)/\partial x_1, \dots, \partial f(x)/\partial x_n)$

$$\nabla f = \frac{\partial f(\vec{x})}{\partial x_1}\vec{e}_1 + \frac{\partial f(\vec{x})}{\partial x_2}\vec{e}_2 + \cdots + \frac{\partial f(\vec{x})}{\partial x_n}\vec{e}_n,$$

де $\vec{e}_i = (e_1, e_2, \dots, e_n), i = 1 \dots n$ – одиничний вектор.

Вектор, протилежний градієнту, називають антиградієнтом. Градієнт перпендикулярний до ліній рівня та спрямований убік найшвидшого зростання функції в даній точці. У точці мінімуму(максимуму) градієнт функції дорівнює нулю. На властивостях градієнта засновані методи оптимізації першого порядку, які дозволяють визначити точку локального мінімуму функції. У разі цільових функцій з сильно витягнутими, вигнутими або такими, що мають гострі кути, лініями рівня, методи першого-другого порядку можуть виявитися нездатним забезпечити просування до точки мінімуму.

Загальний алгоритм градієнтних методів.

1. Задають початкову точку $\vec{x}^k \in \{X\}$ $(k = 0)$, визначають градієнт (∇^k), обирають напрямок руху з цієї точки до наступної точки

$$\vec{x}^{k+1} = \vec{x}^k - \gamma^k \nabla f(\vec{x}^k)$$

де γ^k – величина кроку, k – номер ітерації.

2. Перевіряють умову закінчення обчислень. Для завдань безумовної оптимізації ($x \in \mathbb{R}^n$) в точці екстремуму норма градієнта не перевищує задану похибку ε: $\left\|\nabla f(\vec{x}^k)\right\|_2 = \sqrt{\sum_{j=1}^{n}\left(\partial f(\vec{x}^k)/\partial x_j\right)^2} < \varepsilon$. У разі точного розв'язку в точці екстремуму $\left\|\nabla f(\vec{x}^k)\right\|_2 \equiv 0$. Якщо допустиму множину $\{X\}$ задано, то в точці розв'язку (якщо вона на границі) норма вектора-градієнта може бути будь-якою, а критерій зупинення обирають, обмежуючи збільшення цільової функції на двох послідовних ітераціях.

В задачах безумовної оптимізації за способом руху до екстремуму градієнтні методи можна розділити на дві групи:

- на $(k+1)$-й ітерації послідовно, одна за одною, обчислюють координати вектора $(k+1)$-ї точки (методи координатного спуску);
- всі координати $(k+1)$-ї точки обчислюють відразу.

Крок методу може бути:

- *постійним,* задають до початку розрахунків;
- *адаптивним*, використовують у разі неможливості визначити мінімум с заданим кроком γ^0, тобто $\left\|\nabla f(\vec{x}^k)\right\|_2$ починає збільшуватися з кроку k;
- *обчислюваним* на кожному кроці за кожним напрямком відповідно до визначеного критерію.

Метод координатного спуску (*Гаусса – Зейделя*). На черговій ітерації поступово здійснюють спуск уздовж кожної з координат, а крок γ^k обчислюють n-разів на кожній ітерації.

Алгоритм методу. Задають вектор $\vec{x}^0$ та похибку $\varepsilon > 0$.

Крок 1. Повторюють:

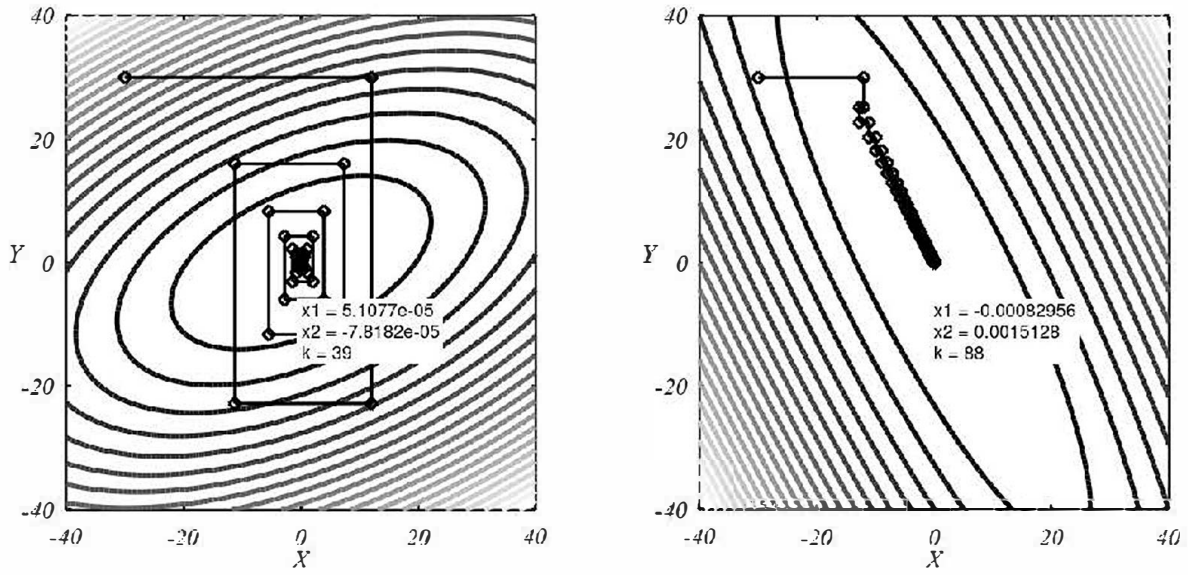
- розраховують крок $\gamma_i^k = \arg\min_{\gamma_i^k \geq 0} f\left(\vec{x}_i^k - \gamma_i^k \nabla f\left(\vec{x}_i^k\right)\right)$, $i = 1 \ldots n$ – відповідний елемент вектора $\vec{x}^k$;
- розраховують $\vec{x}_i^{k+1} = \vec{x}_i^k - \gamma_i^k \nabla f(\vec{x}_i^k)$;
- якщо $\left\|\vec{x}_n^k - \vec{x}_0^k\right\|_2 > \varepsilon$, то $\vec{x}_0^{k+1} = \vec{x}_n^k$, $k = k+1$; інакше закінчують обчислення, а точка $\vec{x}^* \cong \vec{x}^k$ є шуканою точкою мінімуму.

Приклад 8.1. Пошук мінімум функції декількох змінних.

Задача – написати програму для реалізації метода оптимізації *координатного спуску* (*Гаусса – Зейделя* із візуалізацією результатів обчислення.

Результат наведено на рис. 8. В попередній темі наведено приклад роботи аналогічного методу нульового порядку (рис. 7.4). Як ми пам'ятаємо, там при виборі іншої початкової точки не було знайдено розв'язок. В даному випадку метод знайшов мінімум, але за достатньо велику кількість кроків (рис. 8.1б). В цьому прикладі і далі базовою є програма для координатного спуску із теми 7 та програма з цього прикладу, далі зміни вносяться тільки в основний робочий цикл.

```
## Приклад 8.1
clear all; more off;
function [z, df]=func(c, x, y)
  if (nargin==2)
     z=c(1)*x(1).^2 + c(2)*x(2).^2 + c(3)*x(1).*x(2);
  endif
  if (nargin==3)
     z=c(1)*x.^2 + c(2)*y.^2 + c(3)*(x.*y);
  endif
  if (nargout ==2) && (nargin ==2) % градієнт
    df(1)=2*c(1)*x(1) + c(3)*x(2);  df(2)=2*c(2)*x(2) +
c(3)*x(1); endif
  z=-z;
endfunction
c=[-2 -1 1]; x=[-30 30];% початкові значення
g = 0.2;             % постійна кроку
d = 0.00001;         % похибка
k = 1; kmax = 100;  % лічильник ітерацій
x1t = [x(1)]; x2t = [x(2)]; % проміжні точки
xt = [x(1), x(2)];
do % основний робочий цикл
  [temp, gr] = func(c, x); % спуск за першою координатою
  x(1) = x(1) + g * gr(1);
  xt = [xt ; x];
  [temp, gr] = func(c, x); % спуск за другою координатою
  x(2) = x(2) + g * gr(2);
  xt = [xt ; x];
until (norm(gr,'fro') <= d || ++k > kmax)
## побудова графіка, див. тему 7, приклад 1.
```



а) б)

Рисунок 8.1

Градієнтний метод з постійним кроком.

Крок 1. Задають вектор $\vec{x}^0$, крок $\gamma > 0$, похибку $\varepsilon > 0$ та $k = 0$.

Крок 2. Повторюють:

- розраховують вектор градієнта $\nabla f(\vec{x}^k)$;
- якщо $\left\|\nabla f(\vec{x}^k)\right\|_2 \le \varepsilon$, то $\vec{x}^* \cong \vec{x}^k$ – припиняють обчислення;

інакше обчислюють $\vec{x}^{k+1} = \vec{x}^k - \gamma^k \nabla f(\vec{x}^k)$.

Приклад 8.2. Градієнтний метод з постійним кроком.

Задача – написати основний цикл градієнтного методу.

```
## Приклад 8.2
do
  [temp, gr] = func(c, x); % спуск за обома координатою
  xt = [xt ; x];
until (norm(gr,'fro') <= d || ++k > kmax)
```

Більш гнучким є *градієнтний метод із адаптивним кроком.* Задають вектор $\vec{x}^0$, крок $\gamma > 0$, похибку $\varepsilon > 0$ та $k = 0$.

Крок 1. Розраховують вектор градієнта $\nabla f(\vec{x}^k)$.

Крок 2. Повторюють:

- розраховують норму градієнта $\left\|\nabla f(\vec{x}^k)\right\|_2$;
- пробний крок $\vec{x}^{k+1} = \vec{x}^k - \gamma^k \nabla f(\vec{x}^k)$;

• якщо $\left\|\nabla f(\vec{x}^{k+1})\right\|_2 < \left\|\nabla f(\vec{x}^{k})\right\|_2$ та $\left\|\nabla f(\vec{x}^{k})\right\|_2 > \varepsilon$, то $k = k + 1$;

• інакше, якщо $\left\|\nabla f(\vec{x}^{k})\right\| < \varepsilon$, то $\vec{x}^* \cong \vec{x}^k$ – точка мінімуму, припиняють обчислення;

• інакше, якщо $\left\|\nabla f(\vec{x}^{k+1})\right\|_2 > \left\|\nabla f(\vec{x}^{k})\right\|_2$, то відновлюють значення вектору $\vec{x}^{k+1} = \vec{x}^k + \gamma^k \nabla f(\vec{x}^k)$ та змінюють крок $\gamma = \gamma \cdot \delta$, де δ – масштабний множник, наприклад 0,5.

Приклад 8.3. Градієнтний метод із дробленням кроку.

Задача – написати основний цикл градієнтного методу із змінним кроком.

Порівняно до методу із фіксованим кроком (рис. 8.2а) отримали зменшення кількості кроків на 20% (рис. 8.2б)..

```
## Приклад 8.3
[temp, gr1] = func(c, x); g = 0.5;
do
    x = x + g * gr1; % пробний крок
    [temp, gr2] = func(c, x); % переахунок градієнтів
    if norm(gr2, 'fro') < norm(gr1, 'fro');
        gr1 = gr2;
        xt = [xt ; x];
    else
        x = x - g * gr1; % відновлення
        g = 0.5 * g; % дроблення кроку
    end
until (norm(gr1, 'fro') <= d || ++k > kmax)
```

Метод найшвидшого спуску відноситься також до методів другого порядку. Алгоритм методу наступний.

Крок 1. Задають початковий вектор $\vec{x}_0$, похибку $\varepsilon > 0$ та $k = 0$.

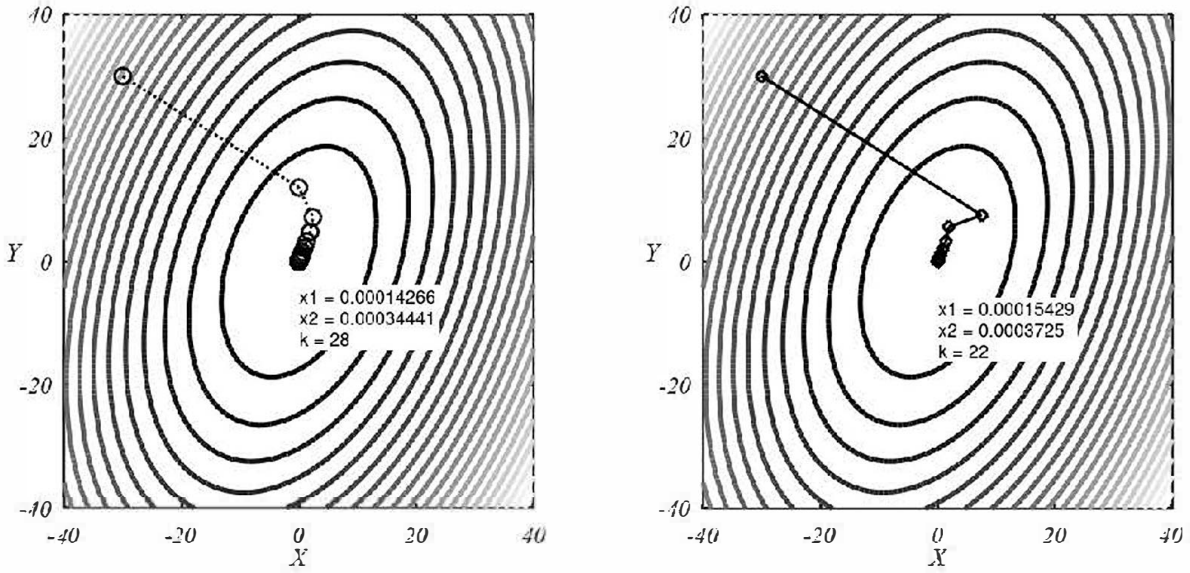
Крок 2. Визначають вектор градієнта $\nabla f(\vec{x}^k)$ і нормований вектор градієнта $\nabla f(\vec{x}^k) / \left\|\nabla f(\vec{x}^k)\right\|_2$.

Перевіряють умову закінчення обчислень $\left\|\nabla f(\vec{x}^k)\right\|_2$.

Крок 3. Повторюють:

• розраховують $\gamma^k = \arg\min_{\gamma^k \geq 0} f(\vec{x}_k - \gamma^k \nabla f(\vec{x}^k))$ – крок в обраному напрямку – одним з методів одновимірної оптимізації.

- розраховують $\vec{x}^{k+1} = \vec{x}^k - \gamma^k \nabla f(\vec{x}^k)$;
- якщо $\left\|\nabla f(\vec{x}^k)\right\|_2 > \varepsilon$, то $k = k + 1$, інакше $\vec{x}^* \cong \vec{x}^k$ – точка мінімуму, припиняють обчислення.



а) б)

Рисунок 8.2

Приклад 8.4. Метод найшвидшого спуску.

Задача – написати основний цикл методу найшвидшого спуску.

Результат роботи програми наведено на рис. 8.3.

```
## Приклад 8.4
do
  [temp , gr] = func(c, x); % отримуємо градієнт
  g = -sumsq(gr)/(2*c(1)*gr(1)^2 +
      2*c(2)*gr(2)^2 + 2*c(3)*gr(1)*gr(2));
  x = x + g * gr;
  xt = [xt ; x];
until(norm(gr, 'fro') <= d || ++k > kmax)
```

За методом Ньютона (другого порядку) використовують точні або наближені значення других часткових похідних цільової функції для обчислення матриці Гессе $H(\vec{x}^k)$ та вектору $\vec{x}^k$

$$\vec{x}^{k+1} = \vec{x}^k - \gamma^k H^{-1}(\vec{x}^k)\, \nabla f(\vec{x}^k)$$

де $H(\vec{x}^k) = \nabla^2 f(\vec{x}^k) \in \mathbb{R}^{nxn}$ – матриця Гессе, а H^{-1} – зворотна

матриця Гессе.

Алгоритм методу Ньютона.

Крок 1. Задають вектор $\vec{x}_0$, похибку $\varepsilon > 0$ та $k = 0$.

Крок 2. Повторюють:

• обчислюють $\nabla f(\vec{x}^k)$ і якщо $\left\|\nabla f(\vec{x}^k)\right\|_2 < \varepsilon$, то $\vec{x}^* \cong \vec{x}^k$ – шукана точка мінімуму, припиняють обчислення;

• обчислюють матрицю Гессе $H(\vec{x}^k)$. Якщо не всі її власні значення є додатними, то припиняють обчислення;

• обчислюють напрямок пошуку $\vec{p}^k = -H^{-1}(\vec{x}^k)\nabla f(\vec{x}^k)$;

• методами одновимірної мінімізації шукають крок $\gamma^k = \arg\min_{\gamma \geq 0} f(\vec{x}^k + \gamma\vec{p}^k)$ та за (8.3) обчислюють $\vec{x}^{k+1}$, $k = k + 1$.

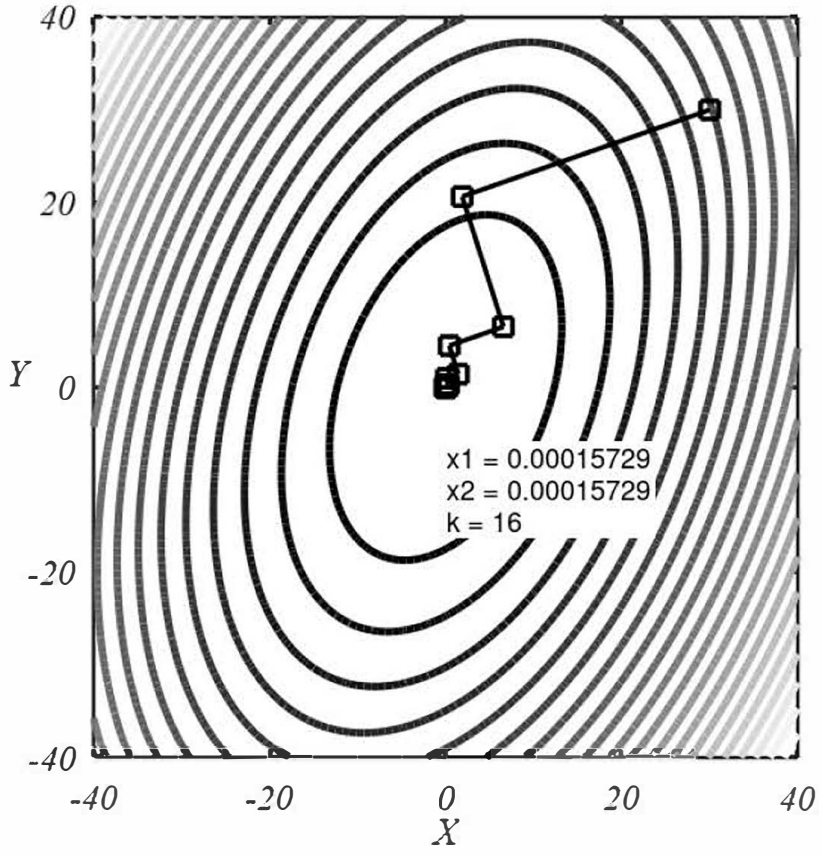


Рисунок 8.3

Методи нелінійної оптимізація в *Octave*

Для задач оптимізації використовують вбудовані функції (аналоги функцій *Matlab*) та пакет розширень *optim*. Функція *nonlin*_min() з нього є інтерфейсом для задач даного класу (замінює

застарілий $minimize()$). У пакет включено файл $optim\_problems.m$, в якому є приклади задач оптимізації. У всіх функціях:

fun – функція, мінімум якої шукають (або покажчик на неї);

$x0$ – початкова точка (вектор);

$control$ – структура $optimset$ для установки параметрів оптимізації[21];

x – знайдена мінімальна точка (або вектор);

y – значення функції в точці мінімуму;

$info$ – додаткова інформація про результати обчислень;

cvg – індикатор успішності процесу оптимізації.

Приклад 8.4. Мінімізація функції декількох змінних

Задача – написати програму мінімізації функцій декількох змінних за допомогою бібліотечних функцій *Octave*

Використання вбудованих функцій мінімізації (для порівняння додано функцію $fmins()$, що є фронтендом для декількох методів оптимізації, дозволяє обчислювати мінімум функцій багатьох змінних та вибирати потрібний алгоритм.

```
## Приклад 8.5 (nonlin_min())
## Приклад 1, тестова функція
clear all; clc;
pbl = optim_problems ().general.schittkowski_281;
[x, objf, cvg, outp] = nonlin_min (pbl.f, pbl.init_p,
                        optimset ("algorithm", "d2_min",
                       "objf_grad", pbl.dfdp, "objf_hessian",
pbl.hessian));
display('nonlin_min => Standard Schittkowski 281');
display(x); display(cvg);
## Приклад 2, тестова функція, функція користувача,
min=(1,1,1), f=3.0,
## див. документацію optim
function cost = foo (x)
  x--;  cost = sum (-cos(x)+x.^2/9);
endfunction
function dc = diffoo (x) ## перші частинні похідні
  x = x(:)' - 1; dc = sin (x) + 2*x/9;
endfunction
```

[21] (https://octave.sourceforge.io/optim/package_doc/Common-optimization-options.html

```
function [d2c] = d2foo (x) ## другі частинні похідні
  d2c = diag (cos (x(:)-1) + 2/9);
end
x0 = [-15; -15; -15];
[x, objf, cvg, outp]=nonlin_min (@foo, x0,
     optimset ("algorithm", "d2_min",
     "objf_grad", @diffoo, "objf_hessian", @d2foo));
display('nonlin_min => sum (-cos(x)+x.^2/9)');
display(x);display(cvg);
## Приклад 3, тестова функція + sqp
function r = g(x)
  r = [ sumsq(x)-10;
        x(2)*x(3)-5*x(4)*x(5);
        x(1)^3+x(2)^3+1 ];
endfunction
function obj = phi (x)
  obj = exp (prod (x)) - 0.5*(x(1)^3+x(2)^3+1)^2;
endfunction
x0 = [-1.8; 1.7; 1.9; -0.8; -0.8];
[x, obj, info, iter, nf, lambda] = sqp (x0, @phi, @g, []);
display('sqp => exp (prod (x)) -
0.5*(x(1)^3+x(2)^3+1)^2');display(x); display(info);
## Приклад 4, тестова функція
xin = [-2; 5];
obj_fun = @ (x) x(1)^2 + x(2)^2;
constr_fun = @ (x) x(1)^2 + 1 - x(2);
[x, objf, cvg, outp] = nonlin_min (obj_fun, xin,
         optimset ("equc", {constr_fun}));
display('nonlin_min => x1^2 + x2^2'); display(x);
display(cvg);
```

Результат:

```
>>
nonlin_min => Standard Schittkowski 281
x =
  0.9169552012980494
  0.9999999990379179
  0.9999999999999999
  1
  1
  1
  1
```

```
  1
  1
  0.9999289237761456
cvg = 0
>>
nonlin_min => sum (-cos(x)+x.^2/9)
x =   1   1   1
cvg = 3
>>
sqp => exp (prod (x)) - 0.5*(x(1)^3+x(2)^3+1)^2
x =
  -1.717143465253209
  1.595709568430945
  1.827245948287367
  -0.7636431065046502
  -0.7636430732426371
info = 104
>>
nonlin_min => x1^2 + x2^2
x =   -1.090917451549388e-08   0.9999999999999981
cvg = 3
```

Резюме

В даній темі ми розібрали приклади побудови програм для використання методів першого та другого порядків для мінімізації функції багатьох змінних. На прикладах функцій з двома змінними навчилися візуалізації роботи алгоритму на двовимірному контурному графіку.

На прикладах познайомилися із роботою нелінійних оптимізаторів – *nonlin_min*(), що є фронтендом для декількох методів, та *sqp*(), що шукає мінімум на основі задачі квадратичного програмування. Функція *sqp*() міститься у ядрі пакету, а *nonlin_min*() – у пакеті *optim*. Для точного налаштування *nonlin*_min() використовують спеціальну структуру *optimset* (див. приклад 8.4).

Для методів оптимізації, що були розглянуті в даній темі, активно використовуються частинні похідні, для знаходження аналітичного вигляду яких для складних функцій можна використати пакет *symbolic*.

Функції *Octave*, які вивчались – *sqp*( ), *nrm*( ), *surf*( ), *contour*( ), *nonlin_min*( ).

Практичне завдання

Знайти мінімум функції $f(x, y)$ на визначеному інтервалі. Для функцій, що приймають більше двох аргументів, рекомендується під час виконання роботи вважати їх за функції двох змінних [11, 12]. Варіанти див. у попередній темі.

Порядок виконання завдання

1. Побудувати графік досліджуваної функції за допомогою функцій $surf(\,)$ та $contour(\,)$.

2. Дослідити приклади 1-4 програм, змінити початкові умови для реалізованих методів оптимізації $\vec{x}_0$, ε, γ та порівняти результати. Прийняти c=[-3 -1 -3] та порівняти методи оптимізації. Скласти порівняльну таблицю, зробити висновки.

3. Змінити програми п. 2 для роботи з заданою згідно з варіантом функцією, дослідити поведінку програми за різних початкових умов.

4. Дослідити функції *Octave,* в яких реалізовано методи першого-другого порядків. Знайти мінімум для своєї функції за допомогою $nonlin\_\min()$ та, $sqp()$ порівняти з результатами, отриманими в п. 3.

5. Реалізувати метод Ньютона-Рафсона другого порядку. Порівняти результат із іншими методами.

Питання для самоперевірки

1. Опишіть метод координатного спуску.
2. Опишіть метод градієнтного спуску з постійним кроком.
3. Опишіть метод градієнтного спуску з адаптивним кроком.
4. Опишіть метод найшвидшого спуску.
5. Опишіть метод Ньютона-Рафсона другого порядку? Як обчислити матрицю Гессе?

ДОДАТКИ
ДОДАТОК А (ДОВІДКОВИЙ). ІНТЕРФЕЙС *IDE GNU OCTAVE*

Графічний інтерфейс пакету *GNU Octave* (рис. А.1) складається із вікна з декількома робочими областями:

1 – текстовий редактор та середовище для налагодження програм;

2 – вікно для введення команд, тут же відображається результат роботи програми;

3 – історія команд, що забезпечує швидке виконання команди подвійним кліком миші;

4 – область змінних із їх фактичним значенням;

5 – файловий менеджер, де відображається вміст робочого каталогу.

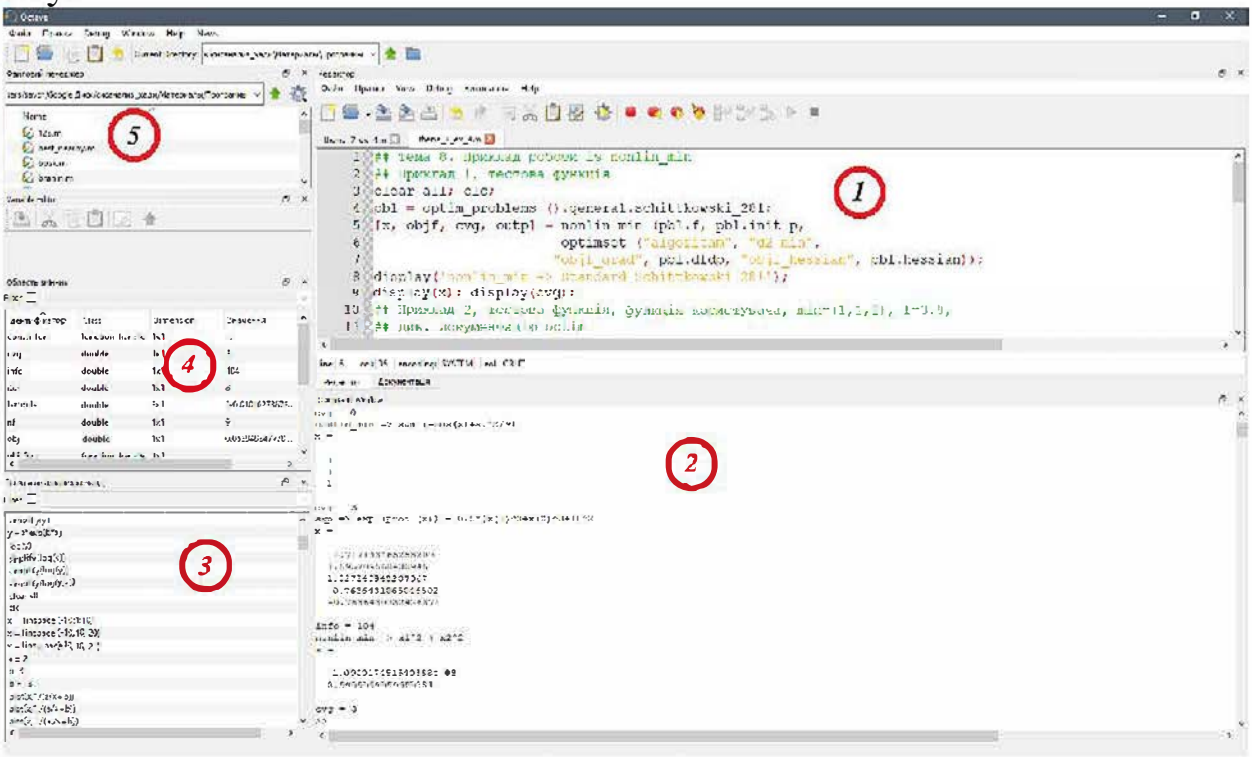


Рисунок А.1

Також є область змінних, в якій задають значення змінних в процесі налагодження, та вікно довідкової системи, яке на рис. А1 показане у вигляді закладки «Документація» під вікном редактора. Для активації довідкової системи достатньо у командному вікні набрати команду *doc* найменування_функції, париклад, *doc sqp*. Для деяких па-

кетів є свої довідкові системи. Так для функцій пакету *optim* використовують *optim_doc* найменування_функції_пакету_*optim*, наприклад, *optim_doc nrm*.

Крім того використовують інтерфейс на основі командного рядка (рис. А.2), але він, можливо, не такий зручний.

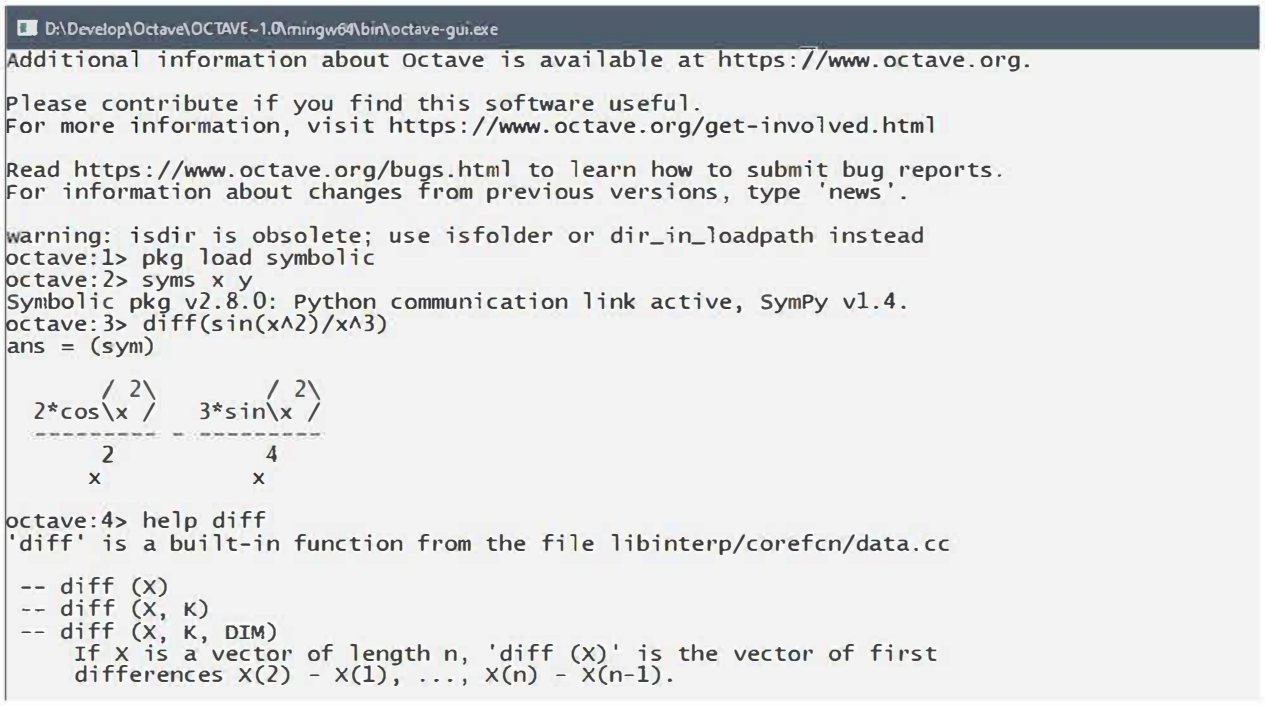


Рисунок А.2

В процесі розробки складних програм виникає необхідність їх налагодження, що може бути виконане засобами *IDE*. Для цього розставляють контрольні точки напроти потрібних рядків програми та виконують програму у покроковому режимі (рис. А.3, А.4).

Панель інструментів у режимі редагування та налагодження забезпечує наступні можливості:

- 1-5 – створення та зберігання файлів;
- 8-10 – робота із буфером обміну;
- 11 – пошук та заміна тексту;
- 12 – зберігання файлу та запуск на виконання;
- 13-16 – робота із точками останова (відповідно додати нову точку, перейти до наступної, перейти до попередньої, очистити всі);
- 17 – виконати рядок (крок);
- 18 – виконати із заходом у функцію (рис. А.3);
- 19 – виконати без заходу у функцію;

- 20 – продовжити до наступної точки останову;
- 21 – зупинити налагодження.

Для відслідковування за станом змінної потрібно виконати команду *openvar* найменування_змінної, після чого активується область «*Variable Editor*». В режимі налагодження дещо змінюється «Область змінних», у змінних з'являються атрибути, також можна включити підсвічування типів (рис. А.4).

Рисунок А.3

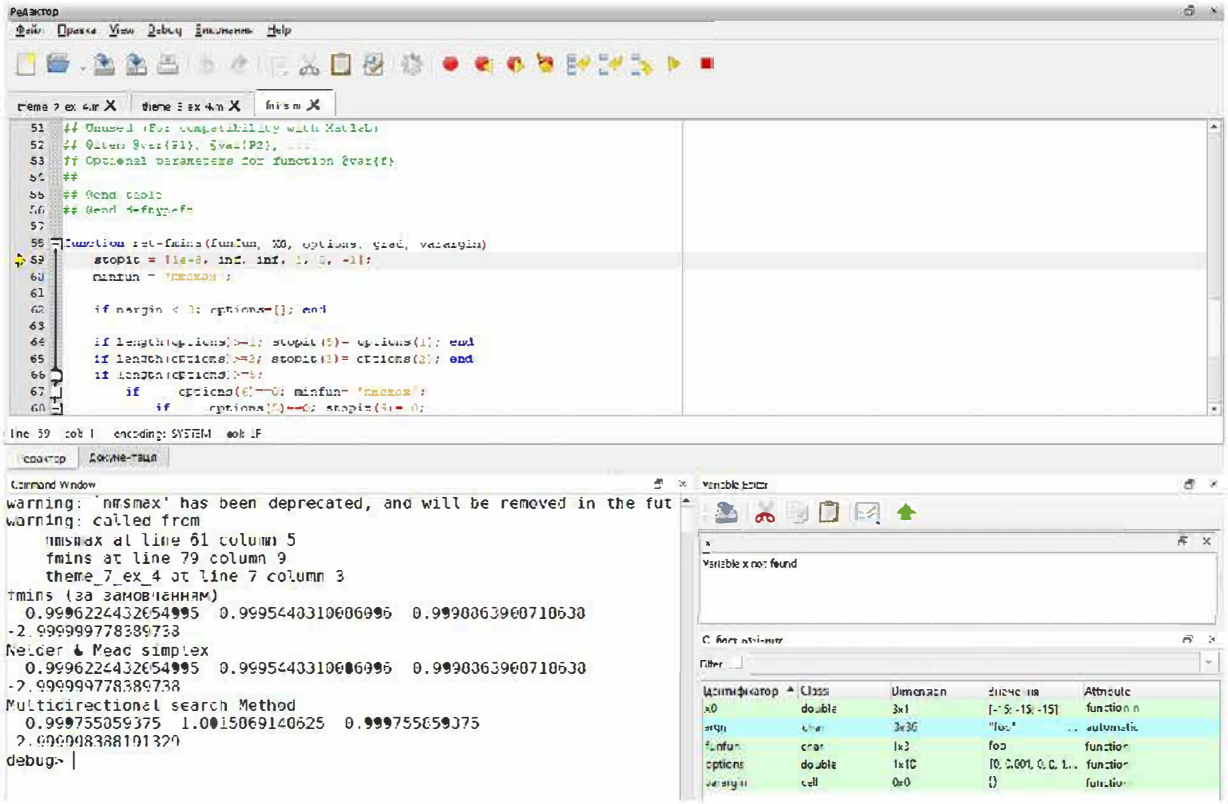


Рисунок А.4

Вікно з графіками (рис. А.5) має функції по керуванню їх зовнішнім виглядом (*Tools*), а також надає можливості з редагування (*Edit*) та зберігання файлу, але вони достатньо обмежені. На панелі

статусу можна спостерігати за значеннями функції під вказівником миші. Професійний вигляд графікам можна задати лише використовуючи програмний код, аналогічно тому як ми робили це в прикладах.

Зауважимо, що при зберіганні файлу з графіком вказують його розширення, що автоматичне призводить до вибору відповідного векторного або растрового формату файлів.

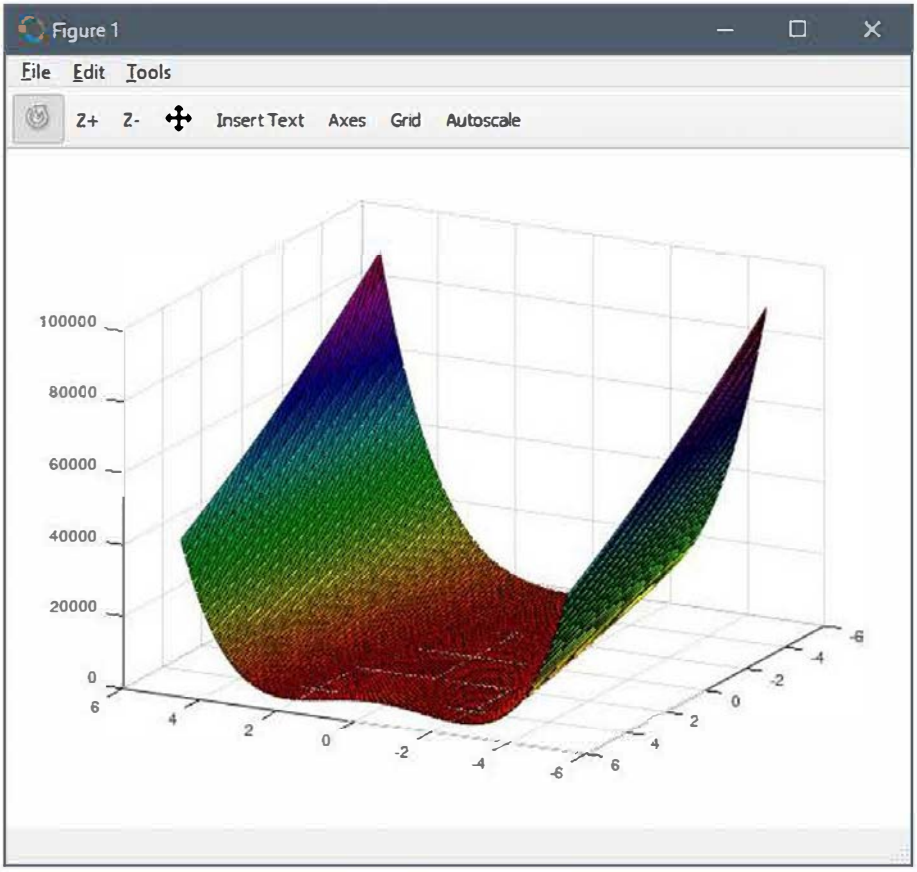


Рисунок А.5

При роботі із програмами часто виникає необхідність писати коментарі або текст, що буде виводитися на екран українською мовою. На жаль, такі надписи можуть неправильно відображатися, тому потрібно встановити кодування (*Text encoding used for loading and saving*) на *System* (рис. А.6), можна проекспериментувати з кодуванням *Unicode* (*UTF*-8), що використовують у *Linux*.

Від правильного вибору кодування залежить зовнішній вигляд результатів роботи пакетів розширення, таких як *symbolic* (див. рис. А.7, на якому верхній результат – консоль налаштовано правильно, нижній – ні).

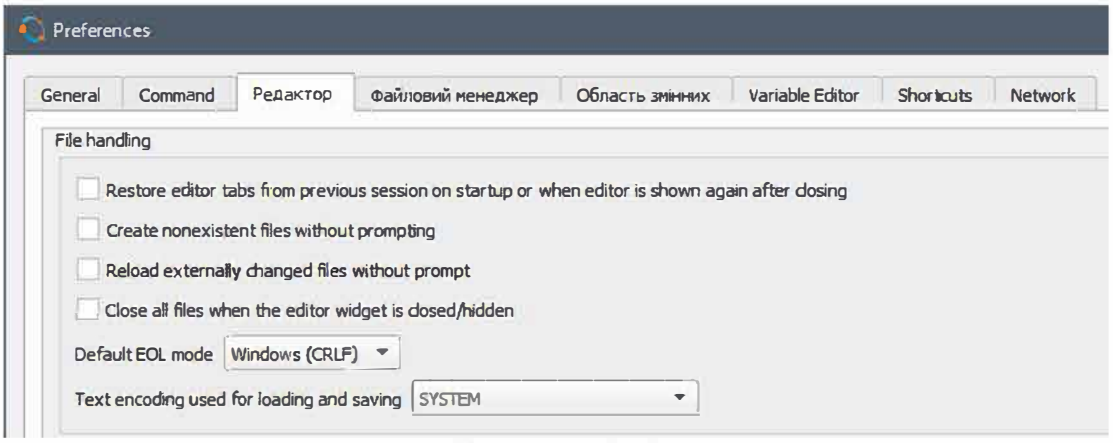


Рисунок А.6

```
octave:4> diff(sin(x^3/(2*tan(x^2))))
ans = (sym)
```

$$\left(-\frac{x^4\cdot\left(\tan^2\left(x^2\right)+1\right)}{\tan^2\left(x^2\right)}+\frac{3\cdot x^2}{2\cdot\tan\left(x^2\right)}\right)\cdot\cos\left(\frac{x^3}{2\cdot\tan\left(x^2\right)}\right)$$

```
octave:5> diff(sin(x^3/(2*tan(x^2))))
ans = (sym)
```

```
/   4 /   2/ 2\    \          2  \    /     3   \
|  x *\tan \x / + 1/      3*x   |    |    x    |
|- ----------------- + ---------|*cos|---------|
|          2/ 2\            / 2\|    |     / 2\|
\       tan \x /       2*tan\x //    \2*tan\x //
```

Рисунок. А.7

ДОДАТОК Б (ДОВІДКОВИЙ). СИСТЕМА ПАКЕТІВ *GNU OCTAVE*

GNU Octave – це мова програмування сумісна із мовою програмування *MathWorks MATLAB* на рівні функцій ядра системи. Основний прикладний функціонал *MathWorks MATLAB* зосереджений в додаткових пакетах (*toolboxes*), наприклад, *Control System Toolbox*.

Аналогічно для *GNU Octave* існує репозиторій пакетів *OctaveForge* (*https://Octave.sourceforge.io/* на момент створення посібника). В залежності від способу встановлення додаткових пакетів може не бути зовсім (через *flatpak*) або встановлений набір популярних пакетів (*Windows*).

Для керування пакетами використовують команду *pkg*. Розглянемо деякі її можливості.

1. Перелік встановлених пакетів *pkg list* (рис. Б.1)[22]

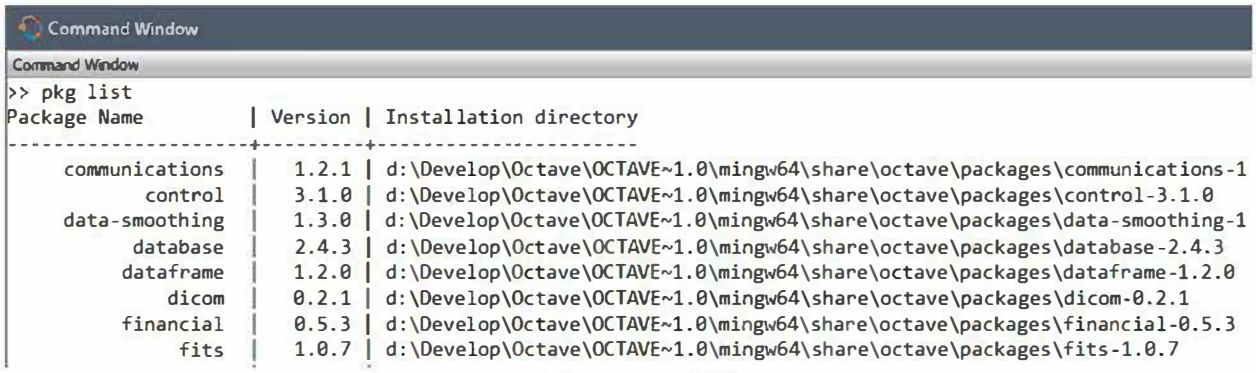


Рисунок Б.1

2. Встановлення пакету[23], ім'я (шлях до пакету) може бути посиланням на сайт, ключ *–forge* вказує на основний репозиторій *OctaveForge* *pkg install* [[−*local* | −*global* |] −*forge* | −*verbose*] ім'я_до_пакету

```
Octave:7> pkg install -forge optim
For information about changes from previous versions of the
optim package, run 'news optim'.
Octave:8> news optim
optim 1.6.0
```

[22] Може не відображатися, якщо пакет ставився із архіву, виконайте post-install.bat із папки Octave

[23] Часто потрібно вручну встановлювати залежності пакету

```
-----------
 ** Build fixes for Octave 5.1 and some bug fixes.
 ** With Octave from version 5.1 on, a parallel cluster
established
…
```

Для встановлення пакету *symbolic* у *Windows*[24] потрібно скачати відповідний пакет з сайту *Symbolic Package for GNU Octave*[25] та встановити його, наприклад, для релізу 2.8.0, це буде команда

```
pkg install symbolic-win-py-bundle-2.8.0.tar.gz
```

Пакет має буди у папці, яку вибрано у області «Провідник файлів» *IDE* або за допомогою відповідних команд. В *Linux* це робиться стандартним шляхом через основний репозиторій.

При роботі із посібником потрібен пакет *optim*, якщо він не встановлений, то потрібно виконати команду

```
pkg install -forge io statistics struct optim
```

за допомогою якої пакет буде встановлений із всіма необхідними залежностями.

3. Видалення пакету, перед видаленням перевіряються залежності від цього пакету і видається попередження *pkg uninstall* ім'я_пакету

4. Оновлення пакетів в залежності від кількості пакетів може зайняти певний час *pkg update* , при цьому, увага, пакети компілюються на вашому комп'ютері. Вихідний код пакетів автоматично скачується у вигляді архіва в каталог, що вибраний у області провідника *IDE Octave*.

5. *Підключення пакету* для роботи з ним *pkg load* ім'я_пакету

6. *Відключення пакету* для роботи з ним *pkg unload* ім'я_пакету

Для автоматичного завантаження в пам'ять потрібних пакетів додають рядки *pkg load ім'я пакету* у файл *.octaverc*, що знаходиться у домашньому каталозі користувача. Зауважимо, що використання додаткових пакетів може нести певний ризик безпеці комп'ютера, особливо, якщо встановлювати пакети невідомого походження.

---

[24] Це обумовлено необхідністю встановити інтерпретатор *Python* та *SymPy*

[25] https://github.com/cbm755/octsympy/releases

ДОДАТОК В (ДОВІДКОВИЙ). ВСТАНОВЛЕННЯ *GNU OCTAVE*

GNU Octave є вільним програмним забезпеченням, що підтримується співтовариством розробників з різних країн світу. Результати, що отримують за допомогою пакету, можуть бути легально використані в процесі наукових та інженерних розробок. В залежності від операційної системи є різні способи використання пакету – із встановленням та у вигляді портативної версії

Основним сайтом для скачування *GNU Octave* є сайт *Octave* на майданчику *GNU*[26], де є пакети для *GNU/Linux*, *macOS*, *BSD* та *Windows*, там же можна скачати вихідний код та зібрати пакет самостійно. На момент написання посібника актуальною є гілка 5.x.

Останнім версіям пакету властива однакова поведінка та універсальна *IDE* для різних операційних систем.

Для *Windows* рекомендується використовувати 64-розрядну версію (без різниці – портативну чи таку, що встановлюється). Її особливістю є те, що в неї інтегровано значну кількість пакетів розширення, за необхідності їх можна додавати (див. додаток Б). Є також деякі проблеми із *UTF-8* та інтерфейсом[27], немає темної теми. В цілому, всі приклади, що наведені в посібнику перевірялись саме на версії для *Windows* та виконувались без проблем. Ряд пакетів розширення вимагають встановленого *Python* визначеної версії, найпростішим шляхом є пошук в мережі версії пакетів із інтегрованим *Python*.

Для *Linux* традиційно підтримка пакета краще та його можна встановити за допомогою системного менеджера пакетів та отримати застарілу версію програмного забезпечення (*Ubuntu* 19).

У *Linux* все більше популярним стає використання віртуалізації програмних додатків на основі технології *flatpak* (контейнерів *LXC*). Спершу потрібно встановити пакет *flatpak* за допомогою менеджера пакетів, а потім вже встановити пакет *Octave*[28]

```
flatpak install flathub org.octave.Octave
```

---

[26] https://www.gnu.org/software/octave/download.html

[27] Іноді після першого старту програми немає рядка заголовка вікна, для його відтворення достатньо натиснути *Alt+Space* та розгорнути вікно

[28] https://flathub.org/apps/details/org.octave.Octave

Встановлення займає деякий час та вимагає безлімітного підключення до мережі Інтернет.

Для запуску виконати команду

$$flatpak\ run\ org.octave.Octave$$

Після запуску за допомогою *pkg* потрібно встановити необхідні пакети розширення, тому що їх немає в контейнері після установки.

Octave можна використовувати у популярному пакеті комп'ютерної алгебри *SageMath*[29], який розповсюджується у вигляді *Linux live-CD* та не вимагає встановлення.

Також можна використовувати он-лайнові сервіси – *Octave-online*[30], *tutorialspoint*[31] та інші.

На ресурсі *GNU Octave Wiki* [32] розташовану актуальну інформацію по використанню пакета, його встановлення в різних операційних системах та інші корисні посилання. Актуальна документація розташована на сайті *Octave.org*[33].

[29] *www.sagemath.org*

[30] *https://octave-online.net/*, після реєстрації доступна робота зі скриптами

[31] *https://www.tutorialspoint.com/execute_matlab_online.php*

[32] *https://wiki.octave.org/GNU_Octave_Wiki#Getting_help*

[33] *https://octave.org/doc/interpreter/*

ДОДАТОК Г (ДОВІДКОВИЙ). ОСНОВНІ ОПЕРАЦІЇ НАД МАТРИЦЯМИ У *GNU OCTAVE*

Приклад Г.1. Визначення матриць, операції зі структурою матриць[34]

```
## Приклад визначення матриць, операції зі структурою матриць
v_in_row = [1, 2, 3, 4, 5] % вектор-рядок
v_in_col = [1; 2; 3; 4; 5] % вектор-стовпець
A = [1 2 3 4; 0 2 6 5; 7 9 7 8]  % матриця
A( 2:3 , 1:2 )      % зріз матриці
display "Створення матриці з одиниць"
C = ones(3)         % 3x3
D = ones(3, 2)      % 3x2
E = ones(3, 3, 2)   % 3x3x2
display "Створення матриць з нулів"
C1 = zeros(2)       % 2x2
D2 = zeros(2, 3)    % 2x3
E2 = zeros(2, 2, 3) % 2x2x3
clear all;
display("Заповення матриці випадковими числами")
a = rand(4)
b = rand([3 2])
v = rand([4;1])
w = rand([1;4])
z = randi([-20 20],4)
## додавання елементів, рядків та стовпчиків
display('Додавання у матрицю стовпця')
a = zeros(3), a = [a, ones(3,1)]
b = zeros(3), b = [ones(3,1), b]
display("Додали не просто елемент, а цілий стовпчик")
b(1,5) = 7 %
display('Додавання у матрицю рядка')
a = zeros(3)
a = [a; ones(1,3)]
b = zeros(3)
```

[34] Для матриць «,» необов'язковий розділовий символ між елементами рідка, «;» – *обов'язковий* символ розділення матриці на рядки. Якщо «;» стоїть наприкінці рядка після визначення матриці – вона не буде відображатися на екрані в командному вікні

```
b = [ones(1,3); b]
display('Додали не просто елемент, а цілий рядок')
b(4,1) = 7 %
clear all
## видалення елементів
display("Вилучення елементів")
A=[5 5 5; 3 10 2; 2 8 4]
display("Видалення другого рядка")
A(2,:)=[]
display("Видалення третього стовпчика")
A=[5 5 5; 3 10 2; 2 8 4]
A(:,3)=[]
display("Видалення двох останніх елементіа")
A=[5 5 5; 3 10 2; 2 8 4]
A(2:end) = []
display("Видалення трьох елементіа")
A=[1 2 3 4 5 6 7 8 9]
A(4:6) = []
## спеціальні матриці
display ("Одинична діагональна матриця")
e = eye(3)
f = eye(4,5)
display ("Магічний квадрат")
m = magic(5)
q = eye(5)
display ("Функція від матриці")
display ("sin(m)")
sin(m)
display ("sqrt(m)")
sqrt(m)
display ("m + 3")
```

Приклад Г.2. Поелементні операції над матрицями

```
## Приклад. Поелементні операції над матрицями
clear all; A = ones(3), = randi([1,7], 3)
display("A.+B"); A.+B
display("A.-B"); A.-B
display("A.*B"); A.*B
display("A./B"); A./B
display("A.\\B"); A.\B
display("2*A.^B"); (2*A).^B
```

Приклад Г.3. Матричні операції

```
## Приклад. Матричні операції
clear all
A = ones(3), B = randi([1,7], 3)
C = [3  4  5], D = [2; 4; 5]
display("A+B"); A+B
display("A-B"); A-B
display("A*B"); A*B
##display("A*C");A*C
display("A*D"); A*D
display("A/B"); A/B
display("A.\\B"); A\B
display("A^3"); A^3
```

Приклад Г.4. Поширені функції для роботи із матрицями

```
## Приклад. Різні матричні функції
A = [1 2; 3 4; 5 6]
display("sum(A)"); sum(A)
display("sum(A,2)"); sum(A,2)
display("sum(A,1)"); sum(A,1)
display("sum(sum(A))"); sum(sum(A))
##
display("prod(A)"); prod(A)
display("prod(A,2)"); prod(A,2)
display("prod(A,1)"); prod(A,1)
display("prod(prod(A))"); prod(prod(A))
clear all; A=[1 2 3; 4 5 6; 7 8 9]
display("diag(A)"); diag(A)
display("diag(A, 1)"); diag(A, 1)
display("diag(A,-1)"); diag(A,-1)
display("rot90(A)"); rot90(A)
display("fliplr(A)"); fliplr(A)
clc, clear; d=1:12; size(d)
display(d); display("size(d)"); size(d)
display("reshape(d,3,4)"); d=reshape(d,3,4);
display(d); display("size(d)"); size(d)
display("reshape(d,4,[])"); d=reshape(d,4,[]);
display(d); display("size(d)"); size(d)
```

СПИСОК ЛІТЕРАТУРИ

1. Сорока К.О. Основи теорії систем і системного аналізу: навч. посіб. / Сорока К.О. – Харків, ХНАМГ : 2004. – 291 с.

2. Системний аналіз складних систем управління: навч. посіб. / [А.П. Ладанюк, Я.В. Смітюх, Л.О. Власенко та ін.] – К. : НУХТ, 2013. – 274 с.

3. Фельдман Л.П., Чисельні методи в інформатиці / Фельдман Л.П., Петренко А.І., Дмитріева О.А. – К.: Видавнича група *BHV*, 2006. – 480 с.

4. Нефьодов Ю.М. Методи оптимізації в прикладах і задачах : навч. посіб. / Ю.М. Нефьодов, Т.Ю. Балицька. – К. : Кондор, 2011. – 324 с.

5. Жалдак М. І. Основи теорії і методів оптимізації : навч. посіб. / М. І. Жалдак, Ю. В. Триус. – Черкаси : Брама-Україна, 2005. – 608 с.

6. Ляшенко Б.М. Методи обчислень: навч.-метод. посіб. / Ляшенко Б.М., Кривонос О.М., Вакалюк Т.А. – Житомир, 2014. – 228 с.

7. Бронштейн И. Н. Справочник по математике для инженеров и учащихся втузов / И. Н. Бронштейн, К. А. Семендяев. – М. : Наука, 1986. – 544 с.

8. Каханер Д. Численные методы и математическое обеспечение / Д. Каханер, К. Моулер, С. Нэш. – М. : Мир, 1998. – 575 с.

9. Лінійна алгебра та аналітична геометрія : навч. посіб. / [В. В. Булдигін, І. В. Алєксєєва, В. О. Гайдей та ін.]. – К., 2011. – 224 с.

10. Алексеев Е. Р. Введение в *Octave* для инженеров и математиков / Е. Р. Алексеев, О. В. Чеснокова. – М.: *ALT Linux*, 2012. – 368 с.

11. Мэтьюз Дж. Г. Численные методы. Использование *MATLAB* / Дж. Г. Мэтьюз, К. Д. Финк. – М. : Издат. дом «Вильямс», 2001. – 720 с.

12. Иглин С.П. Математические расчеты на базе *MATLAB* / Иглин С.П. – СПб.: БХВ-Петербург, 2005. – 640 с.

13. MATLAB Source Codes [Електронний ресурс]. – Режим *доступу : http://people.sc.fsu.edu/~jburkardt/m_src/m_src.html.*

14. Virtual Library of Simulation Experiments: Test Functions and Datasets [Електронний ресурс]. – Режим доступу : http://www.sfu.ca/~ssurjano/index.html.

15. Global Optimization Test Problems [Електронний ресурс]. – Режим доступу : http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm.

16. When the Solver Fails [Електронний ресурс]. – Режим доступу https://www.mathworks.com/help/optim/ug/when-the-solver-fails.html

17. Тихомиров В.Р. Рассказы о максимумах и минимумах / Тихомиров В.Р. – М.: МЦНМО, 2006. – 200 с.

Навчальне видання

Володимир Миколайович САВЧЕНКО
Ольга Борисівна МАЦИЙ
Оксана Василівна МНУШКА

СИСТЕМНИЙ АНАЛІЗ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ В GNU OCTAVE

Навчальний посібник

Відповідальний за випуск *О.Я. Ніконов*

В авторський редакції

ВИДАВНИЦТВО
Харківського національного автомобільно-дорожнього університету
Видавництво ХНАДУ, 61002, Харків-МСП, вул. Петровського, 25.
Тел. /факс: (057) 700-38-64; 707-37-03, e-mail: rio@khadi.kharkov.ua

Свідоцтво Державного комітету інформаційної політики, телебачення та радіомовлення України про внесення суб'єкта видавничої справи до Державного реєстру видавців, виготівників і розповсюджувачів видавничої продукції, серія № ДК №897 від 17.04.2002 р.

Підписано до друку 14.02.2020 р. Формат 60x84 1/16. Папір офсетний. Гарнітура Times New Roman Cyr. Віддруковано на ризографі. Умовн. друк. арк. 8,0. Обл.-вид. арк. 5,88. Замовлення № 14/02/20. Тираж 50 прим. Ціна договірна.

Віддруковано ФОП Гончаренко С.Ю.
Свідоцтво В02 № 247534 видане виконавчим комітетом Харківської міської ради 17.08.2007 р.