МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

Оптимізація систем керування
Комп’ютерний практикум

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Автоматизація, комп'ютерно-інтегровані технології»

спеціальності 174 (151) «Автоматизація, комп'ютерно- інтегровані технології та робототехніка»

Укладач:і : А. І. Жученко, Л. Р. Ладієва

Р. М. Дубік, Є. О. Тюріна

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025

УДК: 681.5:004.942](075.8)
О-62

Укладачі: *Жученко Анатолій Іванович*, д-р техн. наук, проф.,
Ладієва Леся Ростиславівна, канд. техн. наук, доц.,
Дубік Роман Миколайович, канд. техн. наук,
Тюріна Євгенія Олександрівна, доктор філософії

Рецензент Корнієнко Б. Я., д-р техн. наук, проф., *професор кафедри ІСТ*

Відповідальний редактор *Цапар В. С., канд. техн. наук, доц., зав.кафедрою ТПЗА*

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 5 від 06.03..2025р.)
за поданням вченої ради інженерно-хімічного факультету
(протокол № 2 від 24.02.2025 р.)

Жученко А. І., Ладієва Л. Р., Дубік Р. М., Тюріна Є. О.

Оптимізація систем керування. Комп'ютерний практикум : навч. посіб. для здобувачів ступеня бакалавр спец. 174 (151) «Автоматизація та комп'ютерно-інтегровані технології та робототехніка» / А. І. Жученко, Л. Р. Ладієва, Р. М. Дубік, Є. О. Тюріна**;** КПІ ім. Ігоря Сікорського.-.Київ: КПІ ім.. Ігоря Сікорського. 2025, - 183 с.

О-62

Навчальний посібник розрахований на одержання студентами практичних навичок з розв'язку оптимізаційних задач з використанням системи комп'ютерної математики MATLAB. Наведені багаточисельні приклади розв'язку задач оптимізації в режимі командного рядка, ілюстровані двовимірними і тривимірними графіками, а також завдання для восьми комп'ютерних практикумів.

Містить теоретичні положення, порядок виконання практичних робіт, завдання та список літератури з дисципліни «Оптимізація систем керування».

Для студентів всіх форм навчання, які навчаються за спеціальністю 174 (151) «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» інженерно-хімічного факультету КПІ ім. Ігоря Сікорського.

УДК: 681.5:004.942](075.8)

Реєстр. № НП 23/25-393 . Обсяг 7,6 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

Вступ

Для розрахунку оптимальних систем керування технологічними процесами в якості програмного забезпечення може використовуватись програмний продукт MATLAB. В данному посібнику розглядаються питання практичного застосування методів нелінійного програмування, генетичного алгоритму , варіаційного числення для розрахунку оптимальних систем керування.

Пакет Toolbox Optimization представляє собою розширення обчислюваного середовища MATLAB. Цей пакет включає в себе відомі методи мінімізації та максимізації лінійних та нелінійних функцій. Алгоритми представлені для розв'язку безперервних задач оптимізації без та при наявності обмежень, а також дискретних задач оптимізації.

Засоби MATLAB Optimization Toolbox дозволяють легко ставити задачі оптимізації, збирати необхідну інформація, коригувати формулювання задачі та аналізувати отримані результати.

Функції пакета Optimization, що створені та записані за допомогою мови програмування MATLAB дають можливість перевірки обраних алгоритмів, редагування вихідних кодів, створення власних функцій.

Цільова функція та обмеження задаються просто, і так само просто розв'язується поставлена задача. Вони можуть бути записані у вигляді М-файлу або виразу MATLAB.

В виданні надані короткі теоретичні відомості та приклади програм, наступних задач оптимізації: безумовна, оптимізація, оптимізація методом найменших квадратів, послідовно-квадратичне програмування, умовна оптимізація, багатокритеріальна оптимізація, генетичний алгоритм і прямий пошук. Використовуючи додаток можна змінювати вихідні та вхідні аргументи функцій, параметри опцій оптимізації (в залежності від постановки задачі та обраного алгоритму).

MATLAB (MATrix LABoratory) – одна з найбільш потужних систем компютерної графіки, що побудована на розширеному застосуванні матричних обчислень. Вона включає пакети розширення Simulink, Toolbox і Blockset, що

орієнтовані на різноманітні задачі , де потрібні математичні розрахунки, глибокий аналіз і моделювання.

В розширення Toolbox входять два пакети, що призначені для розвязку задач оптимізації: Toolbox Optimization и Toolbox Global Optimization. Данний посібник призначений опису цих пакетів. Перший пакет включає набір функцій, в яких реалізовані методи і алгоритми одновимірної і багатовимірної бузумовної і умовної оптимізації, лінійного і квадратичного програмування, найменьших квадратів, а також алгоритим багатокритеріальної оптимізації.

Функції пакету Global Optimization орієнтовані на пошку глобального мінімуму або багатьох мінімумів. Тут використовуються методи прямого пошуку, генетичний алгоритм. В цей пакет також включені функції багатокритеріального генетичного алгоритма. Для практичного освоєння і використання методів оптимізації, пропонується виконати вісім лабораторних робіт.

Огляд методів оптимізації

Методи оптимізації, зокрема нелінійного програмування використовуються для того, щоб знайти певний набір параметрів $X = \{x_1, x_2, ..., x_n\}$, які в деякому розумінні можуть бути визначені як оптимальні. У вузькому розумінні це може бути пошук мінімуму або максимуму деякої функції від $X = \{x_1, x_2, ..., x_n\}$. В більш широкому значенні це формулювання є мінімізацією або максимізацією цільової функції з обмеженнями

$G(X) = 0$ $i = 1, ..., m_e$ – рівності;

$G(X) \leq 0$ $i = m_e + 1, ..., m$ – нерівності.

Загальне формулювання задачі параметричної оптимізації представляється таким чином: слід знайти вектор $X = \{x_1, x_2, ..., x_n\}$, що забезпечує

$$\min_{x \in \mathcal{R}^n} f(X),$$

за обмежень

$G_i(X) = 0, i = 1, ..., m_e;$

$G(X) \leq 0$ $i = m_e + 1, ..., m;$

$x_i \leq X \leq x_u,$

де X – вектор параметрів, що оптимізуються ($X \in \mathcal{R}^n$) на межі зміни параметрів x_i, x_u, $f(x)$ – цільова функція векторного аргументу від X ($f(x)$: $\mathcal{R}^n \rightarrow \mathcal{R}$), $G(X)$ – також деякі функції векторного аргументу від X ($G(X)$: $\mathcal{R}^n \rightarrow \mathcal{R}^n$) (при цьому задача максимізації може зводитися до задачі мінімізації при заміні $f(x)$ на $-f(x)$).

Ефективність і точність розв'язків даної задачі залежать як від числа параметрів і обмежень, так і від вигляду цільової функції. При лінійних обмеженнях і лінійній цільовій функції приведена задача оптимізації називається задачею лінійного програмування (LP). Задача квадратичного програмування (QP) є мінімізацією або максимізацією квадратичної (по аргументах) цільової функції за наявності обмежень лінійного вигляду. Постановки задач типу (LP) і (QP) достатньо реалістично досяжними задачами. Складнішою є узагальнююча задача нелінійного програмування (NP), коли цільова функція і обмеження є деякими нелінійними функціями. NP, в загальному випадку розв'язується за допомогою ітераційних методів з корекцією напрямку пошуку на кожній

ітерації. Така постановка задачі звичайно розв'язується через рішення окремих проміжних задач LP і QP.

Комп'ютерний практикум № 1

1. Методи нелінійного програмування, що використовують похідні

Існуючі алгоритми оптимізації при відсутності обмежень можуть бути розділений на дві групи алгоритми, що базуються на використовуванні похідних функції (градієнтні і методи другого порядку), і алгоритми, що використовують тільки значення функції.

Методи, що використовують тільки значення функції (наприклад, метод Нелдера - Міда) більш придатні для задач, де функція, що мінімізується, є істотно нелінійною функцією або має розриви. Градієнтні методи (методи першого порядку) звичайно ефективні у випадках цільових функцій, неперервних разом з першими похідними. Методи другого порядку, такі як метод Ньютона, застосовуються рідше, оскільки вимагають великих обчислювальних витрат для розрахунку матриць других похідних.

Градієнтні методи використовують інформацію про нахил функції для вибору напряму пошуку екстремуму. В одному з таких методів - найшвидшого спуску на кожній ітерації рух до точки мінімуму здійснюється в напрямку $-\nabla f(X)$ (де $\nabla f(X)$ – вектор-градієнт цільової функції *f(X)*). Цей метод є неефективним в ситуаціях, коли поверхня цільової функції має вузькі "яри", як, наприклад, у відомої функції Розенброка

$$f(X) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \ . \qquad (1.1)$$

Мінімальне значення даної функції, як бачимо, рівне нулю *f(X)* = 0 при x=[1,1]. Графічне представлення ізоліній даної функції приведено на рис. 1.1, де також представлено траєкторію просування в напрямку до точки мінімуму згідно методу найшвидшого спуску з початкової точки [1.9, 2].

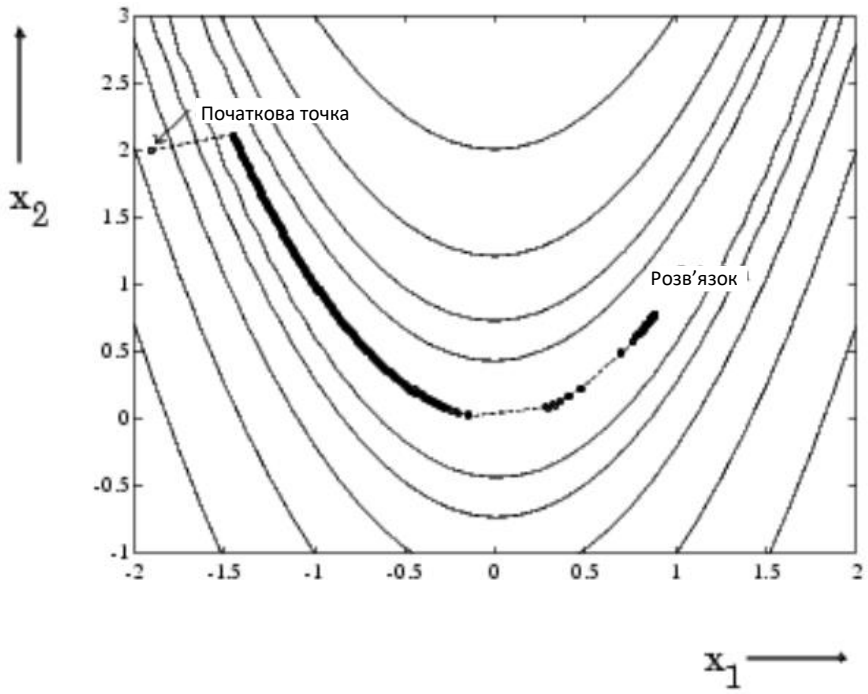


Рис. 1.1. Метод найшвидшого спуску для функції Розенброка

Оптимізація була зупинена після 1000 ітерацій, хоча все ще знаходилася на значній відстані від точки мінімуму. Пунктиром представлені області, де згідно даному методу проходить безперервний зигзагоподібний перехід з одного боку "яру" на інший. Відзначимо, що при напрямі до центру даного графіка число збільшених кроків наголошується у разі, коли вони лежать точно в центрі "яру". Цю функцію через своєрідну форму ліній рівня часто називають "банановою функцією" і використовують як тестову при перевірці різних методів оптимізації. Ізолінії представлені з експоненціальним приростом унаслідок різкої зміни нахилу для U-подібних ярів.

1.1. Квазіньютонівські методи

Серед алгоритмів, що використовують інформацію про градієнт, найпоширенішими є Квазіньютонівські. У цих (ітераційних) алгоритмах цільова функція в околі довільної точки апроксимується квадратичною функцією, при цьому на кожній ітерації вирішується задача локальної мінімізації форми

$$\min_x \frac{1}{2} X^T H X + C^T X + b, \tag{1.2}$$

де H – симетрична й достатньо визначена матриця других похідних (матриця Гессе, або Гессіан), C – постійний вектор, b – константа.

Оптимальний розв’язок наведеної задачі відповідає нульовим значенням перших похідних, тобто

$$-\nabla f(X^*) = HX^* + C = 0 \tag{1.3}$$

Точка оптимального розв’язку X^* може бути представлена як

$$X^* = H^{-1}C. \tag{1.4}$$

Ньютонівські методи й алгоритми (на відміну від квазіньютонівських) безпосередньо обчислюють H і здійснюють рух у розрахованому на черговій ітерації напрямку зменшення цільової функції до досягнення мінімума (з використанням методів одновимірного пошуку). У квазіньютонівських алгоритмах таке обчислення не виконується, а використається деяка апроксимація H.

Серед подібних алгоритмів одним з найбільш популярних і використовуваних у пакеті Optimization Toolbox є так званий BFGS алгоритм, що одержав свою назву за прізвищами авторів (Бройден-Флетчер-Голдфарб-Шанно [5], [11], [17], [24]), у якому апроксимація H виконується ітераційно по формулі

$$H_{k+1} = H_k + \frac{Q_k Q_k^T}{Q_k^T S_k} - \frac{H_k^T S_k^T S_k H_k}{S_k^T H_k S_k} \tag{1.5}$$

де

$S_k = X_{k+1} - X_k;$

$Q_k = \nabla f(X_{k+1}) - \nabla f(X_k);$

Як початкова точка H_0 може бути встановлена якась додатно - визначена матриця, наприклад, тотожна матриця I. Відзначимо, що більш зручно мати апроксимацію не матриці H, а оберненої до неї матриці H^{-1}, наведене рекурентне співвідношення подібну заміну допускає, при цьому сам алгоритм стає практична ідентичним добре відомому методу Девідона-Флетчера-Пауелла [8], [11], [13] за тим виключенням, що в останньому (1.5) вектори замінені на вектори S_k й навпаки.

Інформація про градієнт задається аналітично або розраховується чисельно за допомогою кінцевих різниць. Такий підхід містить у собі почерговий облік всіх напрямків вектора X, а також розрахунок швидкості зміни цільової функції.

Для кожної основної ітерації лінійний пошук здійснюється в напрямку:

$$D = -H_k^{-1}\nabla f(X_k). \qquad (1.6)$$

Застосування квазіньютонівського методу можна проілюструвати за допомогою аналізу шляху пошуку для функції Розенброка (1.1), як це представлено на рис. 1.2.

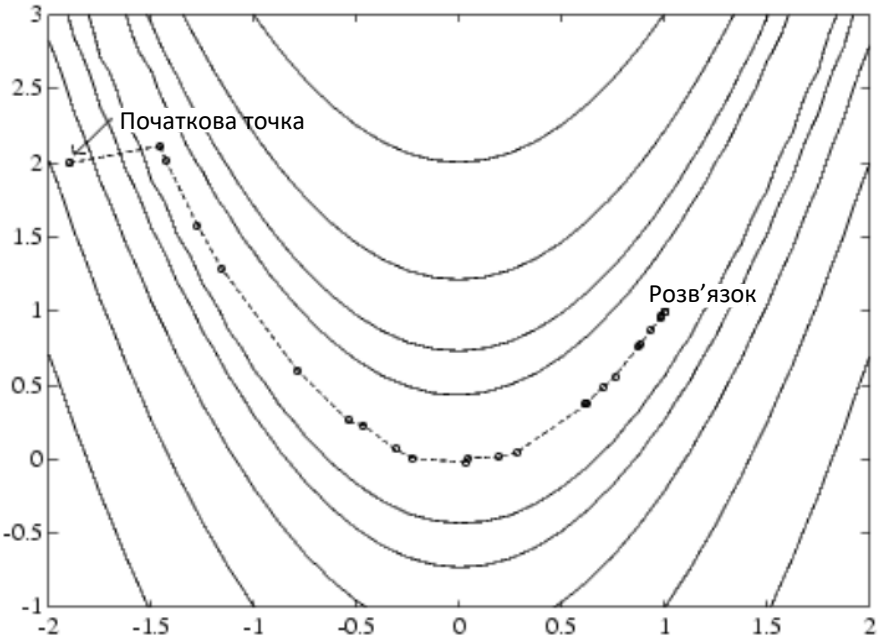


Рис. 1.2. Ілюстрація методу BFGS для функції Розенброка

1.2. Методи лінійного пошуку

Більшість методів пошуку, як у випадку наявності обмежень так і без них, засновані на розв'язку певної підзадачі, що визначає напрямок пошуку вбік шуканого рішення. Процес мінімізації серед напрямків такого пошуку як правило проводиться за допомогою процедури пошуку (наприклад Фібоначчі, Золотого перерізу) або за допомогою методу поліномів, що включає в себе інтерполяцію або екстраполяцію (наприклад квадратичну або кубічну). У методі поліномів проводиться апроксимація якогось числа точок для незмінного полінома, мінімум якого може бути легко обчислений. Інтерполяція

використовується у випадку, коли мінімум перебуває в інтервалі пошуку (тобто мінімум лежить в області, що охоплюється наявними точками), у той час як екстраполяція використовується у випадку, коли мінімум локалізований поза доступними точками. Методи екстраполяції як правило вважаються ненадійними у випадку оцінки мінімуму нелінійних функцій. Однак ці методи досить корисні для оцінки довжини кроку у випадку спроб локалізації мінімуму. Методи поліноміальної інтерполяції звичайно високо ефективні, коли функція, що підлягає інтерполяції є неперервною. Проблема полягає в тому, що б знайти нову ітерацію X_{k+1} у формі

$$X_{k+1} = X_k + \alpha * D_k, \tag{1.7}$$

де через X_k позначена поточна інтерполяція, D_k – знайдений якимось чином вектор відповідного напрямку пошуку, a^* - деякий скалярний параметр довжини кроку.

Метод лінійного пошуку зменшує цільову функцію вздовж $X_k + aD_k$ за допомогою повторної мінімізації поліноміальної інтерполяції цільової функції. Процедура лінійного пошуку має дві основні стадії:

- Визначається ряд точок $X_{k+1} = X_k + \alpha * D_k$, відповідно до інтервалу, що визначає діапазон значень a^*.
- Довжина кроку ділиться на під інтервали, на яких мінімум цільової функції наближається поліноміальною інтерполяцією.

Заключна довжина кроку a^* відповідає наступним умовам

$$\nabla f(X_k + aD_k) \leq f(X_k) + c_1 a f_k^T D_k; \tag{1.8}$$

$$\nabla f(X_k + aD_k)^T D_k \leq c_2 a f_k^T D_k; \tag{1.9}$$

де c_1 і c_2 константи, що задовільняють $0 < c_1 < c_2 < 1$.

Перша умова (1.8) вимагає щоб a^* давала достатнє зменшення цільової функції, а друга (1.9) гарантує, що довжина кроку не є занадто малою.

1.3. Реалізація квазіньютонівських методів та методів лінійного пошуку

Реалізація квазіньютонівських методів та методів лінійного пошук складається з двох частин:

- визначення напрямку пошуку (Коригування Гессіана);
- методи лінійного пошуку.

Коригування Гессіана

Напрямок пошуку визначається після вибору методу, розв'язку. Це може бути або метод BFGS (1.5) або метод DFP (при установці опційного параметра HessUpdate як 'dfp' вибирається метод DFP). Гессіан завжди, відповідно до визначення, підтримується додатнім, так що напрямок пошуку *D* завжди спрямовано по напрямку спуску. Це означає, що для когось довільного малого кроку a у напрямку *D*, відзначається зменшення по величині цільової функції. Після досягнення додатних значень *H*, відповідно до визначення матриці Гессе *H*, встановлюється й додатне значення величини $Q_k^T S_k$, згідно (1.15). Член $Q_k^T S_k$, є добутком параметра довжини кроку лінійного пошуку a_k й сполучення напрямку пошуку *D* з минулим або поточним розрахунковим значенням градієнта.

$$Q_k^T S_k = a_k(\nabla f(X_{k+1})^T D - \nabla f(X_k)^T D) \tag{1.10}$$

При реалізації досить точного лінійного пошуку завжди можна досягати певних додатних значень $Q_k^T S_k$. Оскільки напрямок пошуку завжди спрямовано у бік зменшення, а a_k та $-\nabla f(X_k)^T D$ також завжди будуть мати додатні значення. Таким чином, можливі від'ємні значення члена $\nabla f(X_{k+1})^T D$ можуть бути отримані тільки у випадку малих значень при відповідному зростанні точності лінійного пошуку.

Методи лінійного пошуку

Залежно від того чи є реальний доступ до інформації про градієнт або він повинен обчислюватися чисельно за допомогою кінцевих різниць, використовуються різні підходи. Коли інформація про градієнт є доступною, то за замовчуванням приймається кубічний поліноміальний метод. У випадку, коли інформація про градієнт не доступна, то за замовчуванням використається змішаний квадратичний і кубічний метод.

Кубічний поліноміальний метод

У поліноміальному методі значення градієнта й шуканої функції визначаються на кожній ітерації k. І на кожній ітерації відбувається деяке коригування, у випадку, якщо виявлено нову точку X_{k+1}, що задовольняє умові

$$f(X_{k+1}) < f(X_k) \tag{1.11}$$

Для кожної ітерації з використання кроку a_k проводиться спроба сформувати нову ітерацію у формі

$$X_{k+1} = X_k + a_k D \tag{1.12}$$

Якщо на даному кроці не виконується умова (1.11), то a_k зменшується й формується новий крок уже з a_{k+1}. Найпоширеніший спосіб такого зменшення - це безупинне ділення навпіл розміру кроку доти, поки таке зменшення не досягне $f(X)$. Однак, така процедура є досить повільною у порівнянні з підходом заснованим на використанні значень градієнта й значень функції разом з методом кубічної інтерполяції (екстраполяції) для оцінки значень величини кроку.

У випадку, якщо буде знайдена якась точка, що задовольняє умові (1.11), а член $Q_k^T S_k$ більше нуля, то виконується відповідне коригування. Якщо ця умова не виконується, то далі проводиться кубічна інтерполяція доти, поки значення одновимірного градієнта не буде досить малим, а величина $Q_k^T S_k$ стане додатньою.

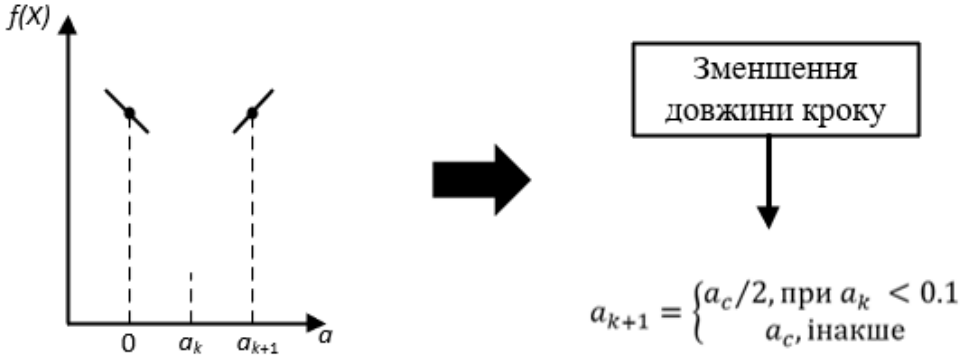
Як правило, після кожної ітерації значення a приймається рівним одиниці, Однак слід зазначити, що квадратична модель (1.2), як правило, добре працює тільки поблизу точки розв'язку. Тому, щоб компенсувати той випадок, коли апроксимація Гессіана монотонно зростає або спадає, то величина a_k зазнає модифікації на кожній основній ітерації. Для гарантії того, що як тільки значенея X_k наближається до точки розв'язку, й відповідно до алгоритма, значення відновлюються поблизу до одиниці, то значення $Q_k^T S_k - \nabla f(X_k)^T D$ й a_k використаються для оцінки близькості до точки розв'язку, й таким чином контролюють варіювання a_k.

Процедура кубічного поліноміального лінійного пошуку

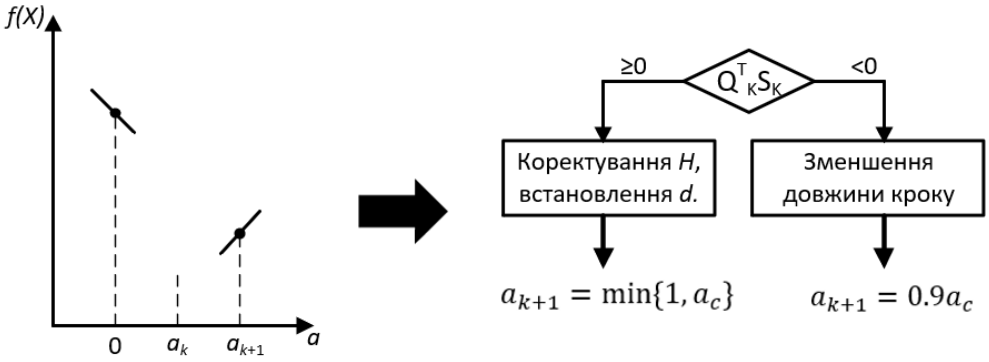
Відповідно до обраного сценарію, після кожної процедури коригування, у міру можливості розглядається значення величини кроку a_k. Подання всіх можливих варіантів досить складна задача, тому вони представлені нижче у вигляді графіків та блок-схем для кожного випадку:

- Ліва точка на графіку представляє дану точку X_k.
- Нахил лінії проекції кожної точки являє собою кутовий коефіцієнт одновимірного градієнта $\nabla f(X_k)^T D$, що завжди від'ємний для лівої точки.
- Права точка являє собою точку X_{k+1}, куди потрапляє після кроку a_k в напрямку D.

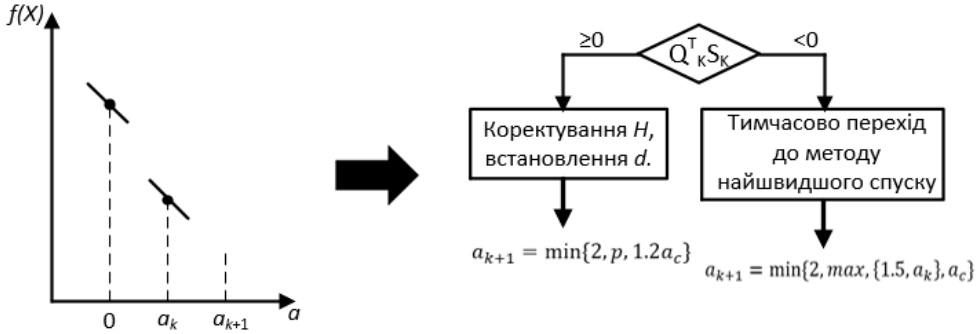
Варіант 1: $f(X_{k+1}) \leq f(X_k), \nabla f(X_{k+1})^T D \geq 0$



Варіант 2: $f(X_{k+1}) \leq f(X_k), \nabla f(X_{k+1})^T D \geq 0$

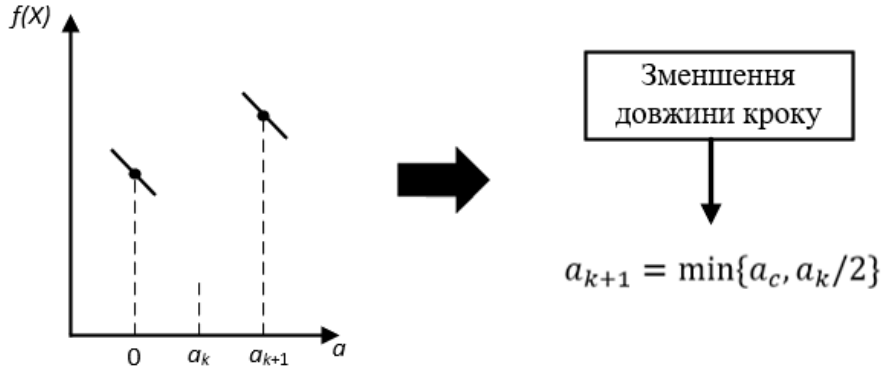


Варіант 3: $f(X_{k+1}) < f(X_k), \nabla f(X_{k+1})^T D < 0$



Варіант 4: $f(X_{k+1}) \geq f(X_k), \nabla f(X_{k+1})^T D \leq 0,$

Де $p = 1 + Q_k^T S_k - \nabla f(X_k)^T D + \min\{0, a_{k+1}\}$



На кожній ітерації розраховується розмір кроку кубічної інтерполяції a_c й потім ця величина використається для уточнення параметра довжини кроку a_{k+1}. Іноді, у випадку сильно нелінійних функцій, величина a_c може бути від'ємна й у цьому випадку a_c приймає значення рівне 2 a_k.

Крім того, у цьому випадку також є певні стабілізуючі міри, такі як, при введенні помилкової інформації про градієнт, зменшення значень функцій *f(X),* можна одержувати від'ємні значення величини кроку. Це може бути виправлено при встановленні a_{k+1} = - a_k /2 в ситуації, коли а стає меншим за якесь певне граничне значення (наприклад, 10^{-8}). У випадку застосування надмірно високих вимог до точності, такий підхід дає певні переваги, чим при використанні тільки кінцево-різницевих градієнтів.

Змішаний кубічний і квадратичний поліноміальний метод

Змішаний кубічний і квадратичний поліноміальний метод знайшов успішне застосування для ряду оптимізаційних задач. Однак, коли аналітичні значення градієнтів недоступні, то звичайно чисельна кінцево-різницева оцінка градієнтів приводить до істотних обчислювальних витрат. Тому введено такий новий інтерполяційний (екстраполяційний) метод, що на кожній ітерації немає необхідності у використанні значень градієнтів. При застосуванні такого підходу у випадку, коли градієнти недоступні, приводить до використання квадратичиної інтерполяції. Такий мінімум звичайно знаходиться за допомогою певних видів бієктивних методів. Однак такому методу про досліджувану функцію властиві недоліки такі, як невикористання всієї недоступної інформації. Наприклад, для

коригування матриці Гессе, на кожній ітерації завжди використають дані розрахунку градієнта. Тому, якщо враховувати три охоплюючи градієнтами точки, то можливо вже використати кубічну інтерполяцію, що ймовірно дасть більш точні результати, ніж квадратична інтерполяція. Отже, у випадку використання кубічної поліноміальної екстраполяції можливо досягти більшої ефективності в порівнянні з бієктивним способом знаходженням мінімуму.

Використовуваний у програмах *fminunc, Isqnonlin* метод оптимізації повинен використовувати три точки для охоплення мінімуму функції, а також кубічну інтерполяцію для пошуку мінімуму на кожному етапі лінійного пошуку. Така процедура оцінки величини кроку на кожній внутрішній ітерації по *j*, наведена нижче на графіках для ряду сполучень аналізованих точок. Сама ліва точка на графіку відображає одержувані після останнього коригування значення функцій $f(X_j)$ й одновимірного градієнта $\nabla f(X_k)$ точки, що залишилися, відображають точки після застосування внутрішніх ітерацій у методі лінійного пошуку.

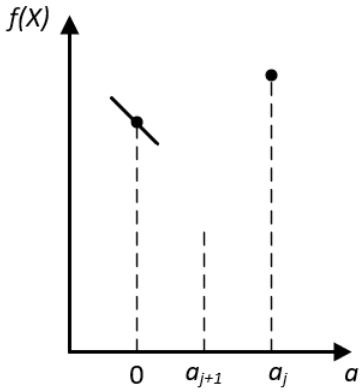
Члени a_q й a_c й прирівнюються до мінімальних значень, отриманих з відповідних методів квадратичної й кубічної інтерполяції або екстраполяції. Для дуже нелінійних функцій значення a_q і a_c, можуть бути від'ємними. В цьому випадку вони встановлюються як 2 a_k . Таким чином, підтримується їхнє додатне значення. У варіантах 1 й 2 використовується квадратична інтерполяція для двох точок та одне значення градієнта для розрахунку третьої точки з метою охоплення мінімуму. У випадку невдачі такого підходу варіанти 3 й 4 представляють можливості для зміни розміру кроку в ситуації коли доступні, принаймні, три точки.

Коли мінімум остаточно охоплений, то застосовується кубічна інтерполяція з обліку одного значення градієнта й трьох значень функції. Якщо інтерпольована точка має значення величини більше, ніж величини для трьох використовуваних у даній інтерполяції точок, то ця точка заміняється точкою з найменшим значенням функції. Дотримуючись методу лінійного пошуку,

процедура коригування матриці Гессе проводиться на основі кубічного поліноміального методу лінійного пошуку.

Графіки і блок-схеми, що нижче приведені, відображають процедуру лінійного пошуку для варіантів з 1 по 4 при наявності якогось градієнта для першої точки.

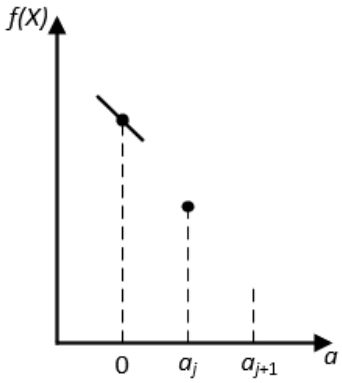
Варіант 1: $f(X_j) \geq f(X_k)$



Зменшення довжини кроку

$$a_{j+1} = a_q$$

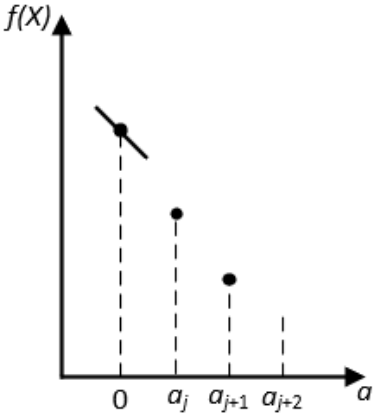
Варіант 2: $f(X_j) < f(X_k)$



Збільшення довжини кроку

$$a_{j+1} = 1.2a_q$$

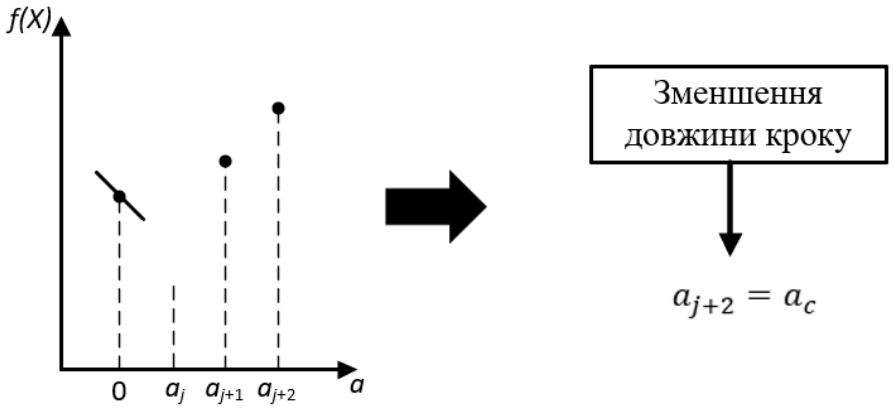
Варіант 3: $f(X_{j+1}) < f(X_k)$



Збільшення довжини кроку

$$a_{j+2} = \max\{1.2\ a_q, 2\ a_{j+1}\}$$

Варіант 4: $f(X_{j+1}) > f(X_k)$



1.4. Приклади програм

Метод найшвидшого спуску

Мінімізувати функцію $f(X) = th(x_1^2 + x_2^2)$, починаючи з точки $X^{(0)} = [1\ -1]^T$.

Запишемо приклад програми, що реалізує цей метод з назвою "NS".

```
function NS
% ведення функції
f = @(x) tanh(x(1)^2 + x(2)^2);
% Початкова точка
x0 = [1; -1];
%параметри опцій оптимізації
options =  optimset('Largescale', 'off',
'HessUpdate', 'steepdesc', 'Display', 'iter');
%Виклик функції оптимізації
[x, fval] = fminunc (f,x0, options)
 x % точка мінімуму
fval % значення функції в точці мінімуму
% Введення функції для побудови поверхні
f = @(x1, x2) tanh(x1.^2 + x2.^2);
% Побудова поверхні функції
ezmeshc(f);
```

Результат виконання програми

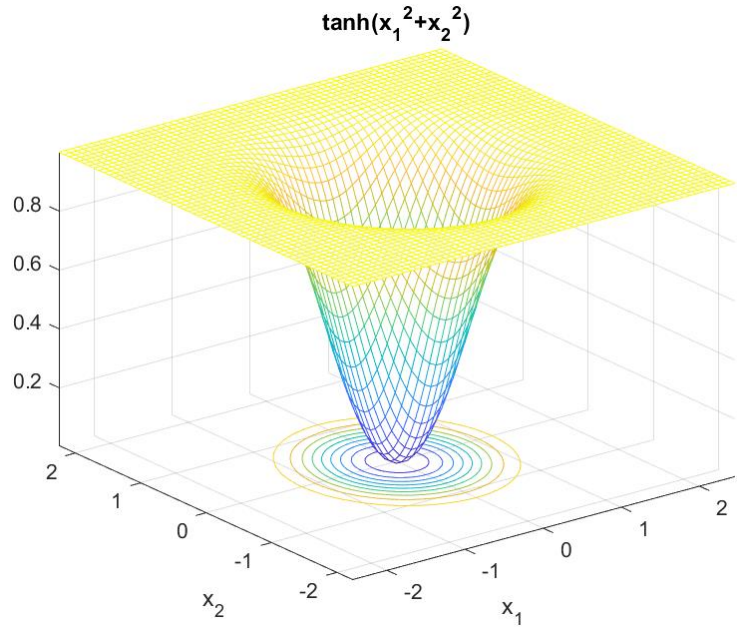
Iteration	Func-count	f(x)	Step-size	First-order optimality
0	3	0.964028		0.141
1	15	7.53827e-05	7.12051	0.0123
2	21	3.37043e-27	0.499999	1.49e-08

Local minimum found.

Optimization completed because the size of the gradient is less than the value of the optimality tolerance.

<stopping criteria details>

```
x =

   1.0e-13 *

   -0.4105
    0.4105


fval =

   3.3704e-27
```



Метод квадратичної та кубічної інтерполяції

Мінімізувати функцію $f(X) = \sin(x_1)\cos(x_2)$, починаючи з точки $X^{(0)} = [1\ -1]^T$.

Запишемо приклад програми, що реалізує цей метод з назвою *"CCI"*.

```
function CCI
% Введення функції
f = @(x) sin(x(1))*cos(x(2));
% початкова точка
x0 = [-1; 1];
% параметри опцій оптимізації
options = optimset('LargeScale','off', 'Display',
'iter');
% Виклик функції оптимізації
[x, fval] = fminunc(f, x0, options);
% точка мінімуму
disp('Точка мінімуму:');
disp(x);
% значення функції в точці мінімуму
disp('Значення функції в точці мінімуму:');
disp(fval);
```

```
% Введення функції для побудови поверхні
f = @(x1, x2) sin(x1).*cos(x2);
% Побудова поверхні функції
ezmeshc(f)
```

Результат виконання програми

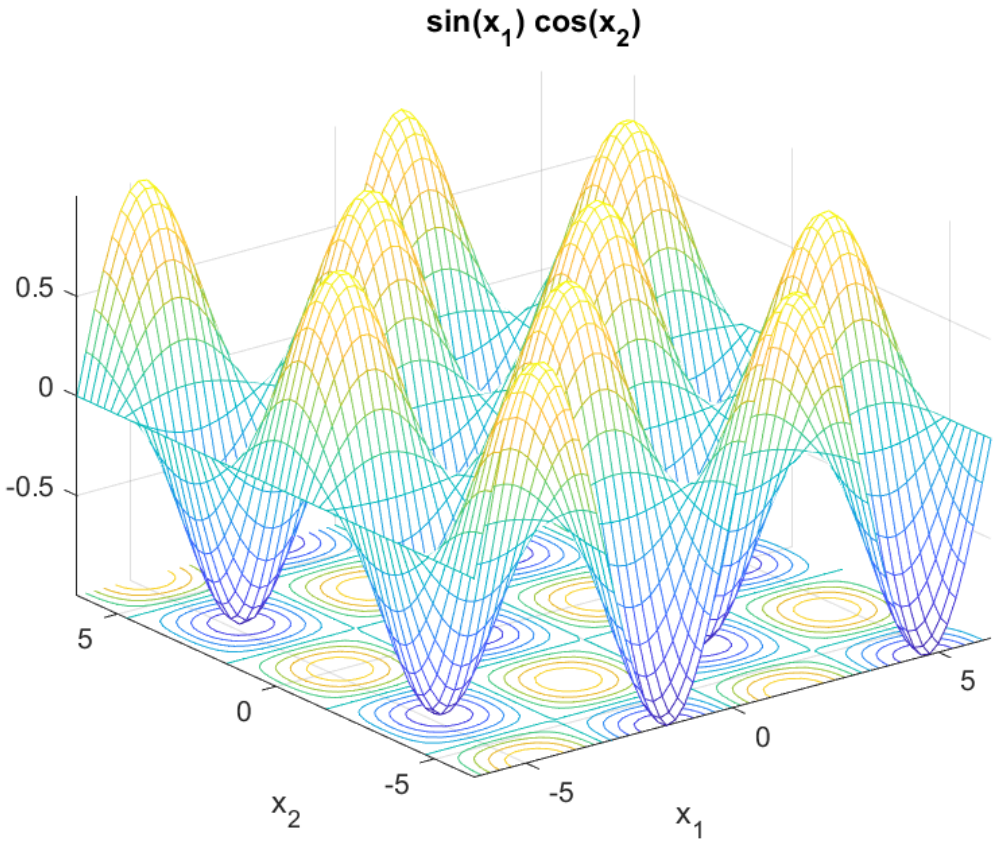
Iteration	Func-count	f(x)	Step-size	First-order optimality
0	3	-0.454649		0.708
1	6	-0.920693	1	0.277
2	12	-0.998843	0.430609	0.0396
3	15	-0.999902	1	0.0112
4	18	-1	1	2.85e-05
5	21	-1	1	1.35e-06

Local minimum found.

Optimization completed because the size of the gradient is less than the value of the optimality tolerance.

<stopping criteria details>
Точка мінімуму:
   -1.5708
    0.0000

Значення функції в точці мінімуму:
   -1.0000



Метод кубічної інтерполяції

Мінімізувати функцію $f(x) = \sin\left(\sqrt{x_1^2 + y_2^2}\right)$, починаючи з точки

$X^{(0)} = [-2 \quad 1]^T$.

Запишемо приклад програми, що реалізує цей метод з назвою «*CI*».

```
function CI
% Введення функції
f = @(x) sin(sqrt(x(1)^2+x(2)^2));
% початкова точка
x0 =  [-2; 1];
% Параметри опцій оптимізації
options= optimset('LargeScale','off', 'Algorithm',
'trust-region', 'Display', 'iter');
%Виклик функції оптимізації
[x, fval] = fminunc(f, x0, options);
% точка мінімуму
disp('Точка мінімуму:');
disp(x);
% значення функції в точці мінімуму
disp('Значення функції в точці мінімуму:');
disp(fval);
% введення функції для побудови поверхні
f = @(x1, x2) sin(sqrt(x1.^2+x2.^2));
% побудова поверхні функції
ezmeshc(f);
```

Результат виконання програми

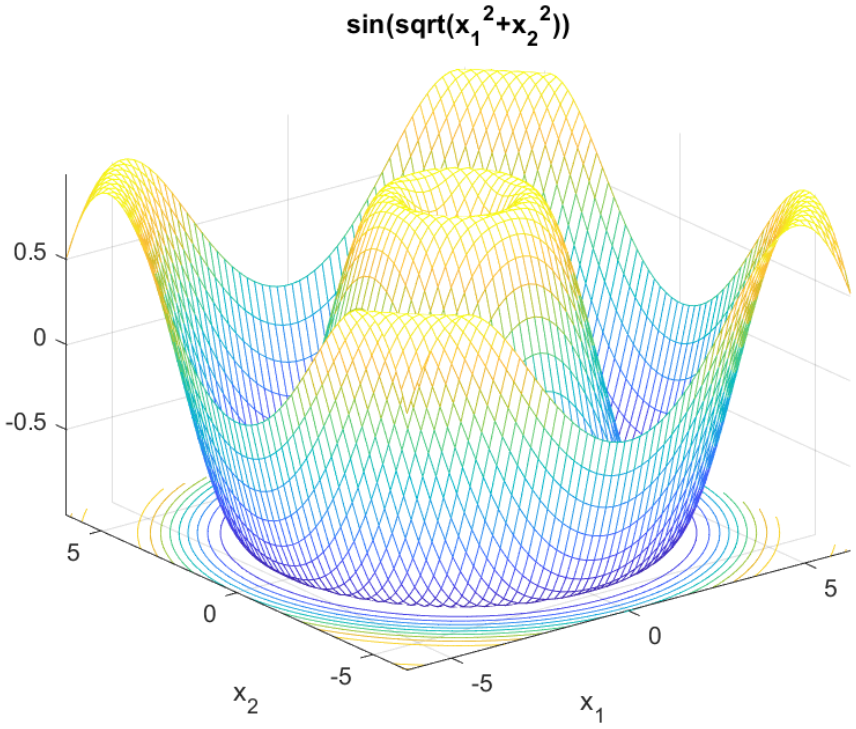
Iteration	Func-count	f(x)	Step-size	First-order optimality
0	3	0.786749		0.552
1	9	-0.99674	4.14257	0.0722
2	15	-1	0.272593	0.000173
3	18	-1	1	1.84e-07

Local minimum found.

Optimization completed because the size of the gradient is less than the value of the optimality tolerance.

<stopping criteria details>

```
Точка мінімуму:
   -4.2149
    2.1074

Значення функції в точці мінімуму:
   -1.0000
```



Метод Девідона-Флетчера-Паулера

Мінімізувати функцію $f(X) = \exp[-(x_1 - 1)^2] \cdot \frac{(x_2^2 - 0{,}5)^2}{0{,}132}$, починаючи з точки $X^{(0)} = [1 \quad 1]^T$.

Запишемо приклад програми, що реалізує цей метод з назвою «DFP».

```
function DFP
% Введення функції
f = @(x) exp(-(x(1)-1)^2)*((x(2)^2-1/2)^2)/0.132;
% Початкова точка
x0 = [1; 1];
% параметри опцій оптимізації
options = optimset('Largescale', 'off', 'Hessupdate',
'dfp', 'Display', 'iter');
% Виклик функції оптимізації
[x, fval] = fminunc(f, x0, options)
% x - точка мінімуму
% fval - Значення функції в точці мінімуму
% Введення функції для побудови поверхні
f = @(x1, x2) exp(-(x1-1).^2).*((x2.^2-
1/2).^2)/0.132;
% Побудова поверхні функції
ezmeshc(f);
```

Результат виконання програми

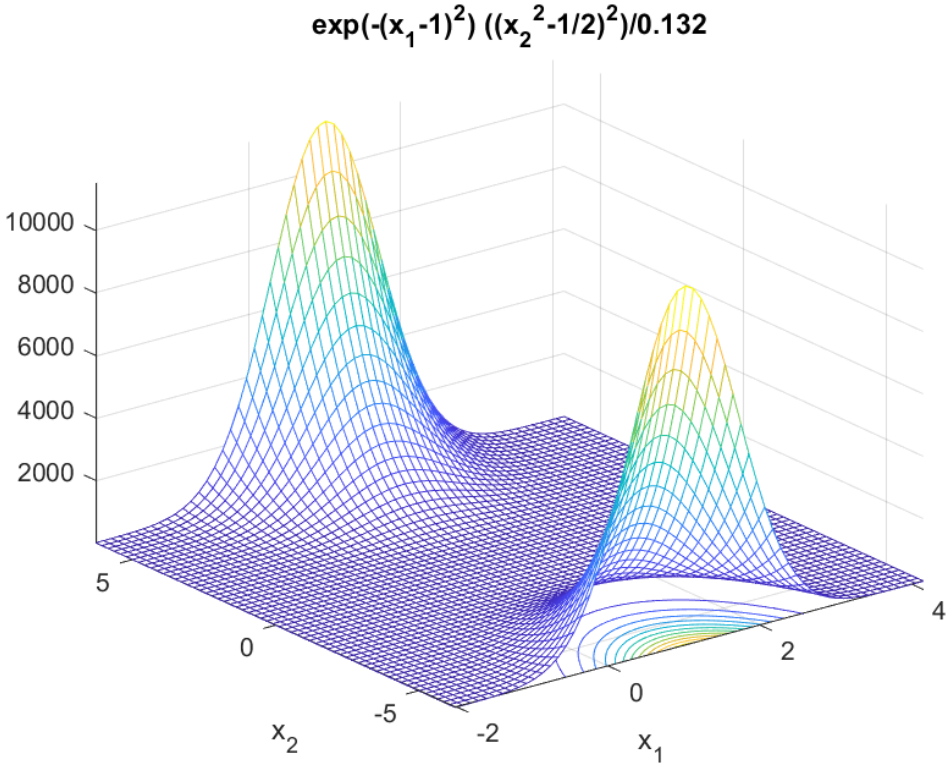
Iteration	Func-count	f(x)	Step-size	First-order optimality
0	3	1.89394		15.2
1	9	0.023382	0.022	1.12
2	12	0.00450134	1	0.509
3	15	4.16041e-05	1	0.0503
4	18	5.997e-08	1	0.00191
5	21	7.8589e-13	1	6.68e-06

Local minimum found.

Optimization completed because the size of the gradient is less than the value of the optimality tolerance.

<stopping criteria details>

```
x =

    1.0000
    0.7071


fval =

   7.8589e-13
```



Метод Бройдена-Флетчера-Голдфарба-Шанно

Мінімізувати функцію $f(X) = 100\,(x_2 - x_1^2)^2 + (1 - x_1)^2$, починаючи з точки $X^{(0)} = [-1.8 \quad 2]^T$.

Запишемо приклад програми, що реалізує цей метод з назвою “*BFGS*”.

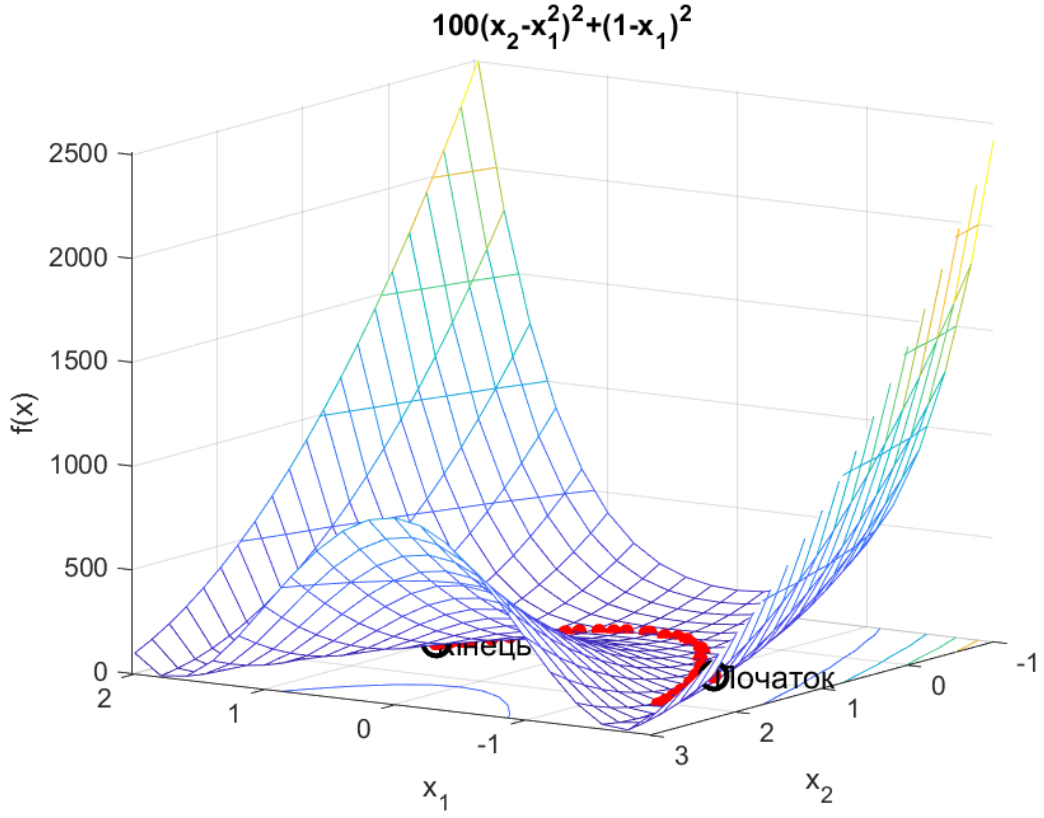
```
function BFGS
    % Введення функції
    f = @(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
    % Початкова точка
    x0 = [-1.8; 2];
    % параметри опцій оптимізації
    options = optimset('LargeScale', 'off',
'HessUpdate', 'bfgs', 'Display', 'iter', 'TolX', 1e-
10, 'TolFun', 1e-10, 'OutputFcn', @OutFcn);
    % Виклик функції оптимізації
    [x, fval] = fminunc(f, x0, options);
    % точка мінімуму
    disp('Точка мінімуму:');
    disp(x);
    % значення функції в точці мінімуму
    disp('Значення функції в точці мінімуму:');
    disp(fval);
    % виведення додаткової інформації
    text(x0(1), x0(2), f(x0), 'Початок', 'Color', [0
0 0], 'FontSize', 12);
    text(x(1), x(2), f(x), 'Кінець', 'Color', [0 0
0], 'FontSize', 12);
    plot3(x0(1), x0(2), f(x0), 'ko', 'Markersize',
10, 'Linewidth', 2);
    hold on;
    plot3(x(1), x(2), f(x), 'ko', 'Markersize', 10,
'Linewidth', 2);
    title('100(x_2-x_1^2)^2+(1-x_1)^2');
xlabel('x_1'); ylabel('x_2'); zlabel('f(x)');
end

function stop = OutFcn(x, optimvalues, state)
    % Виклик функції виводу
    stop = false;
    PlotIter(x);
    PlotObjFun(x);
end

function PlotIter(x)
    % побудова ітерацій
    x1 = x(1); x2 = x(2);
    z1 = 100*(x2-x1.^2).^2+(1-x1)^2;
```

```
    plot3(x1, x2, z1, 'r.', 'Markersize', 20);
    drawnow;
end

function PlotObjFun(x)
    % Побудова цільової функції
    x1 = -2:0.2:2; x2 = -1:0.2:3;
    [x1, x2] = meshgrid(x1, x2);
    Z = 100*(x2-x1.^2).^2+(1-x1).^2;
    hold on;
    meshc(x1, x2, Z);
    contour3(x1, x2, Z);
    grid on;
end
```

Результат виконання програми

Iteration	Func-count	f(x)	Step-size	First-order optimality
0	3	161.6		898
1	9	6.02894	0.00040153	11.3
2	12	5.98871	1	2.6
3	15	5.98438	1	1.44
4	18	5.98311	1	1.27
5	21	5.96756	1	5.22
6	24	5.93888	1	9.7
7	27	5.85918	1	17.9
8	30	5.72409	1	27.4
9	33	5.52973	1	32.8
10	36	5.11743	1	32.7
11	39	4.38664	1	19.9
12	42	3.78985	1	16
13	45	3.09265	1	1.66
14	54	2.82651	0.105962	12.7
15	57	2.58194	1	12.2
16	60	1.98204	1	4.49
17	66	1.85195	0.35732	7.57
18	69	1.56088	1	6.48
19	72	1.24729	1	1.87
20	78	1.04026	0.364169	6.61
21	81	0.928042	1	3.59
22	87	0.753244	0.627481	6.31
23	90	0.566658	1	2.36
24	96	0.486812	0.5	4.8
25	99	0.343686	1	3.58
26	102	0.255064	1	2.17
27	108	0.17791	0.399517	3.22
28	111	0.146111	1	2.32
29	114	0.0980946	1	4.75
30	117	0.0729807	1	6.25
31	123	0.0151018	0.676077	0.842
32	126	0.0134587	1	2.54
33	129	0.00817279	1	1.24
34	132	0.0022829	1	0.522
35	135	0.000632444	1	0.572
36	138	7.57183e-05	1	0.165
37	141	1.90964e-05	1	0.159
38	144	3.32447e-07	1	0.00111
39	147	1.47204e-08	1	0.00184

```
                                                          First-order
 Iteration  Func-count       f(x)         Step-size         optimality
    40          150      4.25337e-11             1         0.000173
    41          153      1.93593e-11             1          8.4e-06

Local minimum possible.

fminunc stopped because it cannot decrease the objective function
along the current search direction.

<stopping criteria details>
Точка мінімуму:
    1.0000
    1.0000

Значення функції в точці мінімуму:
   1.9359e-11
```



Завдання:

1. Мінімізувати такі функції двох незалежних змінних методом найшвидшого спуску:

а) $f(x)=\frac{1}{2}\left\{1-\cos 360\left[(2X_1-1)^2+(2X_2-1)^2\right]^{\frac{1}{2}}\left[1-\frac{(X_2-3X_1)^2}{8}\right]\right\}$

б) $f(x)=X_2+\sin X_1;$

в) $f(X)=-(X_2-X_1)^4+(1-X_1)^2;\quad f(X)=-X_1^2+X_1-X_2^2+X_2+4;$

г) $f(X)=\exp\left[-(X_1-1)^2\right]-\dfrac{(X_2^2-0.5)^2}{0.132}$;

2. Мінімізувати алгоритмом **Девідона-Флетчера-Паулера** такі цільові функції:

а) $f(X)=100(X_2-X_1^2)^2+(1-X_1)^2,\quad X^{(0)}=[-0{,}5\quad 0{,}5]^T$;

б) ‡) $f(X)=4(X_1-5)^2+(X_2-6)^2,\quad X^{(0)}=[8\quad 9]^T$;

в) $f(X)=X_1^2+X_2^2-4,\quad X^{(0)}=[4\quad 4]^T$.

3. Мінімізувати алгоритмом **Бройдена-Флетчера-Голдфарба-Шанно** такі цільові функції:

а) $f(X)=\left\{12+X_1^2+\dfrac{1+X_2^2}{X_1^2}+\dfrac{X_1^2X_2^2+100}{(X_1X_4)^4}\right\}\dfrac{1}{10},\quad X^{(0)}=[0{,}5\quad 0{,}5]^T$;

б) $f(X)=100(X_2-X_1^2)^2+(1-X_1)^2+90(X_4-X_3^2)^2+(1-X_3)^2+$
$+10{,}1\left[(X_2-1)^2+(X_4-1)\right]+19{,}8(X_2-1)(X_4-1),\quad X^{(0)}=[-3\quad -1\quad -3\quad -1]^T$.

Завдання для самостійної роботи:

4. Мінімізувати алгоритмом кубічної інтерполяції такі цільові функції:

а) $f(X)=X_1^3\exp\left[X_2-X_1^2-10(X_1-X_2)^2\right]$;

б) $f(X) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_1^2 - 7)$.

Комп’ютерний практикум № 2

2. Нелінійне програмування з використанням методу найменших квадратів

Використовувані в сполученні із квазіньютонівським методом процедури лінійного пошуку одержали своє застосування в програмі *fminunc*. Ці методи також використовуються й у підпрограмах нелінійної оптимізації методом найменших квадратів, а саме в *Isqnonlin*. У задачах на метод найменших квадратів, що підлягає мінімізації функція *f(X)* являє собою суму квадратів.

$$\min_{x\in\Re} f(X) = \tfrac{1}{2}\|F(X)\|_2^2 = \tfrac{1}{2}\textstyle\sum_i F_i(X)^2\ . \qquad (2.1)$$

Подібного типу задачі широко поширені й мають ряд практичних застосувань, особливо при підборі модельної функції для когось набору даних, тобто підбор нелінійних параметрів. Ці задачі також широко похширені в теорії керування, де в остаточному підсумку необхідно одержати якусь (Х,г), що

відповідає деякій безперервній модельній траєкторії (1) для вектора X і скаляра 4. Дана задача може бути сформульована як:

$$\min_{x\in\Re} \int_{i_2}^{i_1} \big(y(X,t) - \varphi(t)\big)^2 dt. \tag{2.2}$$

При дискретизації інтеграла рівняння (2.2) може бути сформульоване як задача на метод найменших квадратів

$$\min_{x\in\Re} f(X) = \sum_{i=1}^{m} \big(\tilde{y}(X,t_i) - \tilde{\varphi}(t_i)\big)^2, \tag{2.3}$$

де – $\tilde{y}$ і $\tilde{\varphi}$ містять у собі ваги квадратичної схеми. Відзначимо, що в даній задачі від вектором *F(x)* розуміється:

$$F(X) = \begin{bmatrix} \tilde{y}(X,t_1) - \tilde{\varphi}\ (t_1) \\ \tilde{y}(X,t_2) - \tilde{\varphi}\ (t_2) \\ \vdots \\ \tilde{y}(X,t_m) - \tilde{\varphi}\ (t_m) \end{bmatrix}$$

У задачах даного типу нев'язка $\|F(X)\|$, очевидно, повинна бути найменшим в точці оптимуму, оскільки відповідно до загальноприйнятої практики необхідно наближатися як можна ближче до реальної траєкторії. Хоча наведена функція методу найменших квадратів (LS) рівняння (2.2) може бути мінімізована за допомогою загального методу оптимізації при відсутності обмежень. Граді матриця Гессе для задачі LS (2.2) мають особливу структуру.

Після позначення матриці Якобі для *F(X)* розмірністю $m \times n$ через *J(X)*, вектора градієнта функції *f(X)* через *G(X)*, матриці Гессе для *f(X)* через *H(X)* і матриці Гессе для кожної $F_i(X)$ через $H_i(X)$ одержимо

$$G(X) = 2J(X)^T F(X);$$
$$H(X) = 2J(X)^T J(X) + 2Q(X), \tag{2.4}$$

де

$$Q(X) = \sum_{i=1}^{m} F_i(X) H_i(X).$$

Матриця *Q(X)* має властивість, що коли нев'язка $\|F(X)\|$ прямує до нуля при прямуванні X_k до точки розв'язку, то *Q(X)* прямує до нуля. Таким чином, при невеликих значеннях $\|F(X)\|$ в точці розв'язку, одним з найбільш ефективних

методів є використання напрямку Ньютона-Гаусса як основу для процедури й оптимізації.

2.1. Метод Ньютона-Гаусса

Відповідно до методу Ньютона-Гаусса напрямок пошуку D_k перебуває на кожній ітерації к так, що б розв'язок задачі на метод найменших квадратів буде

$$\min_{x\in\Re}\|J(X_k)D_k - F(X_k)\|_2^2, \tag{2.5}$$

Отриманий відповідно до даного методу напрямок є еквівалентом напрямку Ньютона при ігноруванні члена *Q(X)*. Напрямок пошуку X_k може бути використаний як складова стратегія лінійного пошуку, що забезпечує умову, що на кожній ітерації йде зменшення функції *f(X)*.

Розглянемо можливі переваги методу Ньютона-Гаусса. На рис. 2.1 представлена траєкторія пошуку мінімуму для функції Розенброка при використанні постановки задачі як метод найменших квадратів. Метод Ньютона-Гаусса дає збіжність після 48 звернень до розрахунку функції при кінцево-різницевому розрахунку градієнта в порівнянні з 140 ітераціями для BFGS методу без наявності обмежень.

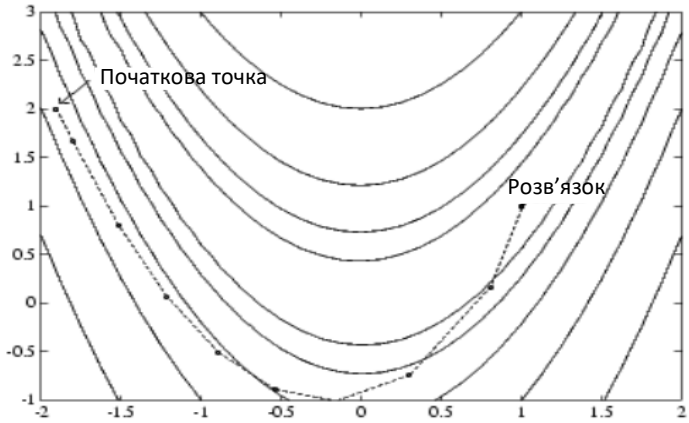


Рис. 2.1. Застосування методу Ньютона-Гаусса для функції Розенброка

У методі Ньютона-Гаусса часто зустрічається ряд проблем у ситуаціях, коли член другого порядку *Q(X)* у рівнянні (2.4) досить значний по своїй величині. Методом, що немає таких труднощів, є метод Левенберга-Марквардта.

2.2. Метод Левенберга-Марквардта

В основу методу Левенбрга-Марквардта покладений напрямок пошуку який знаходиться при розв'язанні системи лінійних рівнянь:

$$(J(X_k)^T J(X) + \lambda_k I) D_k = -J(X_k) F(X_k), \quad (2.6)$$

де скаляр λ_k задає як величину, так і напрямок параметра D_k. Коли λ_k дорівнює нулю, то напрямок D_k буде ідентичний цьому ж параметру, що в методі Ньютона Гаусса. У міру того, як λ_k прямує до нескінченності, то D_k прямує до вектора нульовими компонентами й напрямку найшвидшого спуска. У цьому випадку передбачається, що для досить великих значень λ_k $F(X_k + D_k) < F(X_k)$ залишається справедливим. Отже, член λ_k може контролюватися з метою забезпечення найшвидшого спуску якщо буде потреба обліку членів другого порядку, які, у свою чергу, помітно обмежують ефективність методу Ньютона-Гаусса.

Звідси випливає, що метод Левенберга-Марквардта заснований на напрямку пошуку, що є сполученням напрямку Ньютона-Гаусса й найшвидшого спуска. На рис. 2.2 представлено метод Левенберга-Марквардта для функції Розенброка. Розв'язок для функції Розенброка (рівняння (1.1)) збігається після 90 звертань до розрахунку функції в порівнянні з 48 для методу Ньютона-Гаусса. Така низька ефективність пов'язана з тим, що метод Ньютона-Гаусса звичайно більше ефективний у випадку, коли в розв'язку нев'язка дорівнює нулю. Однак така інформація не завжди є заздалегідь доступною й підвищена стійкість методу Левенберга-Марквардта компенсує її, але іноді має місце мала ефективність.

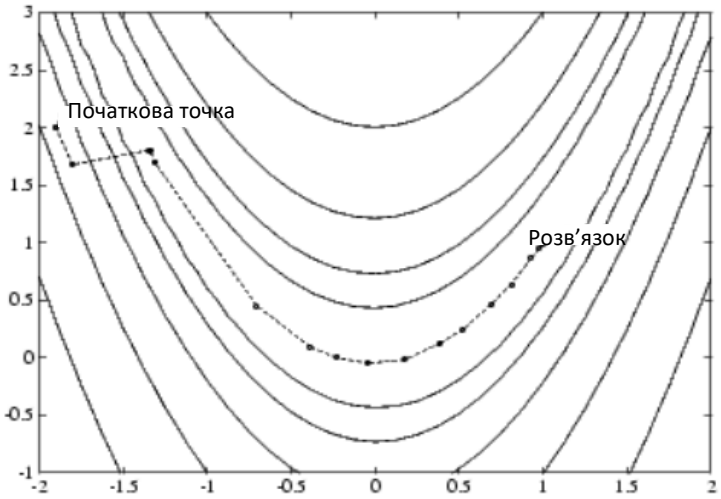


Рис. 2.2. Метод Левенберга-Марквардта для функції Розенброка

2.3. Реалізація методу найменших квадратів

Реалізація даного методу складається з двох частин:

• реалізація методу Ньютона-Гаусса;

• реалізація методу Левенберга-Марквардта.

Реалізація методу Ньютона-Гаусса

Метод Ньютона-Гаусса реалізований за допомогою стратегії поліноміального лінійного пошуку. При розв'язанні лінійної задачі методом найменших квадратів (рівняння 2.2) є можливість уникнути посилення збурень при узгодженні параметрів рівнянь шляхом використовування QR-розкладання для $J(X_k)$ і застосування подібного розкладання до $F(X_k)$. Даний підхід відрізняється від застосування інверсії явної матриці $J(X_k)^T J(X)$, де можливе виникнення несподіваних помилок.

Додаткові заходи по забезпеченню стійкості, також включені в даний метод. Дані заходи включають коригування алгоритму методу Левенберга-Марквардта у випадку, якщо довжина кроку стає нижчою за якесь порогове значення (10^{-15} в даному виконанні) або коли число обумовленості $J(X_k)$ буде менше 10^{-10}. Під числом обумовленості в даному випадку розуміється відношення найбільшого сингулярного значення до якнайменшого.

Реалізація методу Левенберга-Марквардта

Основна складність в реалізації методу Левенберга-Марквардта полягає у виборі ефективної стратегії для управління розміром λ_k для кожної ітерації таким чином, щоб вона підходила для широкого спектру задач. Прийнятий в даному методі підхід реалізації полягає в оцінці відносної не лінійності $f(X)$ на базі застосування лінійно прогнозованих сум квадратів $f_p(X_k)$ і кубічно інтерпольованої оцінки мінімуму $f_k(X_*)$. При такому підході розмір λ_k, може бути визначений для кожної ітерації. Лінійно прогнозована сума квадратів розраховується як

$$f_p(X_k) = J(X_{k-1})D_{k-1} + F(X_{k-1}), \tag{2.7}$$

а член $f_k(X_*)$ визначається з кубічної інтерполяції точок $f(X_k)$ і $f(X_{k-1})$. З дано інтерполяції так само розраховується і параметр довжини кроку a^*, що є оцінкою кроку у напрямку до мінімуму. Якщо $f_p(X_k)$ буде більше, ніж $f_k(X_*)$, то λ_k зменшується, а в протилежному випадку збільшується. Обґрунтовуванням для даного підходу служить той факт, що різниця між $f_p(X_k)$ і $f_k(X_*)$ є мірою ефективності методу Ньютона-Гаусса і ступеня лінійності задачі. В даному випадку необхідно обрати: використовувати напрям згідно напряму для методу найшвидшого спуску або використовувати напрям згідно методу Ньютона-Гаусса. Формули для відповідного зниження або збільшення параметра λ_k, які були отримані з аналізу великого числа тестових задач, приведені на наступному малюнку.

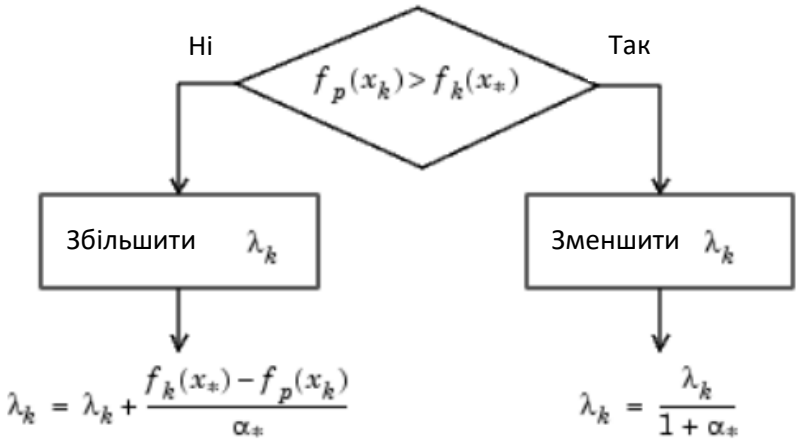


Рис. 2.3. Коригування λ_k

Далі, вслід за коригуванням λ_k, результат розв'язання рівняння (2.6) використовується для знаходження напряму пошуку D_k. Потім у напрямі D_k довжина кроку приймається рівною одиниці з подальшим застосуванням процедури лінійного пошуку аналогічно тому, що було розглянуте в розділі "оптимізація при відсутності обмежень". Процедура лінійного пошуку забезпечує виконання умови $f(X_{k+1}) > f(X_k)$ для кожної основної ітерації і таким чином, даний метод стає методом найшвидшого спуску.

Даний спосіб реалізації був успішно тестований для великого числа нелінійних задач. Було отримано, що цей метод більш стійкий, ніж метод Ньютона-Гаусса, й у багато разів більш ефективний метод за відсутності обмежень.

2.4. Приклади програм

Метод Ньютона-Гаусса

Мінімізувати функцію $f(X) = (8x_1^2 + 4x_1x_2 + 5x_1^2 + 10)^2$, починаючи з точки $X^{(0)} = [-1.8 \quad 2]^T$.

Запишемо приклад програми, що реалізує цей метод з назвою “*NG*”.

```
function NG
    % введення функції
    f = @(x) [8*x(1)^2 + 4*x(1)*x(2) + 5*x(2)^2 +
10];

    % початкова точка
    x0 = [-0.1, 0.1];

    % параметри опцій оптимізації
    options = optimset('Largescale', 'off',
'Algorithm', 'trust-region-reflective', 'Display',
'iter', 'TolX', 1e-10);

    % Виклик функції оптимізації
    [x, resnorm] = lsqnonlin(f, x0, [], [], options);

    % x - Точка мінімуму
    % resnorm - Нев'язка або сума квадратів
    disp('Точка мінімуму:');
    disp(x);
```

```
    % Введення функції для побудови поверхні
    f = @(x1, x2) (8*x1.^2 + 4*x1.*x2 + 5*x2.^2 +
10).^2;

    % Побудова поверхні функції
    ezmeshc(f,[-0.5, 0.5, -0.5, 0.5]);
end
```

Результат виконання програми

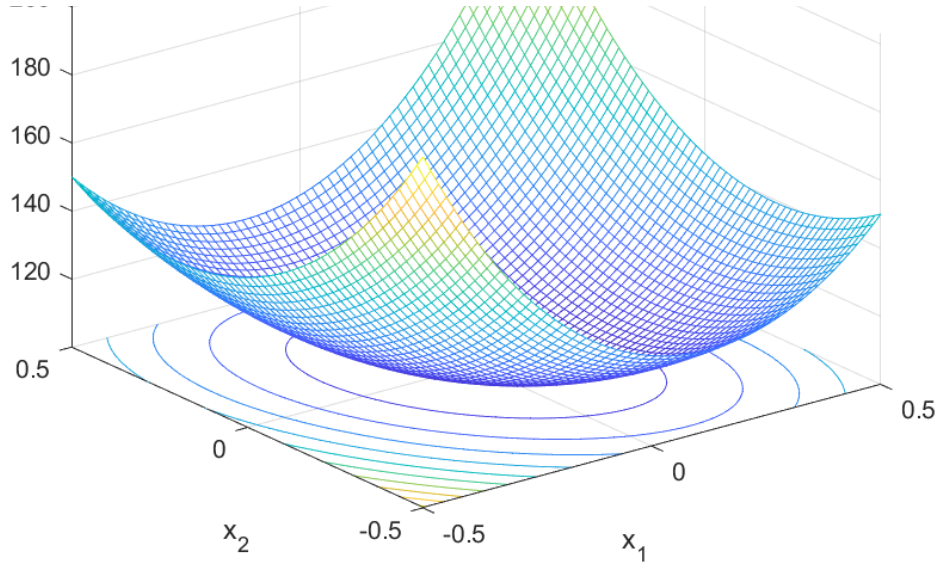
Iteration	Func-count	Residual	First-Order optimality	Lambda	Norm of step
0	3	101.808	12.1	0.01	
1	10	100.283	4.82	100	0.132978
2	13	100.142	4.69	10	0.0667125
3	17	100.089	3.53	100	0.0502577
4	20	100.056	2.86	10	0.0399184
5	24	100.036	2.28	100	0.0318577
6	27	100.023	1.82	10	0.0254505
7	31	100.015	1.46	100	0.0203424
8	34	100.009	1.16	10	0.0162647
9	38	100.006	0.931	100	0.0130071
10	41	100.004	0.744	10	0.0104034
11	45	100.002	0.595	100	0.00832149
12	48	100.002	0.476	10	0.0066566
13	52	100.001	0.381	100	0.00532499
14	55	100.001	0.305	10	0.00425985
15	59	100	0.244	100	0.00340782
16	62	100	0.195	10	0.00272624
17	66	100	0.156	100	0.002181

Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to its initial value is less than the value of the function tolerance.

<stopping criteria details>
Точка мінімуму:
   1.0e-03 *

    0.8670   0.4335



Метод Левенберга-Марквардта

Мінімізувати функцію $f(X) = \left(\frac{x_1}{1+x_2^2+x_1^2} + 25\right)^2$, починаючи з точки $X^{(0)} = [0{,}9 \quad -0{,}1]^T$.

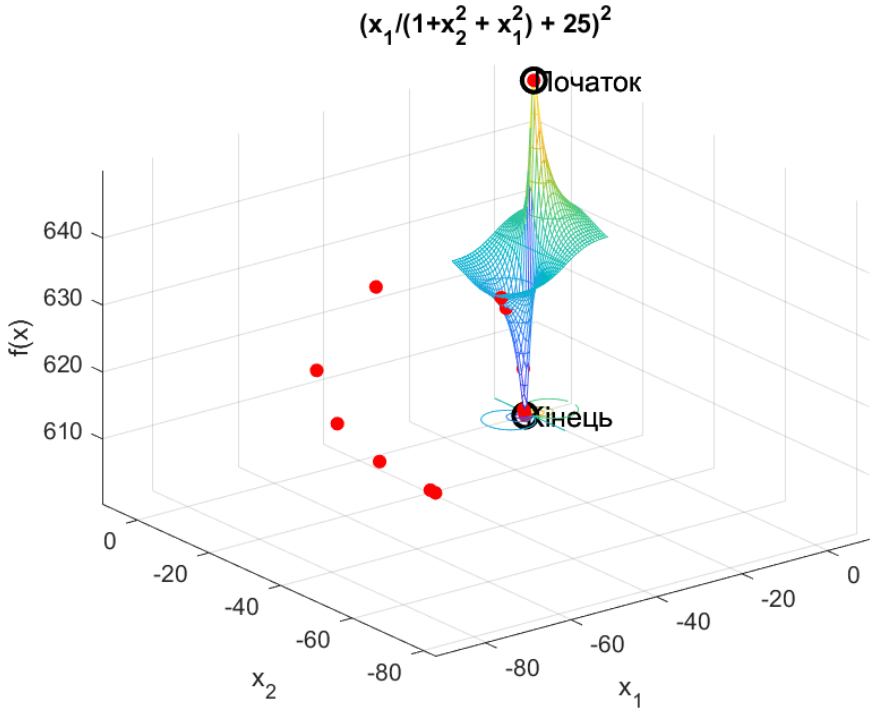
Запишемо приклад програми, що реалізує цей метод з назвою "*LM*".

```
function LM
% Введення функції
f = @(x) x(1)/(1 + x(2)^2 + x(1)^2) + 25;
% Початкова точка
x0 = [0.9; -0.1];
% параметри опцій оптимізації
options = optimoptions('lsqnonlin', 'Algorithm',
'levenberg-marquardt', 'Display', 'iter',
'OutputFcn', @OutFcn);

%Виклик функції оптимізаці
[x, resnorm] = lsqnonlin(f,x0, [], [], options);
% точка мінімуму x
% нев'язка або сума квадратів resnorm
% виведення додаткової інформаці
text(x0(1),x0(2), f(x0)^2, 'Початок', 'Color', [0 0
0], 'EraseMode', 'none', 'Fontsize',12);
text(x(1),x(2), f(x)^2, 'Кінець', 'Color', [0 0 0],
'EraseMode', 'none', 'FontSize',12);
plot3(x0(1),x0(2), f(x0)^2, 'ko', 'MarkerSize', 10,
'Linewidth',2, 'EraseMode', 'none');
plot3(x(1),x(2), f(x)^2, 'ko', 'MarkerSize', 10,
'LineWidth',2, 'EraseMode', 'none');
title('(x_1/(1+x_2^2 + x_1^2) + 25)^2');
xlabel('x_1');
ylabel('x_2');
zlabel('f(x)');
function stop = OutFcn(x, optimvalues, state)
% Виклик функції виводу
stop= false;
PlotIter(x); PlotObjFun(x);
function PlotIter(x)
% Побудова ітерацій
x1 = x(1); x2 = x(2);
z1=(x1./(1+x2.^2+x1.^2)+25).^2;
```

```
plot3(x1,x2,z1, 'r.', 'EraseMode', 'none',
'Markersize',20);
drawnow;
function PlotObjFun(x)
% Побудова цільової функції
x1= -10:0.6:10;
x2= -10:0.6:10;
[x1,x2] = meshgrid(x1,x2);
z =(x1./(1+ x2.^2 + x1.^2) + 25).^2;
hold on; meshc(x1,x2,z); contour3(x1,x2,z), grid on;
```

Результат виконання програми

```
Iteration    Func-count    Residual      First-Order    Lambda
                                         optimality
     0           3          649.97           1.54         0.01
     1           6          624.702       0.00161        0.001
124.767
     2           9          624.697       0.00164       0.0001
1.62093
     3          12          624.637       0.00192        1e-05
16.5244
     4          16          624.543       0.00232       0.0001
20.2067
     5          19           624.35       0.00375        1e-05
27.3215
     6          23          623.464         0.023       0.0001
46.9443
     7          27          619.783         0.246        0.001
24.1698
     8          32          617.418         0.547          0.1
2.91101
     9          35          607.027          6.05         0.01
5.70848
    10          41          601.159          1.96           10
0.734078
    11          44          600.294          0.66            1
0.253545
    12          48          600.251        0.0861           10
0.0688566
    13          51           600.25        0.0203            1
0.00918324
    14          55           600.25       0.00452           10
0.00215639
x =

   -0.9996
   -0.0001


resnorm =

  600.2500
```



Завдання:

1.Мінімізувати методом **Ньютона-Гаусса** і **Левенберга-Марквардта** такі функції:

а) $f(X)=\left(X_1^2+12X_1-1\right)^2+\left(49X_1^2+49X_2^2+84X_1+232X_2-681\right)^2,\quad X^{(0)}=[1;1]^T;$

б) $f(X)=100\left[X_3-\left(\frac{X_1+X_2}{2}\right)^2\right]^2+\left(1-X_1\right)^2+\left(1-X_2\right)^2,\quad X^{(0)}=[1,2\ 2\ 0]^T;$

в) $f(X)=\left(X_1^2+X_2-11\right)^2+\left(1-X_1\right)^2+\left(1-X_2\right)^2,\quad X^{(0)}=\left[1.2\ 2\ 0\right]^T;$

г) $f(X)=\left(X_1+10X_2\right)^2+5\left(X_3-X_4\right)^2+\left(X_2-2X_3\right)^4+10\left(X_1-X_4\right)^4,\ X^{(0)}=\left[3\ -1\ 0\ 1\right]^T;$

д) $f(X)=\frac{\left(X_1-X_2\right)^2}{\left(1-X_1\right)^2}\left[1-X_1-X_2\left(1-X_1\right)^5\right]^2,\quad X^{(0)}=\left[1.2\ 1\right]^T.$

Порівняти збіжність алгоритмів методів **Ньютона-Гаусса** і **Левенберга-Марквардта**.

Завдання для самостійної роботи. Мінімізувати цільову функцію $f(X) = U_1^2 + U_2^2 + U_3^2$ методом **Левенберга-Марквардта**, де $U_i = C_i - X_1(1 - X_2^i)$, $C_1 = 1{,}5$; $C_2 = 2{,}25$; $C_3 = 2{,}625$; $X^{(0)} = [2\ 0{,}2]$. Як початкову вибрати точку $X^{(0)} = [2\ 0{,}2]^T$.

Комп’ютерний практикум № 3

3. Нелінійне програмування при наявності обмежень

Як правило, загальний підхід до розв'язання задач оптимізації при наявності обмежень полягає у заміні вихідної задачі з обмеженнями на іншу більш просте задачу, яка надалі використовується як основа для деяких ітераційних методів. Основною особливістю методів, що були запропоновані на початку є те, що вихідна задача з обмеженнями заміняється на задачу без обмежень, але застосуванням методу штрафних функцій поблизу значень, що накладають, для обмежень. При такому підході задача оптимізації при наявності обмеження вирішувалася через введення деякої послідовності параметричних задля оптимізації без накладення обмежень, які в межі (обраної послідовності) збігали до шуканої задачі з обмеженнями. У наш час такий підхід вважається відносно малоефективним і, відповідно, був замінений на методи розв'язку, засновані формулюванні і наступному розв'язку так званих рівнянь Куна-Таккера (КТ). У рівняннях КТ вводяться додаткові припущення про характер обмежень і поняття оптимальності для задачі оптимізації при наявності обмежень. Якщо поставлена задача є так званою задачею опуклого програмування, тобто $f(X)$ і $G_i(x)$, $i=1,...,m$ є опуклими функціями, то рівняння КТ є необхідними й достатніми умовами для загальної постановки задачі.

$$\nabla f(X^*) + \sum_{i=1}^{m} \lambda_i^* \nabla G_i(X^*) = 0$$

$$\lambda_i^* G_i(X^*) = 0 \quad i = 1, \dots, m; \tag{3.1}$$

$$\lambda_i^* \geq 0.$$

Перші рівняння являють собою опис процесу зникнення градієнта між цільовою функцією й активними обмеженнями в точці розв'язку. Оскільки градієнти підлягають виходу на нульові значення, то множники Лагранжа $(\lambda_i, i = 1, \dots, m)$ будуть необхідні для того, щоб зрівноважити відхилення по ве личині даної цільової функції й градієнтів обмежень. Оскільки тільки активні

обмеження залучені в дану процедуру обнулення, а неактивні обмеження не повинні піддаватися даній процедурі, й тому відповідні множники Лагранжа будуть дорівнювати нулю. Ця обставина неявно виражена у двох останніх рівняннях (3.1).

Подібне розв'язання рівнянь Куна-Таккера є основою для більшості алгоритмів нелінійного програмування. У цих алгоритмах часто використовується пряме обчислення множників Лагранжа. Квазіньютонівські методи забезпечують лінійну збіжність шляхом нагромадження інформації другого порядку щодо рівнянь Куна-Таккера, що використовують процедури квазіньютонівського коригування. У загальному випадку ці методи можна віднести до задач послідовного квадратичного програмування (SQP), оскільки проблема QP розв'язується на кожній головній ітерації (іноді їх ще називають методами Ітераційного квадратичного програмування, Рекурсивного квадратичного програмування або Змінної метрики при наявності обмежень).

3.1. Послідовно квадратичне програмування (SQP)

Метод SQP є одним з найсучасніших методів в області нелінійного програмування. Шитковский успішно реалізував і провів тестові розрахунки за даною версією оптимізації й одержав всебічну перевагу, у порівнянні з іншими тестовими методами, у частині ефективності, точності, й відсотка успішного розв'язання задачі для великої кількості тестових задач.

Заснований на роботах Бигса [3], Хана [18] і Пауелла [20], даний метод дозволяє досить точно імітувати метод Ньютона для оптимізації при наявності обмежень, як це зроблено для оптимізації при відсутності обмежень. На кожній основній ітерації здійснюється апроксимація Гессіана для функції Лагранжа за допомогою квазіньютонівського модифікованого методу. Такий підхід далі буде потрібний для постановки підзадачі у QP, розв'язання якої далі вже використовується для формування напрямку пошуку в процедурі лінійного пошуку. Огляд методів SQP можна знайти в роботах Флетчера [12], Гіля [16], Пауелла [22], Шитковского [19]. Проте, далі приводиться опис узагальненого

методу. Відповідно до опису основної задачі основна ідея постановки підзадачі ОР полягає у квадратичній апроксимації функції Лагранжа.

$$L(X,\lambda) = f(X) + \sum_{i=1}^{m} \lambda_1 g_1(X) \tag{3.2}$$

При припущенні, що зв'язані обмеження можуть бути представлені через обмеження у вигляді нерівностей. За допомогою лінеаризації нелінійних обмежень можна одержати підзадачу QP.

Підзадача квадратичного програмування (QP)

$$\begin{aligned} &\min_{D\in\mathcal{R}^n} \frac{1}{2} D^T H_k D + \nabla f(X_k)^T D\,; \\ &\nabla g_i(X_k)^T D + g_i(X_k) = 0 \quad i = 1,\dots,m_e; \\ &\nabla g_i(X_k)^T D + g_i(X_k) \le 0 \quad i = m_e+1,\dots,m. \end{aligned} \tag{3.3}$$

Дана підзадача може бути вирішена за допомогою застосування алгоритму QP. Таке рішення основане на формулюванні нової ітерації наступного вигляду

$$X_{k+1} = X_k + a_k D_k. \tag{3.4}$$

Параметр при довжині кроку a_k визначається з відповідної процедури лінійного пошуку, що забезпечує прийнятне зменшення одержуваної цільової функції. Матриця H_k є додатно визначеною апроксимацією матриці Гессе для функції Лагранжа (рівняння (3.3)), H_k може коригуватися за допомогою кожного із квазіньютонівських методів, хоча метод BFGS швидше за все, є найбільш ефективним.

На відміну від розв'язання методом SQP для задач без обмежень, нелінійні задачі при наявності обмежень вирішуються за певне число ітерацій. Однією із причин такого факту є те, що внаслідок наявності меж на доступні для огляду області оптимізації можуть прийматися усвідомлені рішення щодо напрямків пошуку й розміру кроку.

Розглянемо функцію Розенброка (рівняння (1.1)) при наявності додаткових нелінійних обмежень у вигляді нерівностей *g(X),*

$$x_1^2 + x_2^2 - 1.5 \le 0 \tag{3.5}$$

При застосуванні методу SQP ця задача вирішується за 96 ітерацій у порівнянні з 140 ітераціями для задач без обмежень. На рис. 3.1 представлений шлях до точки розв'язку X = [0.9072, 0.8228] починаючи з початкової точки X = [-1.9, 2].

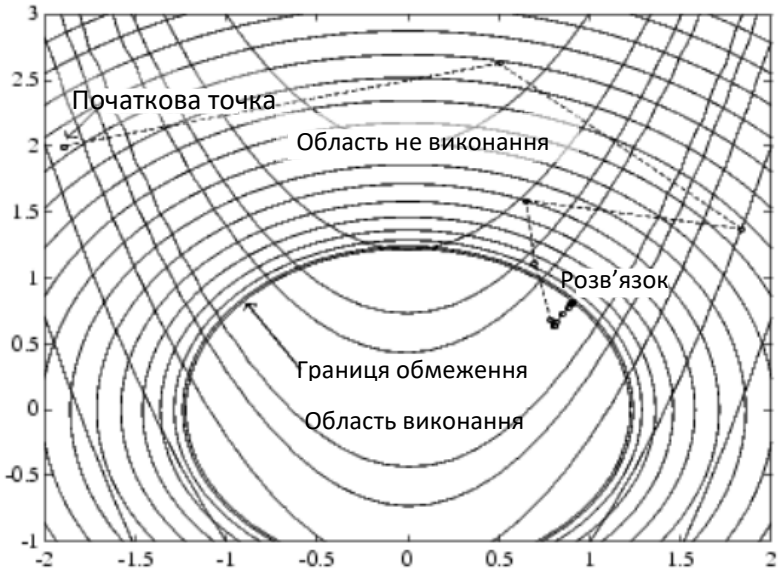


Рис. 3.1. Застосування методу SQP для функції Розенброка (1.1) з лінеаризованими нелінійними обмеженнями

3.2. Реалізація послідовно квадратичного програмування (SQP)

Реалізація методу SQP складається з трьох основних стадій.

- Коригування матриці Гессе для функції Лагранжа.
- Розв'язання задачі квадратичного програмування.
- Обчислення лінійного пошуку й функції вигоди.

Коригування матриці Гессе для функції Лагранжа

На кожній головній ітерації додатно визначена квазіньютоновська апроксимація H для функції Лагранжа розраховується за допомогою методу BFGS, де λ_i, i = 1, ..., m являють собою оцінку множників Лагранжа.

$$H_{k+1} = H_k + \frac{Q_k Q_k^T}{Q_k^T S_k} - \frac{H_k^T H_k}{S_k^T H_k S_k}, \tag{3.6}$$

де

$S_k = X_{k+1} - X_k$;

$$Q_k = \nabla f(X_{k+1}) + \sum_{i=1}^{n} \lambda_i \nabla g_i(X_{k+1}) - \left(\nabla f(X_k) + \sum_{i=1}^{n} \lambda_i \nabla g_i(X_k) \right).$$

Пауелл [21] рекомендує підтримувати значення матриці Гессе додати визначеною, навіть незважаючи на те, що ці розв'язки можуть і не мати додатнього. Додатне значення матриці Гессе підтримується в тому випадку, якщо величина, буде більше нуля для кожного коригування *H*. Коли величин $Q_k^T S_k$, не є додатною, то параметр Q_k модифікується поелементно, крок за кроком так щоб виконувалася умова $Q_k^T S_k > 0$. Узагальнена мета такої модифікації полягає в тім, що б трохи змінити елементи Q_k, які становлять основний внесок додатне визначене коригування. Таким чином, на початковій стадії прийняття модифікації, найбільший від'ємний елемент з набору $Q_k^T S_k$, послідовно зменшується на половину. Така процедура триває доти, поки $Q_k^T S_k$, більше або дорівнює 10^{-5}. Якщо після такої процедури $Q_k^T S_k$ однаково залишається від'ємним, то створюється модифікація Q_k шляхом додавання якогось вектора υ, помноженого на якусь скалярну постійну ω, а саме,

$$Q_{k+1} = Q_k + \omega\upsilon, \tag{3.7}$$

де

$$\upsilon_i = \begin{cases} \nabla g_i(X_{k+1}) g_i(X_{k+1}) - \nabla g_i(X_k) g_i(X_k), \textit{ якщо } (q_k)_i \omega < 0 \wedge (q_k)_i (s_k)_i < 0, i = 1, \dots, m \\ 0, \textit{ інакше} \end{cases}$$

або ж у протилежному випадку методично збільшують ω доти, поки член $Q_k^T S_k$ стане додатним.

Програми *fmincom, fminmax, fgoalattain* та *fseminf* реалізовані на основі методу SOP. Якщо опційний параметр Display встановлений як 'iter" відображення інформації на кожній ітерації, то виводиться різна інформація, включаючи значення функції й максимальне порушення поставлених обмежень. Також відображається модифікований Гессіан, у випадку, якщо він зазнає певної корекції на початковій стадії процедури, що виконується з метою підтримки його додатнього значення. Якщо Гессіан піддається повторному коригуванню на другій стадії наведеного вище підходу, то відображається його вторинне значення. Якщо підзадача ОР не є здійсненною, то цей факт так само

відображається. Така відображувана інформація, як правило, не представляє особливої цікавості, але вказує на те, що дана задача є істотно нелінійною, а збіжність може бути більш тривалою, ніж зазвичай. Іноді відображуване повідомлення про те, що величина $Q_k^T S_k$, не змінюється, вказує на те, що вона буде рівна нулю. Це може вказувати, що постановка задачі є неправильною або проводиться оптимізація розривної функції.

Розв'язання задачі квадратичного програмування

На кожній основній ітерації методу SQP вирішується задача QP в такій формі, де A_i ставиться до i-го рядку матриці A розмірністю $m \times n$:

$$\begin{aligned} &\min_{D \in \mathcal{R}^n} q(D) = \frac{1}{2} D^T H D + C^T D; \\ &A_i D = b_i \;\; i = 1, \dots, m_e; \\ &A_i D \leq b_i \;\; i = m_e + 1, \dots, m; \end{aligned} \tag{3.8}$$

Оптимізаційний метод заснований на стратегії активних наборів (більш відомий як метод проекцій) й аналогічний методу, описаному в роботі Гіля [15], [14]. Цей метод був модифікований для задач лінійного програмування (LP) і квадратичного програмування (QP).

Процедура розв'язання містить у собі дві фази. Перша фаза являє собою розрахунок найбільш імовірної точки. Друга фаза містить у собі генерацію якоїсь ітеративної послідовності найбільш імовірних точок, що вже збігається до необхідного розв'язку. У такому методі стверджується, що активний набір, $\bar{A}_k$, є певною оцінкою активних обмежень (тобто те, що являє собою обмежувальні межі) у даній точці розв'язку. Фактично всі алгоритми QP є методами активних наборів. Це варто особливо підкреслити, оскільки існує багато різних методів, які надзвичайно схожі по своїй суті, але представлені у вигляді різних способів свого подання.

$\bar{A}_k$ коригується на кожній ітерації й k використовується для побудови основи для напрямку пошуку $\widehat{D}_k$. Обмеження у вигляді рівностей завжди залишаються в даному активному наборі $\bar{A}_k$. Позначення для змінної у вигляді

$\widehat{D}_k$ в цьому випадку використовується для того, щоб розрізняти ці змінні від D використовуваних на головних ітераціях методу SQP. Напрямок пошуку $\widehat{D}_k$ розраховується й далі використовується для мінімізації цільової функції незважаючи на те, що залишаються в силі всі можливі межі активних обмеження. Весь можливий простір для змінних $\widehat{D}_k$, утвориться з базису Z_k, стовпчики якого ортогональними щодо отриманих результатів розрахунку активного набору (тобто, $\bar{A}_k Z_k = 0$). Таким чином, напрямок пошуку, що формується шлюз підсумовування всіх можливих сполучень колонок Z_k, забезпечує збереже межі активних обмежень.

Прийнята матриця Z_k формується з останніх m - l стовпчиків QR – розкладання матриці $\bar{A}_k^T$, де l є число активних обмежень, а так само справедливо $l < m$. Z_k визначається так

$$Z_k = Q[:, l+1:m], \tag{3.9}$$

де

$$Q^T \bar{A}_k^T = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

Як тільки Z_k буде знайдено, то далі шукається новий напрямок пошуку $\widehat{D}_k$, що у свою чергу приводить до мінімуму $q(D)$, де $\widehat{D}_k$ є нульовий простір активних обмежень. $\widehat{D}_k$ є лінійна комбінація стовпців Z_k: $\widehat{D}_k = Z_k P$ для деякого вектора P. На наступному етапі, якщо за допомогою відповідної постановки для $\widehat{D}_k$, ввести квадратичне рівняння як деяку функцію від P, одержимо:

$$q(P) = \frac{1}{2} P^T Z_k^T H Z_k P + C^T Z_k P. \tag{3.10}$$

Після диференціювання по P одержимо

$$\nabla q(P) = Z_k^T H Z_k P + Z_k P \tag{3.11}$$

Член $\nabla q(P)$ має відношення до проекції градієнта квадратичної функції, оскільки він є проекцією градієнта в підпросторі функцій Z_k. Член $Z_k^T H Z_k$ є так звана проекція Гессіана. Вважаючи, що матриця Гессе H є додатно визначена (що справедливо стосовно до реалізації методу SQP), то мінімум функції $q(P)$ у

підпросторі функцій Z_k буде визначатися умовою $\nabla q(P) = 0$, отже, для пошуку мінімуму функції необхідне розв'язання системи лінійних рівнянь

$$Z_k^T H Z_k P = -Z_k^T C \tag{3.12}$$

А крок у напрямку мінімуму буде в такому вигляді

$$X_{k+1} = X_k + a\widehat{D}_k, \text{де } \widehat{D}_k = Z_k^T P \tag{3.13}$$

Внаслідок квадратичної природи цільової функції на кожній ітерації потрібне тільки одне можливе значення розміру кроку a. Одиничний крок у напрямку $\widehat{D}_k$, саме і є крок у напрямку мінімуму функції, обмеженої межами нульового простору $\bar{A}_k$. Якщо буде можливо прийняти такий крок, без порушення прийнятих обмежень, то це саме й буде розв'язок QP. У протилежному випадку, крок уздовж напрямку $\widehat{D}_k$, убік найближчого обмеження буде менше одиниці й в активний набір на наступній ітерації будуть включені нові обмеження. Відстань до границь обмежень у будь-якому напрямку $\widehat{D}_k$, можна представити як

$$a = \min_i \left\{ \frac{-(A_i X_k - B_i)}{A_i \widehat{D}_k} \right\}, i = 1, \dots, m. \tag{3.14}$$

яке визначено для обмежень не з активного набору, де напрямок й, показує напрямок до границь обмежень, тобто $A_i \widehat{D}_k > 0$, i=1, …, m.

У випадку включення n незалежних обмежень в активний набір, без локалізації мінімуму, то для відповідності невиродженної системі лінійний рівнянь,

$$\bar{A}_k \lambda_k = C. \tag{3.15}$$

варто розраховувати множники Лагранжа.

Якщо всі елементи λ_k більше нуля, то X_k с розв'язком для точки оптимум задачі. Однак, якщо який-небудь із елементів λ_k буде менше нуля, то компоненти не будуть відповідати обмеженням у вигляді рівностей, а отже відповідний елемент необхідно визначати з активного набору, так само потрібен обіг до нової ітерації.

У даному алгоритмі для успішного початку роботи потрібна як припустима початкова точка. Якщо поточна точка з методу SQP не є припустимою, то тоді якусь точку можна знайти з розв'язку задачі лінійного програмування.

$$\min_{\gamma \in \mathcal{R}, X \in \mathcal{R}^n} \gamma;$$

$$A_i X = b_i, \;\; i = 1, \ldots, m_e; \tag{3.16}$$

$$A_i X - \gamma \leq 0, \;\; i = m_e + 1, \ldots, m.$$

Позначення A, вказує на *i*-у точку матриці. Припустиму точку для рівнянь (3.16) можна визначити з розв'язку недо- або перевизначеної системи лінійних рівнянь, отриманої із системи обмежень типу рівностей. Якщо розв'язок такої задачі існує, то фіктивна змінна у встановлюється в цьому випадку як якась максимальна розбіжність.

Згаданий вище алгоритм QP можна модифікувати стосовно до задач LP шляхом установки на кожній ітерації напрямку пошуку в напрямок найшвидшого спуска, де g_k є градієнтом узятої цільової функції (рівність коефіцієнтів лінійної цільової функції).

$$\hat{d}_k = -Z_k Z_k^T g_k \tag{3.17}$$

Якщо після використання раніше згаданого методу LP знайдена якась припустима точка, то далі запускається основна фаза методу QP. Напрямок пошуку $\widehat{D}_k$, ініціалізується разом з напрямком пошуку $\hat{d}_1$, отриманим з розв'язку системи лінійних рівнянь

$$H\hat{d}_1 = -g_k \tag{3.18}$$

де є градієнт цільової функції для поточної ітерації X_k, (тобто $HX_k + C$).

Якщо для задачі QP неможливо визначити припустиму точку, то напрямок пошуку для основної підпрограми SQP $\widehat{D}_k$, приймається з умови мінімізації величини γ.

Лінійний пошук і функція цілі

Розв'язання підзадачи QP приводить до формування вектора $\widehat{D}_k$, що у свою чергу, використовується при формуванні нової ітерації типу

$$X_{k+1} = X_k + aD_k. \tag{3.19}$$

Для того, що б одержати достатнє зменшення функції цілі оцінюється параметр довжини кроку a_k. Реалізована в даному алгоритмі функція цілі була раніше використана Ханом [18] і Пауеллом [21] і має такий вигляд

$$\psi(X) = f(x) + \sum_{i=1}^{m_e} r_1\, g_1(X) + \sum_{i=m_e+1}^{m} r_1 \max\{0, g_i(X)\}.$$

Пауелл, рекомендує введення наступного штрафного параметра:

$$r_i = (r_{k+1}) = \max_i \left\{\lambda_i, \frac{1}{2}((r_k)_i) + \lambda_i\right\}, i = 1, \dots, m. \tag{3.21}$$

Даний підхід дає позитивний внесок для прийнятих обмежень, але які є неактивними для розв'язання задачі ОР, хоча раніше це були активні значення. У прийнятому методі реалізації параметр штрафу як вихідний параметр має вигляд:

$$r_i = \frac{\|\nabla f(X)\|}{\|\nabla g_i(X)\|} \tag{3.22}$$

де $\| \, . \, \|$ являє собою евклідову норму.

Такий підхід забезпечує істотний внесок від обмеження з невеликими параметрами в значення штрафних параметрів, що особливо актуально для активних обмежень поблизу точки розв'язку.

3.3. Приклад програми

Пошук мінімума нелінійної задачі з обмеженнями з застосуванням методу **послідовного квадратичного програмування**

Знайти значення X, що мінімізує $f(X) = 4x_1^2 + 5x_2^2 - 2x_1x_2$, починаючи з точки $X^{(0)} = [1{,}5 \;\; -1]^T$, з урахуванням обмежень.

$$x_1^2 + x_2^2 - 0{,}5 \;\leq 0$$

$$x_1 x_2 - 0{,}01 = 0$$

Запишемо програму, що розв'язує дану задачу з назвою "ConOptim"

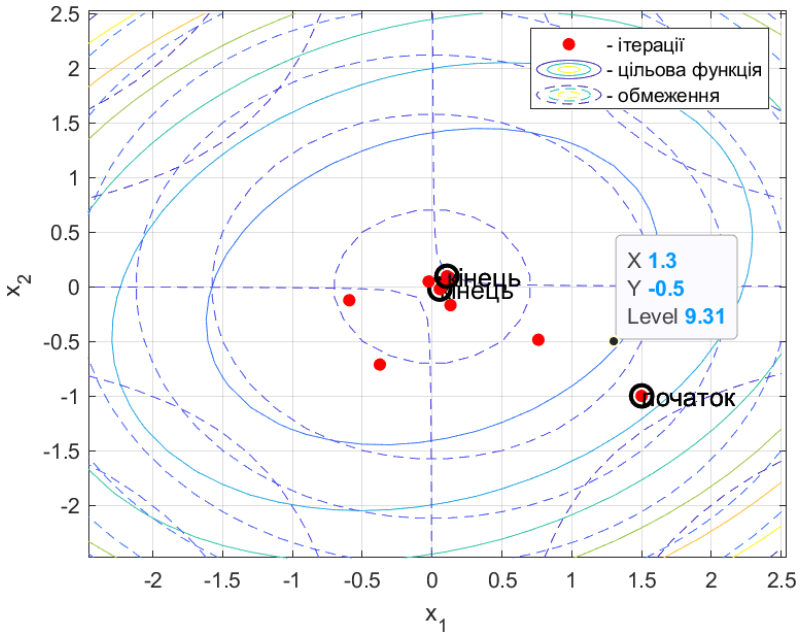
```
function Conoptim
% Нижні межі
lb = [];
% Верхні межі
ub = [];
% Початкова точка
x0 = [1.5; -1];
% Параметри опцій оптимізації з обмеженням
options = optimset('LargeScale', 'off',
'Display','iter', 'OutputFcn', @OutFcn);
% Виклик функції оптимізації
[x, fval, exitflag, output] = fmincon(@objfun, x0,
[], [], [], [], lb, ub, @confun, options);
x % точка мінімуму
fval % значення функції в точці мінімуму
% обмеження задовольняють розв'язку
[c, ceq] = confun(x)
% виведення додаткової інформації
text(x0(1),x0(2), 'початок' , 'Color', [0 0 0],
'EraseMode', 'none', 'Fontsize', 12);
text(x(1), x(2), 'кінець', 'Color', [0 0 0],
'EraseMode', 'none', 'Fontsize', 12);
plot(x0(1),x0(2), 'ko', 'Markersize', 10,
'LineWidth', 2, 'EraseMode', 'none');
plot(x(1), x(2), 'ko', 'Markersize', 10, 'LineWidth',
2, 'EraseMode', 'none');
xlabel('x_1'); ylabel('x_2'); zlabel('f(x)');
legend('- ітерації','- цільова функція','-
обмеження');
end

function f = objfun(x)
% цільова функція
f = 4*x(1)^2 + 5*x(2)^2 - 2*x(1)*x(2);
end

function [c, ceq] = confun(x)
% обмеження типу нерівності
c = x(1)^2 + x(2)^2 - 0.5;
```

```
% обмеження типу рівність
ceq = x(1)*x(2) - 0.01;
end

function stop = OutFcn(x, optimvalues, state)
% Виклик функції виводу
stop = false;
PlotIter(x); PlotObjFunConFun(x);
end

function PlotIter(x)
% Побудова ітерацій
x1 = x(1); x2 = x(2);
z1 = 4*x1.^2 + 2*x2.^2 - 2*x1.*x2;
plot(x1,x2, 'r.', 'EraseMode', 'none', 'MarkerSize',
20);
drawnow;
end

function PlotObjFunConFun(x)
% побудова цільової функції та обмежень
x1 = -2.5:0.2:2.5;
x2 = -2.5:0.2:2.5;
[x1,x2] = meshgrid(x1,x2);
zf = 4*x1.^2 + 5*x2.^2 - 2*x1.*x2;
zc1 = x1.^2 + x2.^2 - 0.5;
zc2 = x1.*x2 - 0.01;
hold on; contour(x1,x2,zf), contour(x1,x2,zc1,'--'),
contour(x1,x2,zc2,'--'), grid on;
end
```

Результат виконання програми

Iter	F-count	f(x)	Feasibility	First-order optimality	Norm of step
0	3	1.700000e+01	2.750e+00	9.569e+00	
1	6	4.235206e+00	3.799e-01	4.900e+00	9.005e-01
2	9	2.570375e+00	2.573e-01	8.043e+00	1.158e+00
3	14	1.340435e+00	6.415e-02	5.055e+00	6.278e-01
4	17	2.566597e-01	3.222e-02	1.374e+00	7.265e-01
5	20	1.697410e-02	1.124e-02	3.798e-01	1.654e-01
6	24	1.592883e-02	1.118e-02	4.046e-01	1.059e-01
7	28	1.929166e-02	9.834e-03	5.895e-01	1.050e-01
8	32	6.340944e-02	1.685e-03	1.202e+00	8.324e-02
9	35	7.305142e-02	4.957e-05	2.582e-01	1.258e-02
10	38	6.856396e-02	1.346e-04	9.994e-02	1.713e-02
11	41	6.942389e-02	2.712e-06	2.008e-02	2.574e-03
12	44	6.944272e-02	1.822e-10	2.005e-04	1.954e-05
13	47	6.944272e-02	2.311e-13	6.072e-06	6.818e-07
14	50	6.944272e-02	8.581e-14	2.000e-06	4.155e-07
15	53	6.944272e-02	2.602e-17	2.000e-08	7.278e-09

```
x =

    0.1057
    0.0946


fval =

    0.0694


c =

   -0.4799


ceq =

  -2.6021e-17
```



Завдання

1. Мінімізувати $f(X) = 100(X_2 - X_1^2)^2 + (1 - X_1)^2$ при обмеженнях:

$$g_1(X) = X_1 + 1 \geq 0;$$

$$g_2(X) = 1 - X_2 \geq 0;$$

$$g_3(X) = 4X_2 - X_1 - 1 \geq 0;$$

$$g_4(X) = 1 - 0{,}5X_1 - X2 \geq 0;$$

Записати для цієї задачі штрафні функції, використовуючи: логарифмічний штраф; штраф, який задано оберненою функцією. Дослідити кожну із функцій у точках $X^{(0)}$=[-1 1]^T та [-0,5 0,5]^T, звернувши увагу на труднощі, що зустрічаються.

2. Використовуючи метод послідовного квадратичного програмування мінімізувати функцію $f(X) = -X_1^2 - X_2^2$ при обмеженнях

$$X_1 \geq 0;$$

$$X_2 \geq 0;$$

$$X_1 + 2X_2 \leq 3.$$

3. Використовуючи метод послідовного квадратичного програмування, мінімізувати функцію $f(X) = \frac{4}{X_1} + \frac{9}{X_2} + (X_1 + X_2)$ при обмеженнях

$$X_1 \geq 0;$$

$$X_2 \geq 0;$$

$$X_1 + 2X_2 \leq 4.$$

4. Мінімізувати функцію $f(X) = X_1^2 + X_2^2 + X_3^2$ при обмеженнях використовуючи метод послідовного квадратичного програмування

$$X_1 + X_2 + X_3 \geq 3;$$

$$X_1X_2X_3 \geq 3;$$

$$X_1 > 0;$$

$$X_2 \geq 0;$$

$$X_3 \geq 0;$$

5. Мінімізувати функцію $f(X) = (X_1 - X_2)^2 + (X_2 - 1)^2$ при обмеженнях використовуючи метод штрафних функцій

$$y_1(X) = \frac{X_1^2}{4} + X_2^2 + 1 \geq 0,$$
$$h_2(X) = X_1 - 2X_2 + 1 = 0$$

а) побудувати функцію $P(X, r)$ при $r = 0{,}04$. Зобразити побудовану Р-функцію графічно;

б) визначити напрямок оптимізаційного пошуку з початкової внутрішньої точки

$X^{(n)}=[0{,}748 \;\; 0{,}548]^T$;

в) знайти вектор $X^{(1)}$;

г) Чи може одна з точок $X^{(k)}$ виявитися ззовні припустимої області.

Комп’ютерний практикум № 4

4. Багатокритеріальна оптимізація

Достатньо часто в реальних ситуаціях якість експлуатації досліджуваного об'єкту або системи оцінюється не єдиним критерієм або показником якості, а сукупністю таких критеріїв, причому однаково значущими. Така постановка задачі приводить до задачі оптимізації з векторною цільовою функцією $F(X) = \{F_1(X), F_2(X), \ldots, F_m(X)\}$, яка повинна трактуватись певним чином. Як правило, відносна значущість цієї цілі загалом невідома до тих пір, поки не будуть визначені всі основні властивості системи і повністю не тлумачитимуть всі можливі взаємозв'язки. У міру того, як число можливої цілі зростає, то очевидно, що ці взаємозв'язки утворюють складну структуру і їх стає важче ідентифікувати В даному випадку багато що залежить від інтуїції дослідника і уміння точно виражати ті або інші переваги в процесі оптимізації. Таким чином, стратегія побудови багатокритеріальної оптимізації полягає, перш за все, в здатності адекватно визначити постановку задачі так, що б ця задача допускала свої розв'язання, а також виразити необхідні переваги у вигляді числової залежності і зберігши при цьому реальність поставленої задачі.

Введення в багатокритеріальну оптимізацію

Багатокритеріальна оптимізація є мінімізацією якогось вектора цілі *F(X)*, на якій можуть бути накладені додаткові обмеження або граничні значення:

$$\min_{X \in \mathcal{R}} F(X);$$

$$G_i(X) = 0 \;\; i = 1, \dots, m_e; \tag{4.1}$$

$$G_i(X) \leq 0 \;\; i = m_e + 1, \dots, m;$$

$$x_i \leq X \leq x_u.$$

Відзначимо, що оскільки *F(X)* є певним вектором, то будь-які компоненти *F(X)* є конкуруючими і відсутній якийсь єдиний розв'язок поставленої задачі. Замість цього, для опису характеристик мети вводиться концепція безлічі точок не покращуваних розв'язків [25] (так звана оптимальність за Паретто [6], [7]). Не покращуваний розв'язок є такий розв'язок, в якому покращення в одній з цілей приводить до якогось погіршення в іншій. Для більш точного формулювання даної концепції розглянемо деяку область припустимих розв'язків Ω в параметричному просторі $X \in \mathcal{R}^n$, яке задовольняє всім прийнятим обмеженням, тобто

$$\Omega = \{X \in \mathcal{R}^n\} \tag{4.2}$$

за обмежень

$g_i(X) = 0, i = 1, \dots, m_e;$

$g_i(X) \leq 0, i = m_e + 1, \dots, m;$

$x_i \leq X \leq x_u.$

Звідси можливо визначити відповідну область припустимих розв'язків для простору цільових функцій Λ

$$\Lambda = \{y \in \mathcal{R}^n\}, \tag{4.3}$$

де $y = F(X)$ за умови $X \in \Omega$.

У двовимірному випадку, як це представлено на рис. 4.1, вектор характеристик *F(x)* відображає параметричний простір в простір цільових функцій

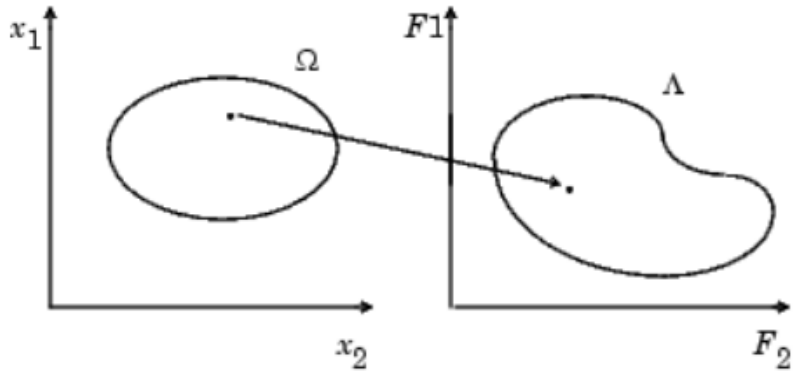


Рис. 4.1. Відображения параметричного простору в простір цільових функцій

Точка не покращуваного розв'язку може бути визначена як:

Визначення. Точка $x^* \in \Omega$ є не покращуваним розв'язком, якщо для деякої точки в околі x^* немає деякого Δx такого, що $(X^*+\Delta X) \in \Omega$ та

$$F_i(X^* + \Delta X) \leq F_i(X^*), i = 1, \dots, m; \quad (4.4)$$

$$F_i(X^* + \Delta X) < F_i(X^*) \textit{ для деякого } j.$$

Для двовимірної інтерпретації рис. 4.2. безліч не покращуваних розв'язків, лежить на кривій між точками *C* і *D*. Точки *A* і *B* представляють специфічні не покращувані розв'язки.

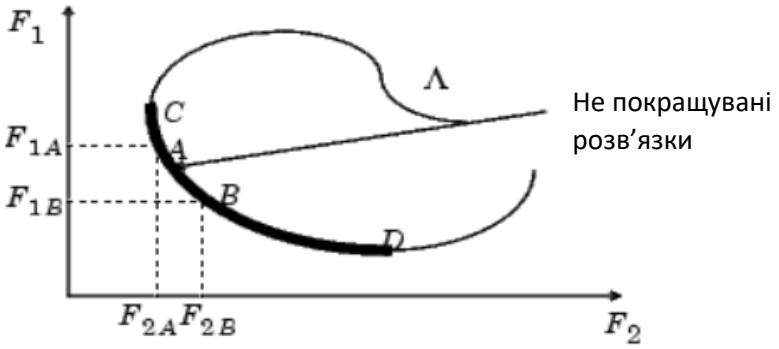


Рис. 4.2. Безліч не покращуваних розв'язків

Точки *A* і *B* є безумовними точками не покращуваних розв'язків, оскільки будь-яке покращення для однієї мети F_1 викликає погіршення для іншої вибраної мети F_2, тобто $F_{1B} < F_{1A}$, $F_{2B} < F_{2A}$.

Оскільки будь-яка точка простору Ω, тобто простору, в якому відсутні не погіршуванні розв'язки, представляє точку, в якій будь-яке покращення може

бути досягнено у всій вибраній меті, то така точка не представляє ніякої цінності. Отже, багатокритерійна оптимізація повинна включати певну генерацію і вибір точок з не покращуваними розв'язками. Подібні методики для багатокритеріальної оптимізації дуже різноманітні.

Стратегія зважених сум

Дана стратегія зважених сум перетворить багатокритеріальну задачу мінімізації вектора $F(X)$ в якусь скалярну задачу шляхом побудови певних зважених сум для всіх вибраних об'єктів

$$\min_{X \in \Omega} f(X) = \sum_{i=1}^{m} \omega_i F_i(X) \qquad (4.5)$$

Далі вже до даної задачі оптимізації вже може бути застосований стандартний алгоритм мінімізації без наявності обмежень. В цьому випадку розглядаються зважені коефіцієнти для кожної з вибраних цілей. Зважені коефіцієнти необов'язково повинні відповідати напряму відносної значущості відповідної мети або брати до уваги взаємовплив між конкретно вибраною метою. Більш того, межі не покращуваних розв'язків можуть бути і не досягнуті, так що певні розв'язки є по суті недосяжними.

Все це допускає геометричну інтерпретацію. Розглянемо випадок двох взятих цілей, як це представлено на рис. 4.3. Відобразимо лінію L $\omega^T F(x) = c$ в просторі цільових функцій, мінімізацію рівняння (4.5) можна інтерпретувати як пошук такого значення c, при якому лінія L якраз торкатиметься межі A при проведенні даної процедури розв'язку задачі зовні вибраної області.

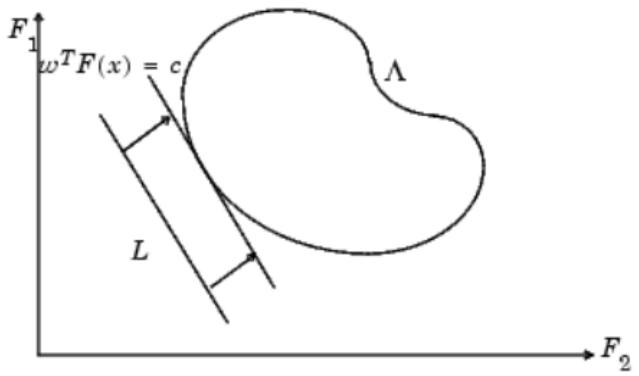


Рис. 4.3. Геометрична інтерпретація методу зважених сум

Шляхом підбору вагів ω_1 і ω_2 можливо, по суті, задати нахил лінії L таким чином, щоб у свою чергу, приводить до шуканої точки розв'язку в місці торкання лінії L з межею області пошуку розв'язку Λ.

Згадана вище проблема опуклості стає актуальною у разі, коли нижня межа області А не є опуклою, як це показано на рис. 4.4. Межа не опуклого розв'язку. В цьому випадку безліч не погіршуваних розв'язків між точками A і B не є досяжним при використанні подібних процедур.

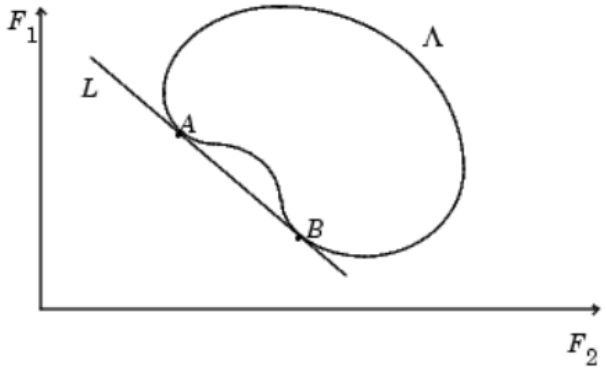


Рис. 4.4. Межа не опуклого рішення

Метод ε-обмежень

Є певний спосіб, який частково дозволяє подолати проблему опуклості методу зважених сум, це метод ε - обмежень. В цьому випадку здійснюється мінімізація основної цілі Fp і при представленні інших цілей у формі обмежень типу нерівностей

$$\min_{X\in\Omega} F_P(X), \tag{4.6}$$

за умови

$$F_i(X) \le \varepsilon_i, i = 1,\dots,m, i \ne p.$$

На рис. 4.5 представлена двовимірна інтерпретація методу ε-обмежень для задачі з двома цілями.

$$\min_{X\in\Omega} F_1(X) \text{ за умови } F_2X \le \varepsilon_2$$

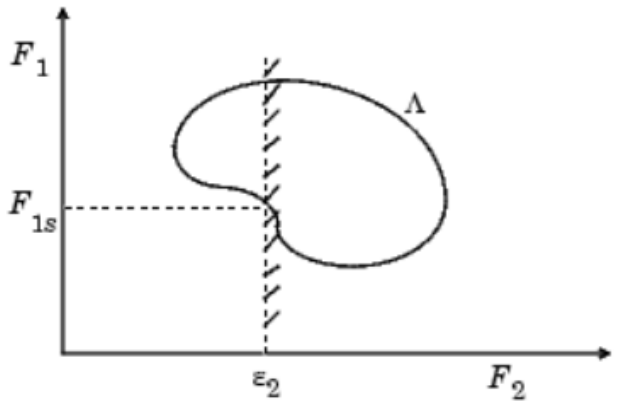


Рис. 4.5. Геометрична інтерпретація методу ε обмежень

Подібний підхід дозволяє визначити деяку кількість не покращуваних розв'язків для випадку опуклої вниз межі, що, по суті, є неприступним в методі зважених сум, наприклад, в точці шуканого розв'язків $F_1 = F_{1S}$, і $F_2 = \varepsilon_2$. Проте проблемою даного методу є відповідний вибір ε, який міг би гарантувати припустимість якогось розв'язку. Наступний недолік даного методу полягає в необхідності використовування жорстких обмежень, які не завжди є адекватними для точної побудови цілі, що задається. Подібні методи, як це приведено в роботі Валтза [40], будуються на основі вибору певного пріоритету для мети, що задається. Процедура оптимізації виконується відповідно до вибраних пріоритетів і в межах припустимості прийнятих меж. Основна складність даного методу полягає в точній інтерпретації подібної інформації на ранніх стадіях оптимізаційного циклу.

Для того, що б в математичному описі були представлені істинні цілі розробника необхідно, щоб розробник чітко представляв повний набір всіх його переваг, а також припустимі рівні діапазону поєднань значень вибраної цілі. Таким чином, необхідна методика повинна бути реалізованою таким чином, щоб максимально врахувати все вище сказане. Подібні методи були побудовані для дискретних функцій з використанням окремих досягнень теорії статистики, а саме, теорії ухвалення рішень і теорії ігор. Додаток до безперервних функцій може бути реалізований за допомогою відповідних стратегій дискретизації, а також комплексних методів розв'язання. Оскільки тільки в окремих випадках, коли розробнику відома така детальна інформація, то, мабуть, приведений вище

метод є швидше за все мало практичним.

Для даної задачі, перш за все, необхідно знати, що є достатньо доступним для точного формулювання і чисельно реалізованим, але разом з тим, що залишається в рамках інтересів розробника.

4.1. Метод досягнення цілі

Описаний далі метод є методом досягнення цілі Гембіки. Даний метод включає вираз для безлічі намірів розробника $\{F^* = F_1^*, F_2^*, \ldots, F_m^*\}$, який пов'язаний з безліччю цілей $F(X) = (F_1(X), F_2(X), ..., F_m(X))$. Таке формулювання задачі допуска, що мета може бути або недо або передосягненою, і що дає розробнику можливість відносно точно виразити початкові наміри. Відносний ступінь недо - або передосягненості поставлених намірів контролюється за допомогою вектора зважених коефіцієнтів $\Omega = \{\omega_1, \omega_2, ..., \omega_m\}$ і може бути представлений як стандартна задача оптимізації за допомогою наступного формулювання

$$\min_{\gamma \in \mathcal{R}, X \in \Omega} \gamma, \tag{4.7}$$

Член од вносить в дану задачу елемент послаблення, що, інакше кажучи, позначає жорсткість заданого наміру. Ваговий вектор Ω дає досліднику можливість достатньо точно виразити міру взаємозв'язку між двома цілями.. Наприклад, установка вагового вектора Ω як рівного початковому наміру указує на те, що досягнутий той же самий відсоток недо або передосягнення цілі F^*. За допомогою установки в нуль окремого вагового коефіцієнта (тобто $\Omega_i = 0$) можна внести жорсткі обмеження в поставлену задачу. Метод досягнення цілі забезпечує відповідну інтуїтивну інтерпретацію поставленої дослідницької задачі і яка, свою чергу, є цілком вирішуваною за допомогою стандартних процедур оптимізації. Ілюстративний приклад подібного використовування методу досягнення мети для систем управління можна знайти у Флемінга [9] та [10].

На рис. 4.6 представлена геометрична інтерпретація двовимірної задачі даного методу.

$$\min_{\gamma, X \in \Omega} \gamma \text{ за умови } F_1(X) - \omega_1 \gamma \leq F_1^*$$
$$F_1(X) - \omega_1 \gamma \leq F_1^*$$

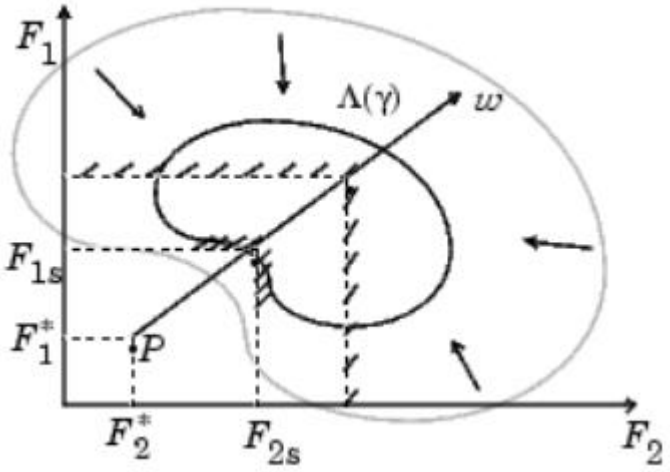


Рис. 4.6. Геометрична інтерпретація методу досягнення мети

Завдання компонентів намірів $\{F_1^*, F_2^*\}$ визначає точку намірів P. Ваговий вектор визначає пошук від точки P у бік простору припустимих функцій $\Lambda(\gamma)$. В процесі оптимізації відбувається зміна величини у, що викликає зміну розміру заданої припустимої області. Межі обмежень стягуються до єдиної у своєму роді точки розв'язку F_{1S}, F_{2S}.

4.2. Алгоритмічне поліпшення методу досягнення цілі

Метод досягнення цілі має певні переваги, а саме, на нього може бути накладена задача нелінійного програмування. Характерною особливістю даної проблеми є те, що в даному випадку можуть так само розв'язуватися задачі нелінійного програмування. Що стосується методу послідовного квадратичного програмування, то вибір функції цілі у разі лінійного пошуку є далеко нелегкою задачею, оскільки в більшості випадків достатньо важко "визначити" відносну значущість між покращенням цільової функції і ступенем відхилення від порушення обмежень, що накладаються. Такий підхід приводить до ряду диференціальних схем для побудови цільової функції (Шидковський [23]). В задачі програмування досягнення цілі може бути знайдена більш відповідна функція цілі, яка може бути отримана шляхом накладення рівняння (4.7) на задачу мінімакса

$$\min_{X\in\mathcal{R}^n} \max_i \{\Lambda_i\}, \tag{4.8}$$

де

$$\Lambda_i = \frac{F_i(X) - F_i^*}{\omega_i}, i = 1,\ldots,m.$$

Слідуючи висновкам Брайтона [4] для мінімаксної оптимізації на основі методу SOP, застосування функції вигоди до рівняння (3.20) для задачі досягнення цілі згідно рівнянню (4.8) приводить до наступної постановки задачі

$$\Psi(X,\gamma) = \gamma + \sum_{i=1}^{m} r_i max\{0, F_i(X) - \omega_i\gamma - F_i^*\} \tag{4.9}$$

Хоча функція цілі згідно рівнянню (4.9) використовується як основа процедури лінійного пошуку, то, хоча $\Psi(X,\gamma)$ може і зменшуватися для якогось кроку в заданому напрямі пошуку, максимум функції Λ_i, може, як це не парадоксально, збільшуватися. Це знаходиться у згоді з погіршенням найсприятливішої цілі. Оскільки найнесприятливіша ціль також є частиною значення цільової функції γ, то в даному випадку допускається якийсь крок, який, збільшує цільову функцію, що підлягає мінімізації. І навпаки величина $\Psi(X,\gamma)$ може зростати при зменшенні максимуму Λ_i, тим самим, маючи на увазі якесь відкидання кроку, що приводить до покращення найсприятливішої цілі.

Згідно шляху руху Брайтона [4] пошук деякого розв'язку полягає у встановленні значення (X) як рівного найсприятливішій меті, тобто як

$$\Psi(X) = \min_i \Lambda_i \tag{4.10}$$

Проблема в методі досягнення цілі полягає в тому, що в загальному випадку для того, що б враховувати жорсткі обмеження необхідно використовувати або прирівняти нулю ваговий коефіцієнт. В цьому випадку функція вигоди згідно рівнянню (4.10) стає нескінченною при довільних відхиленнях від заданих обмежень. Для того, що б подолати цю проблему, а також зберегти властиві рівнянню (4.10) особливості, функцію цілі представляють у вигляді певної комбінації з рівнянням (4.2), що приводить до наступних співвідношень:

$$\Psi(X) = \sum_{i-1}^{m} \begin{cases} r_i max\{0, F_i(X) - \omega_i\gamma - F_i^*\}, при\ \omega_i = 0 \\ \max\limits_i \Lambda_i\ ,\ \ i = 1, \ldots, m,\ \ инакше \end{cases} \quad (4.11)$$

Інша особливість, яка так само може бути використана в методі SQP, є прийнята цільова функція γ. Виходячи з рівнянь КТ (рівняння (3.3)) можна показати, що апроксимація матриці Гессе для Лагранжиана *H*, повинна мати нулі в пов'язаних із змінною γ відповідних рядках і стовпчиках матриці. Проте дана властивість ні як не виявляється, якщо матриця *H* ініціалізувалася як деяка тотожна матриця. Отже, матриця Н повинна ініціалізуватися і зберігатися як матриця з нулями, пов'язаними із змінною у відповідних рядках та стовпчиках матриці.

Подібні дії роблять матрицю Гессе *H* нескінченною. Тому необхідно встановлювати нулі в пов'язаних із змінною у відповідних рядках та стовпчиках матриці за винятком діагональних елементів, де вносяться невеликі додатні числа (наприклад 10^{-10}). Такий підхід дозволяє використовувати QP метод, що достатньо швидко збігається.

Приведені вище модифікації були внесені в програму *fgoalattain* і було знайдене, що ці корекції приводять до ефективності даного методу. Проте, внаслідок швидкої збіжності методу SQP, вимога, щоб функція цілі строго зменшувалася, іноді приводить до більшого числа розрахунків функції при упровадженні методу SOP з використанням функції цілі згідно рівнянню (4.1).

4.3. Приклади програм [1]

Автоматичний регулятор по виходу зворотного зв'язку К, сконструйований так, що 6 система була з замкненим контуром

$$\hat{x} = (A + BKC)x$$

$$y = Cx$$

Характеристичні значення системи з замкненим контуром визначаються з матриць А, В, С та К з використанням команди eig(A+B*K*C). Характеристичні значення замкненого контуру повинні бути на дійсних осях комплексної

площини зліва від точок [-5, -3, -1]. Для того, що б не переповнювалися вхідні значення, жоден з елементів матриці К не може бути більш ніж 4 або менше ніж -4. Система є дво вхідною, та дво вихідною, з розімкненим контуром, нестійкою системою, матриці у просторі станів будуть:

$$A = \begin{matrix} -0{,}5 & 0 & 0 \\ 0 & -2 & 10 \\ 0 & 1 & -2 \end{matrix}$$

Установка значень цілі (goal) для характеристичних значень замкненого контуру ініціалізувалася як goal = [-5, -3, -1].

Для забезпечення однакової частки для пере- або не досягнення в активній цілі в даному розв'язку вагова матриця weight встановлюється як abs(goal).

Початкова матриця регулятора К = [-1, -1; -1, -1].

Запишемо спочатку М - файл і збережемо його як eigfun.m.

```
function F = eigfun(K,A,B,C)
    F = sort(eig(A+B*K*C)); % оцінка цілей
end
```

Далі запишемо програму оптимізації

```
disp('Матриця A:')
A =[-0.5 0 0; 0 -2 10; 0 1 -2] % Введення матриці A
disp('Матриця B:')
B = [1 0; -2 2; 0 1] % Введення матриці B
disp('Матриця C:')
C = [1 0 0; 0 0 1] % Введення матриці C
disp('Вектор початкових умов X0:')
x0 = [-0.1 0.1 0.3]' % Завдання вектора початкових умов
disp('Ініціалізація матриці регулятора:')
K0= [-1 -1; -1 -1] % Введення оочаткової матриці регулятора
disp('Значення goal для характеристичних значень:')
goal = [-5 -3 -1] % Введення значення goal для характеристичних значень
disp('Ваги по однаковій частці:')
```

```
weight = abs(goal) % Введення ваг по однаковій частці
disp('Нижня межа для регулятора:')
lb = -4*ones(size(K0)) % Введення нижньої межі для регулятора
disp('Верхня межа для регулятора:')
ub = 4*ones(size(K0)) % Введення верхньої межі для регулятора
disp('Час спостереження tf:')
tf = 5 % Завдання часу спостереження
disp('Довжина кроку dt:')
dt = 0.01 % Завдання довжини кроку
disp('Кількість кроків n:')
n = tf/dt
options = optimset('Display','iter'); % Установка параметрів відображення
[K, fval, attainfactor] = fgoalattain(@eigfun, K0, goal, weight, [], [], [], [], lb, ub, [],
options, A, B, C)
% Визначення розмірності задачі
SA = size(A); SA = SA(1); SB = size(B); SB = SB(2); SC = size(C); SC = SC(1);
% Формування векторів x u
x = zeros(SA,n); u = zeros(SB,n-1); y = zeros(SC,n);
% Формування початкового стану
x(:,1) = x0;
% Знаходження оптимального керування u та змінних стану x
for i = 1:n - 1
   u(:,i) = K*C*x(:,i);
   x(:,i + 1) = (A*x(:,i) + B*u(:,i))*dt + x(:,i);
   y(:,i) = C*x(:,i);
end
% Побудова динаміки змінних стану
plot(0:dt:tf-dt,x), grid
title('Динаміка руху змінних стану'); xlabel('Час, t');
```

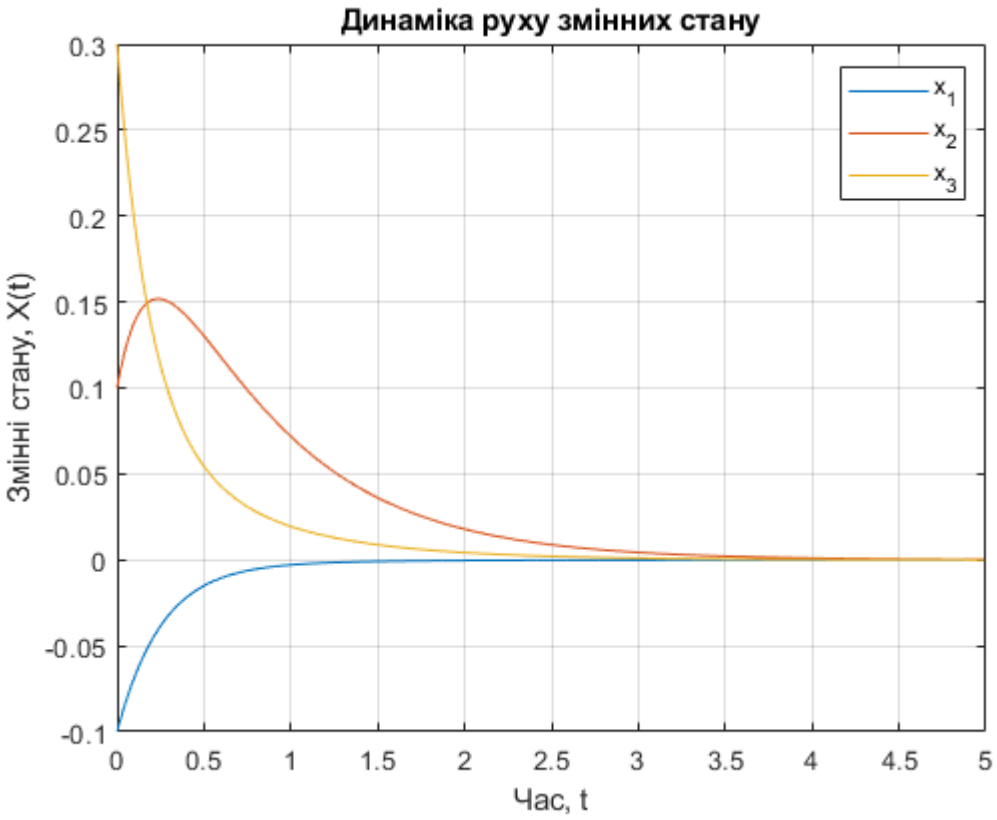
```
ylabel('Змінні стану, X(t)'); legend('x_1','x_2','x_3')
% Побудова вектора керування
figure(2)
plot(dt:dt:tf-dt,u), grid
title('Динаміка зміни керування'); xlabel('Час, t'); ylabel('Керування, U(t)');
legend('u_1','u_2')
% Побудова виходу
figure(3)
plot(0:dt:tf-dt,y), grid
title('Вихід системи'); xlabel('Час, t'); ylabel('Вихід, Y(t)'); legend('y_1','y_2')
```

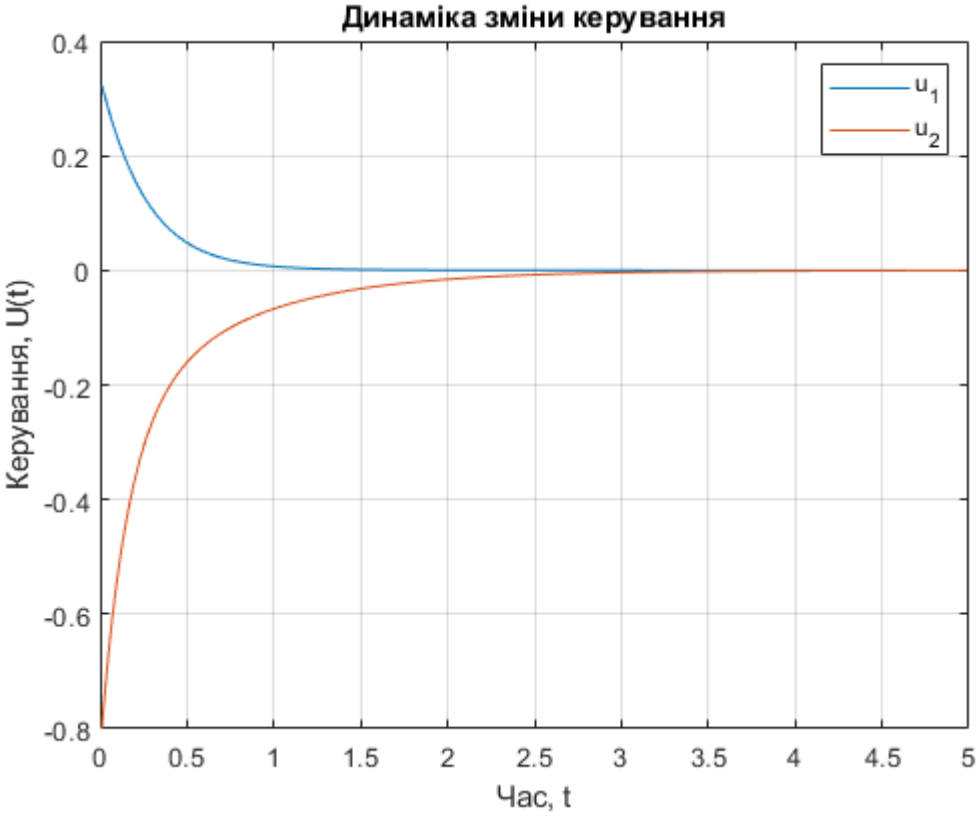
Результат виконання програми

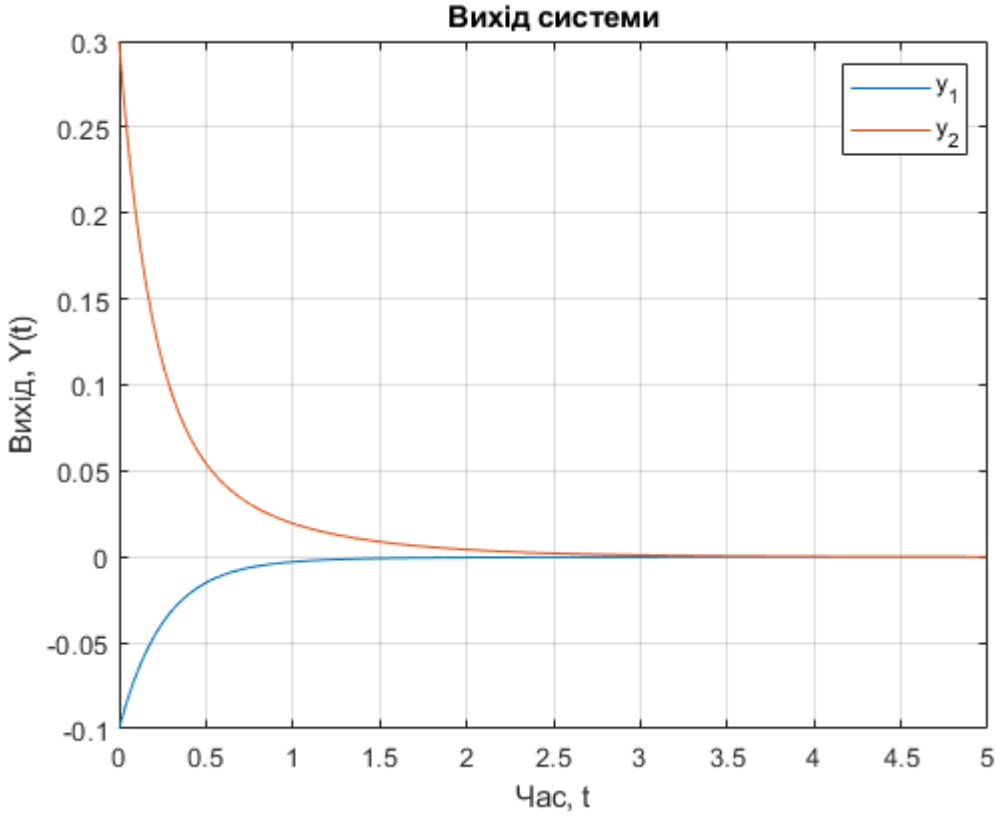
```
Матриця A:

A =

   -0.5000         0         0
         0   -2.0000   10.0000
         0    1.0000   -2.0000

Матриця B:

B =

     1     0
    -2     2
     0     1

Матриця C:

C =

     1     0     0
     0     0     1

Вектор початкових умов X0:

x0 =

   -0.1000
    0.1000
    0.3000

Ініціалізація матриці регулятора:

K0 =
```

```
    -1    -1
    -1    -1

Значення goal для характеристичних значень:

goal =

    -5    -3    -1

Ваги по однаковій частці:

weight =

     5     3     1

Нижня межа для регулятораі:

lb =

    -4    -4
    -4    -4

Верхня межа для регулятораі:

ub =

     4     4
     4     4

Час спостереження tf:

tf =

     5

Довжина кроку dt:

dt =

    0.0100

Кількість кроків n:

n =

   500


                Attainment        Max     Line search     Directional
 Iter F-count        factor    constraint   steplength      derivative   Procedure
    0      6             0       1.88521
    1     13         1.031       0.02998            1           0.745
    2     20        0.3525       0.06863            1          -0.613
    3     27       -0.1706        0.1071            1          -0.223    Hessian modified
    4     34       -0.2236       0.06654            1          -0.234    Hessian modified twice
    5     41       -0.3568      0.007894            1         -0.0812
    6     48       -0.3645      0.000145            1          -0.164    Hessian modified
    7     55       -0.3645             0            1        -0.00515    Hessian modified
    8     62       -0.3675     0.0001547            1        -0.00812    Hessian modified twice
    9     69       -0.3889      0.008327            1        -0.00751    Hessian modified
```

```
   10     76        -0.3862             0           1         0.00568
   11     83        -0.3863     4.551e-13           1          -0.998    Hessian modified twice

Local minimum possible. Constraints satisfied.

fgoalattain stopped because the size of the current search direction is less than
twice the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.


K =

   -4.0000   -0.2564
   -4.0000   -4.0000


fval =

   -6.9313
   -4.1588
   -1.4099


attainfactor =

   -0.3863
```







Коефіцієнт досягнення указує на те, що кожна ціль досягнута, принаймні, з імовірністю понад 38.63% від початкової розрахункової цілі (goals). Активні обмеження, у разі 1 і 2, є такими обмеженнями, які встановлюють бар'єр

подальшому поліпшенню і для яких відсоткова частка точно задоволена. Три обмеження по нижній межі також є активними.

В приведеному вище розрахунку програма намагається отримати ціль менше ніж goal. В якнайгіршому випадку, для задачі, коли ціль повинна якомога ближче goal, задаються опції options. GoalsExactAchieve з урахуванням числа необхідної цілі.

Розглянемо приведену задачу так, як якби потрібно було, що б всі характеристичні значення були рівні значенням цілі (goal). Рішення даної задачі здійснюється запуском fgoalattain з опцією options.GoalsExactAchieve, рівної 3.

```
disp('Матриця A:')
A =[-0.5 0 0; 0 -2 10; 0 1 -2] % Введення матриці A
disp('Матриця B:')
B = [1 0; -2 2; 0 1] % Введення матриці B
disp('Матриця C:')
C = [1 0 0; 0 0 1] % Введення матриці C
disp('Вектор початкових умов X0:')
x0 = [-0.1 0.1 0.3]' % Завдання вектора початкових умов
disp('Ініціалізація матриці регулятора:')
K0= [-1 -1; -1 -1] % Введення оочаткової матриці регулятора
disp('Значення goal для характеристичних значень:')
goal = [-5 -3 -1] % Введення значення goal для характеристичних значень
disp('Ваги по однаковій частці:')
weight = abs(goal) % Введення ваг по однаковій частці
disp('Нижня межа для регулятора:')
lb = -4*ones(size(K0)) % Введення нижньої межі для регулятора
disp('Верхня межа для регулятора:')
ub = 4*ones(size(K0)) % Введення верхньої межі для регулятора
disp('Час спостереження tf:')
tf = 5 % Завдання часу спостереження
disp('Довжина кроку dt:')
dt = 0.01 % Завдання довжини кроку
disp('Кількість кроків n:')
n = tf/dt
options = optimset('Display','iter','GoalsExactAchieve',3); % Установка параметрів
відображення
[K, fval, attainfactor] = fgoalattain(@eigfun, K0, goal, weight, [], [], [], [], lb, ub, [],
options, A, B, C)
% Визначення розмірності задачі
SA = size(A); SA = SA(1); SB = size(B); SB = SB(2); SC = size(C); SC = SC(1);
% Формування векторів x u
```

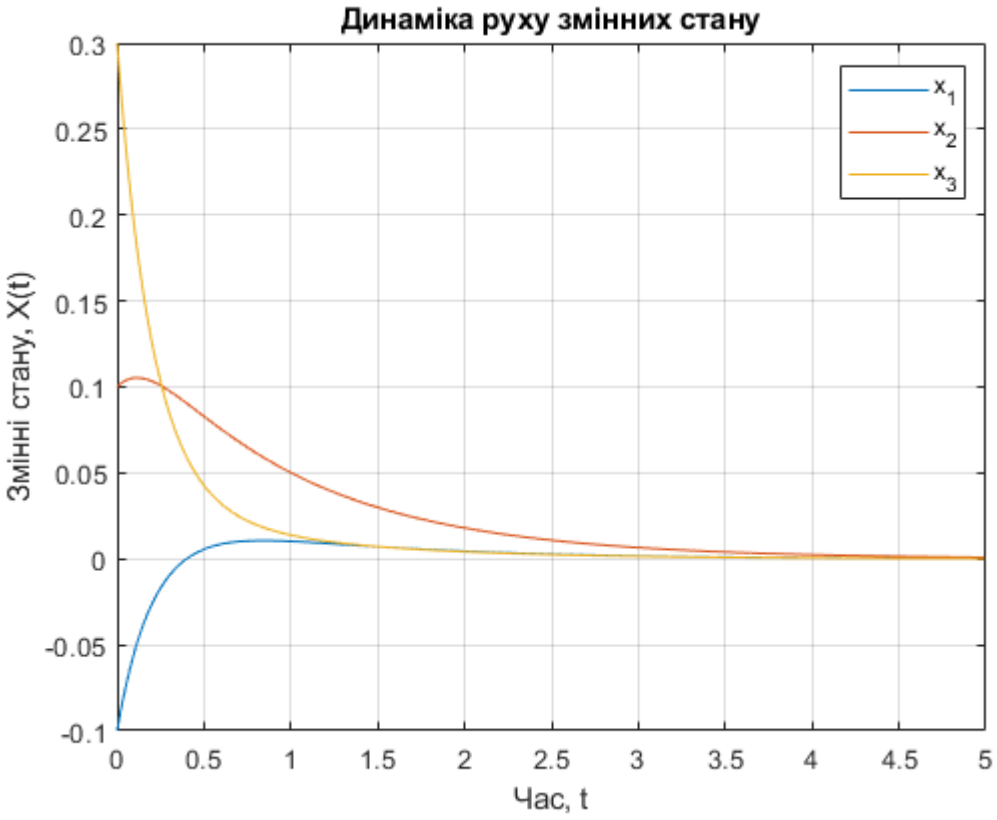
```
x = zeros(SA,n); u = zeros(SB,n-1); y = zeros(SC,n);
% Формування початкового стану
x(:,1) = x0;
% Знаходження оптимального керування u та змінних стану x
for i = 1:n - 1
    u(:,i) = K*C*x(:,i);
    x(:,i + 1) = (A*x(:,i) + B*u(:,i))*dt + x(:,i);
    y(:,i) = C*x(:,i);
end
% Побудова динаміки змінних стану
plot(0:dt:tf-dt,x), grid
title('Динаміка руху змінних стану'); xlabel('Час, t');
ylabel('Змінні стану, X(t)'); legend('x_1','x_2','x_3')
% Побудова вектора керування
figure(2)
plot(dt:dt:tf-dt,u), grid
title('Динаміка зміни керування'); xlabel('Час, t'); ylabel('Керування, U(t)');
legend('u_1','u_2')
% Побудова виходу
figure(3)
plot(0:dt:tf-dt,y), grid
title('Вихід системи'); xlabel('Час, t'); ylabel('Вихід, Y(t)'); legend('y_1','y_2')
```

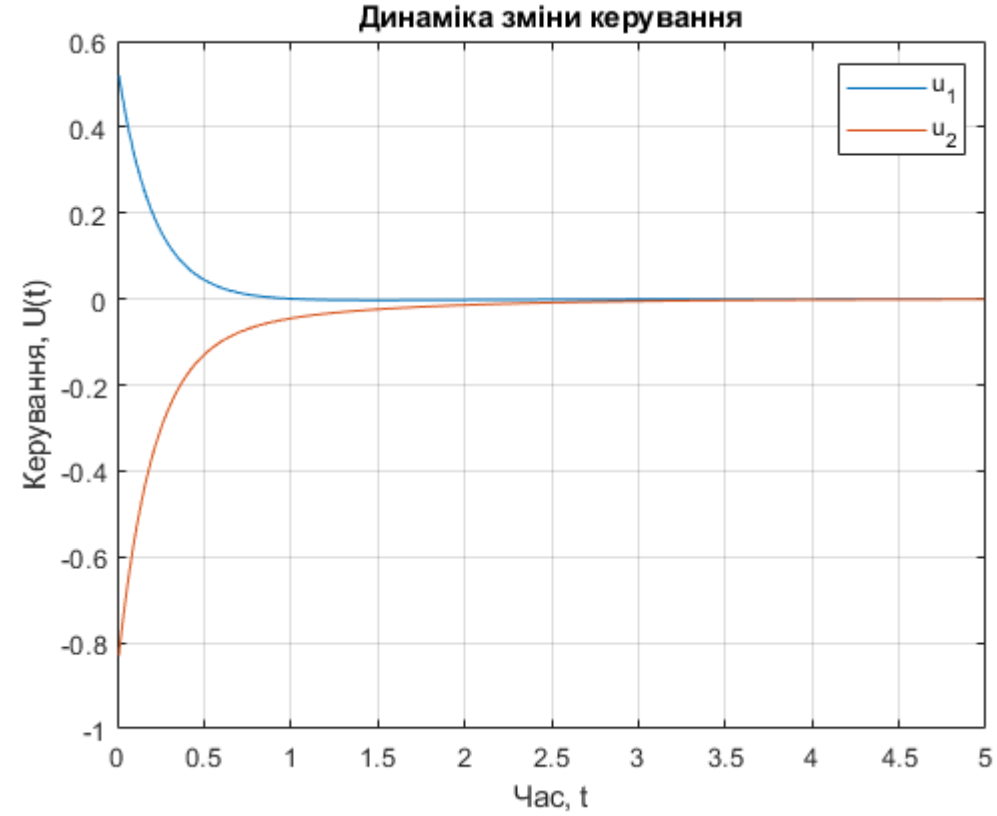
Результат виконання програми

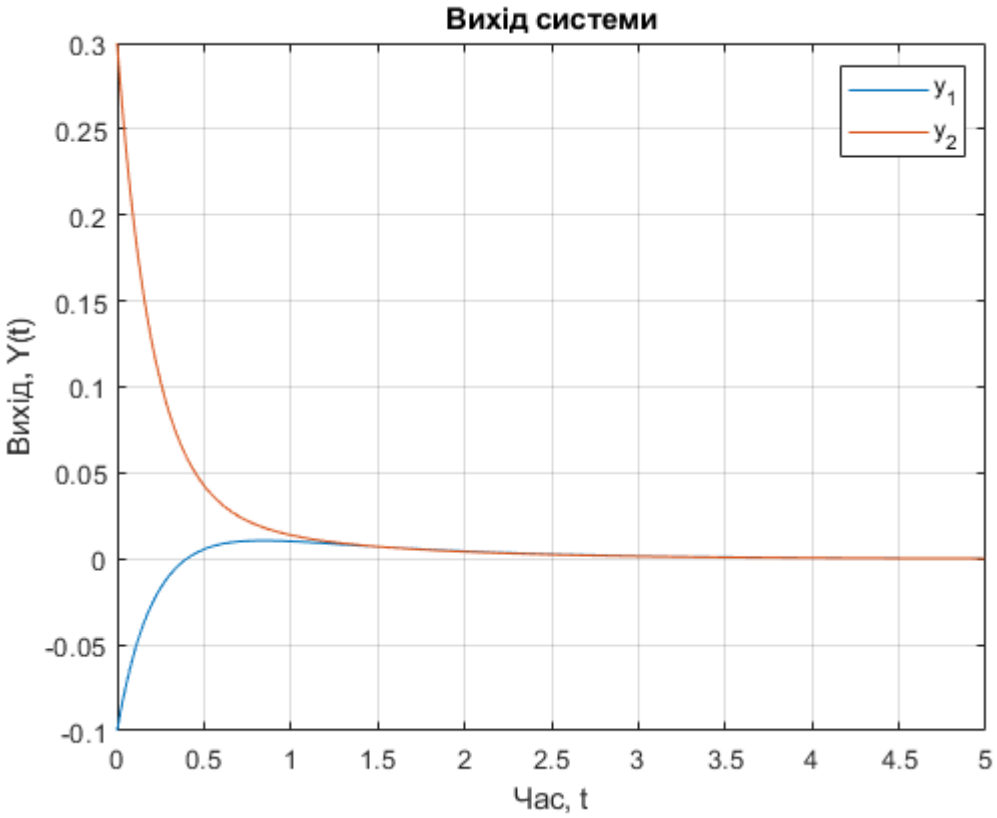
```
Матриця A:

A =

   -0.5000         0         0
         0   -2.0000   10.0000
         0    1.0000   -2.0000

Матриця B:

B =

     1     0
    -2     2
     0     1

Матриця C:

C =

     1     0     0
     0     0     1

Вектор початкових умов X0:

x0 =

   -0.1000
```

```
    0.1000
    0.3000

Ініціалізація матриці регулятора:

K0 =

    -1    -1
    -1    -1

Значення goal для характеристичних значень:

goal =

    -5    -3    -1

Ваги по однаковій частці:

weight =

     5     3     1

Нижня межа для регулятораі:

lb =

    -4    -4
    -4    -4

Верхня межа для регулятораі:

ub =

     4     4
     4     4

Час спостереження tf:

tf =

     5

Довжина кроку dt:

dt =

    0.0100

Кількість кроків n:

n =

   500


                Attainment        Max     Line search     Directional
 Iter F-count        factor    constraint   steplength      derivative   Procedure
    0      6             0       1.88521
    1     13         1.031       0.02998            1           0.745
    2     20        0.3525       0.06863            1          -0.613
    3     27        0.1528     -0.009105            1           -0.22    Hessian modified
```

```
     4      34         0.02684        0.03723            1          -0.166    Hessian modified
     5      41       3.123e-17       0.005703            1          -0.116    Hessian modified
     6      48       8.184e-19      9.592e-06            1       -2.62e-15    Hessian modified
     7      55       3.193e-23      4.531e-11            1       -4.33e-14    Hessian modified

Local minimum possible. Constraints satisfied.

fgoalattain stopped because the size of the current search direction is less than
twice the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.


K =

   -1.5953    1.2040
   -0.4201   -2.9047


fval =

   -5.0000
   -3.0000
   -1.0000


attainfactor =

   3.1930e-23
```







В даному випадку програма намагається підігнати ціль до goal. Коефіцієнт досягнення(-6.9846e-22) вказує, що ціль точно була досягнута.

Завдання

1. Синтезувати систему оптимального керування об`єктом

X'=AX+BU

Y = CX

Варіанти:

а) $A=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -10 & -60 & -41 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$; $C=\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$,

б) $A=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -3 & -6 & -4 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$; $C=\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$,

в) $A=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -6 & -4 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$; $C=\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$,

г) $A=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -5 & -10 & -8,5 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$; $C=\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$,

д) $A=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -3,5 & -6 & -3,5 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$; $C=\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$,

е) $A=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -5 & -6 & -4,5 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 0 \\ 1,5 \end{bmatrix}$; $C=\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$,

ж) $A=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -3 & -8 & -4,5 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 0 \\ 0,5 \end{bmatrix}$; $C=\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$,

з) $A=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -6 & -2 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$; $C=\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$,

к) $A=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -6 & -3 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$; $C=\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$,

л) $A=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -25 & -10 & -3,5 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$; $C=\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$.

2. Дослідити вплив початкових умов на систему.

3. Сформувати систему у відхиленнях і розвязати її.
4. Дослідити вплив початкової матриці регулятора на систему.
5. Дослідити вплив на систему верхньої та нижньої межі для регулятора.

Комп’ютерний практикум № 5

5. Задача мінімакса.

5.1. Комбінаторна оптимізація

При всіх нетривіальних задачах оптимізації область, що допускається $X \in R^n$ містить нескінченно багато кандидатів для шуканого мінімуму. Навпроти, для задачі комбінаторної оптимізації мова йде про те, щоб серед маси кандидатів вибрати ті, котрі мінімізують значення функції якості. Може здатися, що задачі оптимізації цього роду легко розв’язуються. (Адже необхідно тільки порівняти між собою обчислювальну безліч кандидатів). При пошуку ефективних точних методів рішення широкого класу дискретних задач треба враховувати можливість відсутності таких методів, визнавши, що існують важкорозв’язувані задачі. У переборних задачах, як правило, мається кінцева безліч варіантів, серед яких потрібно знайти рішення. Але з ростом число векторів швидко росте і задача стає практично нерозв'язаною. Крім того, як правило виключається також систематичне використання викладеними раніше методиками, що дають недостатню ефективність і однозначні уявлення про градієнти. Для випадків комбінаторної оптимізації повинні бути розвинуті відповідні нові методики, що, при нових умовах задач, забезпечують систематичний і ефективний пошук мінімуму.

Корисним методом представлення (зображення) об'єднання багатьох подібних груп задач комбінаторної оптимізації є теорія графів, що за останні роки значно розвинута і формалізована. Але докладний виклад методу комбінаторної оптимізації не вкладається в рамки цього посібника [75,141,187]. Нашою задачею тут, в більшій мірі, є продемонструвати за допомогою прикладів

вигляд і застосування постановки комбінаторних задач, а також їхня відмінність від постановки аналітичних задач.

Приклад 5.1.

(Вузлове фарбування графа). Розглянемо планування в часі визначеної кількості подій, без небажаного їхнього перетинання (перехлеста). Нехай, наприклад, $x_1,...,x_6$ шість одногодинних лекцій, що повинні прослухати сім груп слухачів, яких цікавлять такі лекції

$\{x_1,x_2\},\{x_1,x_4\},\{x_3,x_4\},\{x_2,x_6\},\{x_4,x_5\},\{x_5,x_6\},\{x_1,x_6\}$.

Кожна лекція, у даному випадку, збігається в часі з іншими лекціями. Проводиться пошук оптимального планування часу для мінімізації загальної тривалості так, щоб кожній групі слухачів дати можливість взяти участь (прослухати) ті лекції, що їх цікавлять. Очевидно, що при постановці цієї задачі є безліч альтернативних планів. Найгірше з точки зору мінімізації загальної тривалості, планування в часі, полягає в тому, щоб розташувати лекції послідовно $(x_1),(x_2),...,(x_6)$ (загальна тривалість 6 годин). Більш гарний варіант (розв'язок), з загальною тривалістю 4 годин, виглядає як $(x_1x_3),(x_2x_4),(x_5),x_6$. Який же мінімальний загальний час?

Для систематичного розгляду цієї задачі, розглянемо граф з шістьма вузлами $x_1,...x_6$. Два вузли зв'язують між собою лінією (прямою гранню), якщо обидва вони належать до області інтересів визначеної групи слухачів.

Тим самим, задача зводиться до більш загальної задачі вузлового фарбування графа:

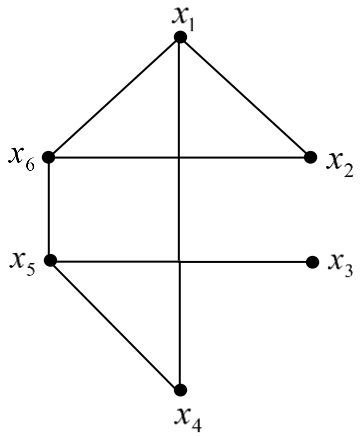


Рис. 5.1. Граф прикладу 5.1

Скільки потрібно мінімально фарб, щоб зв’язані один з одним вузли мали завжди різні (з’єднання) кольори?

Розв’язок цієї задачі, для нашого прикладу, буде: 3 кольори (тобто загальна тривалість 3 години), а саме: $(x_1),(x_2,x_5),(x_3,x_4,x_6)$.

Приклад 5.2.

(Задача найкоротшого шляху). Розглянемо граф, грані якого (направлені або ненаправлені) оцінені певним чином (наприклад довжина або тривалість у часі). Постановка задачі полягає в тому, щоб визначити найбільш сприятливі з точки зору витрат з’єднання між двома вузлами графа. Ця задача пошуку найкоротшого шляху (відрізка) з’єднання, при наявності безлічі альтернатив.

Можна ставити зворотню задачу, тобто шукати замість найкоротшого шляху, між двома вузлами,найдовший. Постановка задачі пошуку найдовшого шляху знаходиться в тісному зв'язку з методом сітьового планування і може, у багатьох випадках приводити до відповідних розв’язків з меншою витратою обчислень, ніж при лінійному програмуванні.

Приклад 5.3.

(Мінімальне оснащення графа) (граф мінімального оснащення) П’ять населених пунктів $a_1,...,a_5$ повинні бути з'єднані з очисною установкою за допомогою відповідних водовідвідних каналів. Можливі варіанти з'єднань показані на графі рис. 5.2.

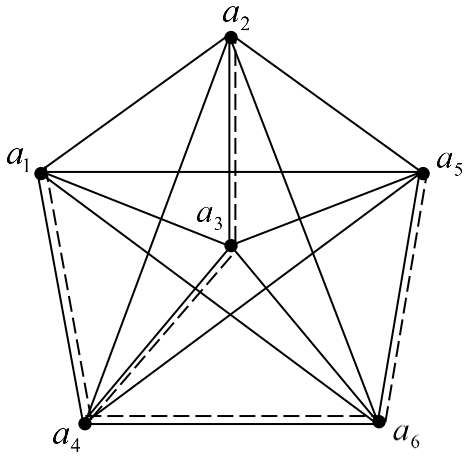


Рис. 5.2. Графи з'єднань до прикладу 5.3

Вартість (витрати на) з’єднання до прикладу 5.3 **Таблиця 5.1**

	a_2	a_3	a_4	a_5	a_6
a_1	1307.2	683.4	850	1790.8	1409.7
a_2		490	1040.3	860	1000
a_3			561	840	864
a_4				1309.8	673.2
a_5					620

Конструктивні витрати на окремі грані (види) з’єднань наведені в таблиці 5.1. Ведеться пошук варіанта прокладки водовідвідних каналів, що вимагає мінімальних витрат.

Ця задача розв’язується за допомогою загальної задачі визначення мінімальної величини оснащення графа. На рис. 5.2 підкреслене рішення вищенаведеного прикладу. У цьому випадку вартість (величина) витрат = 3027.6.

Приклад 5.4.

Організаційні задачі в подвійних графах. Групі студентів $S_1,...,S_5$ повинні бути видані теми дипломних робіт. На кафедрі лежить перелік шести робочих тем $T_1,...,T_6$. Кожен студент вибрав наступні теми, що його цікавлять.

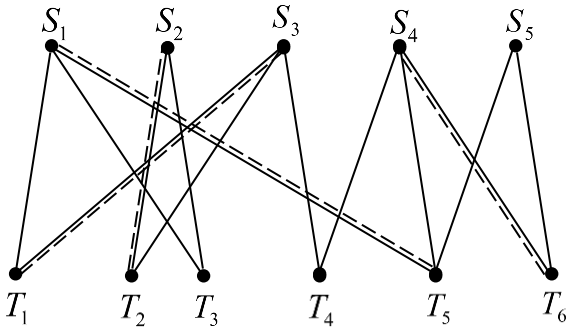


Рис. 5.3. Подвійний граф до прикладу 5.4

$$S_1 : T_1, T_3, T_5$$

$$S_2 : T_2, T_3$$

$$S_3 : T_1, T_2, T_4$$

$$S_4 : T_4, T_5, T_6$$

$$S_5 : T_5, T_6$$

Вимагається знайти розподіл тем, при якому можливо більше число студентів одержить бажану тему.

Для формалізації цієї проблеми вузлами $S_1,...,S_5$ і $T_1,...,T_6$, причому вузли S_i пов'язують з вузлами T_j, у роботі над якою зацікавлений студент S_i (див. рис. 5.3). Задача тоді може бути зведена до організаційної задачі в “подвійних” графах, причому необхідно знайти максимальне число граней без загальних вузлів. Граф називається подвійним, якщо для фарбування кожного з його вузлів (див. приклад 5.4) досить 2-х кольорів.

Неоднозначне рішення розглянутого приклада показане на рис. 5.3 подвійним пунктиром і призводить до забезпечення чотирьох студентів $S_1,...,S_4$ бажаними темами. Студент S_5 змушений буде задовольнитися темою, що йому не бажана.

5.2. Теорія ігор

У процесі попереднього викладення, ми виходили з мінімізації або максимізації функції якості по відношенню до єдиного вирішального параметру. Тепер вже в політиці, економіці, техніці й іграх для 2-х чоловіків зустрічаються ситуації, при яких значення функції якості можуть бути піддані впливу двох істотних параметрів із протилежними (конкуруючими) цілями. Тоді мова йде про теорію з двома гравцями.

Для формалізації цих випадків ми визначаємо функцію якості $f(x_A, x_B)$, при цьому головні змінні x_A встановлюються гравцем А, а головні змінні В встановлюються гравцем В. Тепер гравець А намагається мінімізувати функцію f_1, в той час як гравець В має за мету максималізувати цю функцію. Обидва

гравця знають мету супротивника. Для наочності ми приймемо, що функція $f(x_A, x_B)$ представляє грошові відношения й А повинний платити В, після того, як буде встановлено, після завдання x_A^* і x_B^* значення $f(x_A^*, x_B^*)$. Отже, гравець А хоче $f(x_A, x_B)$ мінімізувати, однак, він знає, що супротивник прагне до зворотного результату. Якщо гравець А повинний грати першим, він, внаслідок цього, спробує мінімізувати виграш свого супротивника вирішуючи наступну мінімаксну задачу:

$$\min_{X_A} \max_{X_B} f(x_A, x_B)$$

Якщо б гру починав гравець В, то він повинний був-би вирішувати задачу мінімакса

$$\min_{X_B} \max_{X_A} f(x_A, x_B)$$

Тим самим для цього простого випадку ми одержимо сідлову точку

$$f(x_A^*, x_B) \le f(x_A^*, x_B^*) \le f(x_A, x_B^*)$$

що встановлює оптимальну стратегію для двох гравців.

Задачі теорії ігор стають цікавіші, але і складнішими, якщо розглядаються багатоступінчасті ігри (наприклад, шахи й інші ігри для двох) або так звані диференціальні ігри. Для останніх мова йде про мінімаксималізацію функції якості, з урахуванням динамічних граничних умов, на увазі динамічної оптимізації. Відомим прикладом з теорії диференціальних ігор є проблема переслідування-втікання, при якій рухомий об'єкт з визначеними динамічними властивостями (максимальна швидкість, здатність відхилятися і т.п) переслідує в 2-х або 3-х мірному просторі, інший рухомий об'єкт, що намагається врятуватися втечею. Спеціальна література по теорії ігор велика і представляє широкий спектр можливих застосувань.

5.3. Розв`язання задачі мінімакса

Використовується метод послідовного квадратичного програмування (SQP). Модифікації торкаються лінійного пошуку і матриці Гессе. Лінійний

пошук закінчується, коли функція *f(X)* показує поліпшення. Так само використовується модифікована матриця Гессе, що дає виграш для спеціалізованої структури даної задачі.

Приклад програми [1]

Знайти значення *X*, що мінімізує максимальне значення

$f(X)=[\ f_1(X)\ f_2(X)\ f_3(X)\ f_4(X)\ f_5(X)]$,

де

$f_1(x) = 2x_1^2+x_2^2-48x_1-40x_2+304;$

$f_2(x) = -x_1^2-3x_2^2;$

$f_3(x) = x_1+3x_2-18;$

$f_4(x) = -x_1-x_2;$

$f_5(x) = x_1+x_2-8,$

починаючи з точки $X^{(0)}=[0,1\ 0,1]^T$.

Спочатку запишемо М- файл (процедуру) для досліджуваної функції та збережемо її як “myfun.m”.

```
function f = myfun(x)
f(1)= 2*x(1)^2+x(2)^2-48*x(1)-40*x(2)+304;
f(2)= -x(1)^2-3*x(2)^2;
f(3)= x(1)+3*x(2)-18;
f(4)= -x(1)-x(2);
f(5)= x(1)+x(2)-8;
```

Програма, що розв’язує задачу

```
% Початкова точка
x0 = [0.1; 0.1];
% Параметри опцій оптимізації
options=optimset('LargeScale','off','Display','iter')
;
% Виклик функції оптимізації
[x,fval,maxfval] =
fminimax(@myfun,x0,[],[],[],[],[],[],[],options);
x % точка мінімуму
```

```
fval % значення функції в точці мінімуму
maxfval % максимальне значення фунції
```

Результат виконання програми

```
                Objective        Max     Line search     Directional
 Iter F-count        value    constraint   steplength      derivative   Procedure
    0       4            0        295.23
    1       9       -1.087         36.78            1          -0.222
    2      14     -0.01413        0.8376            1           0.816
    3      19     0.009458      0.002697            1           0.442
    4      24    -0.009026        0.2982            1         -0.0418    Hessian modified
    5      29    -8.635e-06     0.0002849            1           0.494
    6      34    -1.288e-12     4.244e-11            1           0.809    Hessian modified

x =

    4.0000
    4.0000


fval =

    0.0000  -64.0000   -2.0000   -8.0000   -0.0000


maxfval =

   4.1155e-11
```

Завдання

Знайти розв'язок, що мінімізує максимальне значення функцій:

а) $f_1(x) = 2x_1^2 + x_2^2 + 48x_1 + 40x_2$,

$f_2(x) = -x_1^2 - 3x_2^2$,

$f_3(x) = x_1 + 3x_2 - 18$,

$f_4(x) = -x_1 - x_2$.

б) $f_1(x) = (1 - x_1)^4 + (2{,}5x_1 - x_2)^2$,

$f_2(x) = (5 - x_1)^4 + (2{,}5x_1 - x_2)^2$,

$f_3(x) = (4 - x_1)^4 + (x_1 - x_2)^2$.

в) $f_1(x) = (1 - x_1)^4 + (x_1 - x_2)^2$,

$f_2(x) = (4 - x_1)^4 + (x_1 - x_2)^2$.

Комп’ютерний практикум № 6

6. Алгоритми великої розмірності

6.1. Методи, основані на використанні довірчих областей

Достатньо значне число методів, що використовуються в пакеті Optimisation Toolbox основані на методі довірчих областей, що є простим і в той же час цілком ефективним прийомом розв'язку оптимізаційних задач.

Для пояснення підходу при розв'язанні задач оптимізації при використанні довірчих областей розглянемо задачу оптимізації при відсутності обмежень min $f(X)$, де $f(X)$ цільова функція. Припустимо, що пошук знаходиться в точці $X \in \mathcal{R}^n$ і необхідно поліпшити розв'язок, тобто перейти в точку з меншим значенням цільової функції. Основна ідея полягає в тому, що функцію $f(X)$ необхідно представити у вигляді деякої апроксимації за допомогою більш простої функції q, яка достатньо точно відображає поведінку функції $f(X)$ в околі N поблизу точки X. Цей окіл і називається довірчою областю. Пробний крок S розраховується за допомогою мінімізації у вибраному околі N. Такий підхід складає так звану підзадачу довірчої області

$$\min_S \{q(S), S \in N\}, \tag{6.1}$$

при виконанні умові $f(X+S) < f(X)$ поточна точка переходить в нове значення $X+S$, в протилежному випадку поточна точка залишається без зміни і довірча область N зазнає певного скорочення і розрахунок пробного кроку повторюється.

Ключовими моментами у визначенні специфічного підходу довірчих областей для мінімізації $f(X)$ є такі питання:

- вибрати і розрахувати апроксимуючу функцію q (визначену в поточній точці X);
- вибрати і відкоригувати довірчу область N;
- точно вирішити підзадачу довірчої області.

В стандартному методі довірчих областей [30] квадратична апроксимація q визначається через використання двох перших членів розкладання Тейлора

функції $f(X)$ в точці X, при цьому окіл N, як правило, приймається сферичної або еліптичної форми. В математичній формі підзадача на метод довірчих областей ставиться як

$$min\left\{\frac{1}{2}S^T HS + S^T g,\ при \|DS\| \le \Delta\right\} \qquad (6.2)$$

де g - градієнт функції $f(X)$ в поточній точці; H - матриця Гессе; D - діагональна масштабуюча матриця; Δ - додатній скаляр і $\|.\|$ - друга норма вектора.

Існують алгоритми для вирішення рівняння (6.2), подібні алгоритми звичайно включають розрахунок повної системи власних значень, а так само метод Ньютона стосовно вагового рівняння

$$\frac{1}{\Delta} - \frac{1}{\|S\|} = 0$$

Подібні алгоритми приводять до точного розв'язку рівнянь типу (6.2) [30]. Проте необхідно враховувати, що при їх розв'язанні потрібен час, пропорційний числу членів від розкладання матриці Гессе H [28], [30]. Саме тому для вирішення задач великої розмірності потрібен пошук різних типів розв'язку. Прийнятий в пакеті Optimization підхід апроксимації полягає у зведенні прийнятої підзадачі з довірчою областю до деякого двовимірного підпростору S [27], [28]. Як тільки підпростір S буде визначений, то проблема за розв'язком рівняння (6.2) стає достатньо тривіальною задачею, навіть, не дивлячись на те, що необхідно знати повну інформацію про власні значення і власний вектор (оскільки у вибраному підпросторі дана задача є однозначно двовимірною). Вся основна діяльність тепер зводиться до визначення необхідного підпростору.

Даний двовимірний підпростір S визначається з метою формування процесу з попередньої обробки спряжених градієнтів, як це буде представлено нижче. В даному алгоритмі визначається величина $S = (S_1, S_2)$, де S_1, є напрям градієнта g, а S_2 є або напрям апроксимації Ньютона, тобто розв'язком рівняння

$$HS_2 = -g, \qquad (6.3)$$

або напрям від'ємної кривизни

$$S_2^{\mathrm{T}} HS_2 < 0. \qquad (6,4)$$

Внутрішній підтекст подібного вибору S полягає в тому, що б забезпечити глобальну збіжність (через напрямок найшвидшого спуску або через напрямок з від'ємною кривизною) і досягти швидкої локальної збіжності (через крок Ньютона, якщо той звичайно існує).

Основні особливості даного підходу в пакеті Optimization Toolbox стосовно мінімізації при відсутності обмежень шляхом використання положення про довірчі області можна представити як:

- Сформулювати двовимірну підзадачу на основі методу довірчих областей.
- Вирішити рівняння (6.2) з метою визначення пробного кроку S.
- Якщо $f(X+S) \leq f(X)$, то $X=X+S$.
- Встановити Δ.

Всі ці чотири кроки повторюються до тих пір, поки не буде досягнута необхідна збіжність. Розмірність довірчої області A встановлюється згідно стандартних правил. Зокрема, вона зменшується, якуцо пробний крок не приймається, тобто $f(X+S) \geq f(X)$ [29].

Спряжені градієнти з попередньою обробкою даних

Одним з найпоширеніших шляхів розв'язку симетричної додатньо визначеної системи лінійних рівнянь великої розмірності $HP = -g$ є метод спряжених градієнтів з попередньою обробкою даних (PCG). В даній ітераційній процедурі потрібна наявність можливості розрахунку добутків матричних векторів вигляду HV, де є V деякий довільний вектор. Симетрична додатньо визначена матриця M, що є попередньою для H. Таким чином $M = C^2$, де $C^{-1}HC^{-1}$ добре обумовлена матриця або матриця з кластеризованими власними значеннями.

Даний алгоритм знайшов застосування в пакеті Optimization Toolbox і далі називається як алгоритм PCG.

```
% Ініціалізація
r = -g; p = zeros(n,1);
% попередня підготовка
Z = M\r: inner1 = r'*z; inner2 = 0; d = z;
```

```
 % Ітерація зі спряженими градієнтами
for k = 1:kmax
    if k > 1
    beta inner1/inner2;
    d = z + beta*d;
    end
    w = H*d; denom = d'*w;
    if denom <= 0
        p = d/norm(d);
% напрямок з негативною / нульовою кривизною
break
% вихід якщо визначена негативна / нульова кривизна
    else
        alpha = inner1/denon;
        p = p + alpha*d;
        r = r - alpha*w;
    end
    z = M\r;
    if norm(z)<-tol
 % вихід якщо рівняння нр=-9 дозволено в межах заданої
точності
        break
    end
inner2 = inner1;
inner1 = r'*z;
end
```

В контексті процедури мінімізації можна припустити, що матриця Гессе H є симетричною. Проте матриця H гарантовано є додатньо визначеною тільки в околі p помітно визначеним процесом мінімізації. Алгоритм PCG використовується тоді, коли виявляється напрям з від'ємною (або нульовою) кривизною, тобто $D^T HD \leq 0$. Результуючий напрям PCG P, є або напрям з від'ємною кривизною або є розв'язком апроксимації (параметр tol визначає вид апроксимації) системи Ньютона $HP=-g$, а інакше, варіант напряму P використовується для визначення двовимірного підпростору, який далі використовується в методі довірчих областей.

Задачі з лінійними обмеженнями

Накладення лінійних обмежень приводить до ускладнення ситуації, розглянутої у разі мінімізації при відсутності обмежень. Проте більшість методів, як це було описано раніше, можуть привести до абсолютно ясних і

ефективних розв'язків. Стосовно рамок пакета Optimization Toolbox, методи великої розмірності приводять до цілком виправданих ітераційних процедур:

- Метод проекцій використовується у разі обмежень типу лінійної рівності;
- Метод віддзеркалень використовується в спрощеному випадку обмежень типу нерівність.

Обмеження типу лінійної рівності

В узагальненому випадку, задача мінімізації за наявності лінійних обмежень типу рівності може бути записаний як

$$\min\{f(X), \textit{ при } AX = b\}. \tag{6.5}$$

Тут A матриця розмірністю $m \times n$ ($m \leq n$). В пакеті Optimization проводиться попередня обробка матриці A з метою визначення деякої певної лінійної залежності за допомогою застосування процедури на основі використання LU розкладання матриці A^{T} [29]. Тут передбачається, що матриця A має ранг m, що використовується для вирішення рівняння (7.5), метод відрізняється від підходу за наявності обмежень по двох аспектах. По-перше, початкова припустима точка X_0 розраховується шляхом використання деякого кроку для розрідженого методу найменших квадратів, так що $AX_0, = b$. По-друге, Алгоритм PCG замінюється на метод спряжених градієнтів з проведеною попередньою обробкою даних (RPCG) [29]. Ключовим моментом в застосуванні лінійної алгебри є розв'язок системи рівнянь у формі

$$\begin{bmatrix} C & \tilde{\mathrm{A}}^{\mathrm{T}} \\ \tilde{\mathrm{A}} & O \end{bmatrix} \begin{bmatrix} S \\ t \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix}, \tag{6.6}$$

де $\tilde{A}$ являється деякою апроксимацією від A (невеликі ненульові значення матриці A встановлюються в нуль, що забезпечує збереження наявного рангу) і C буде розріджена симетрична додатно визначена апроксимація матриці H, тобто $C = H$ [29].

Обмеження типу нерівність

Задача з обмеженнями типу нерівність має вигляд

$$\min\{f(X), \textit{ при } L \leq X \leq U\}, \tag{6.7}$$

де L - вектор для нижньої межі і U - вектор для верхньої межі. Деякі (тобто не всі) компоненти вектора L можуть бути рівні - ∞ і деякі (тобто не всі) компоненти вектора U можуть бути рівні ∞. В даному методі генерується деяка послідовність істотно допустимих точок. Для досягнення припустимості і при збереженому режиму стійкої збіжності використовуються наступні дві процедури. По-перше модифікований крок Ньютона 3 урахуванням масштабних ефектів використовується замість кроку Ньютона для умови без обмежень (використовувався для визначення двовимірного підпростору S). По-друге, для збільшення розміру кроку використовується процедура віддзеркалень. Модифікований крок Ньютона з урахуванням масштабування походить від розгляду необхідних умов оптимальності Куна-Таккера для рівняння (6.7).

$$[D(X)]^{-2}g = 0, \tag{6.7}$$

де

$D(X) = diag\ [|v_k|^{-1/2}]$

і вектор $V(X)$ далі визначається для кожного $1 \leq I \leq n$:

якщо $g_i < 0$ і $u_i, < \infty$, то $v_i = x_i, -u_i$;

якщо $g_i \geq 0$ і $l_i, < -\infty$, то $v_i = x_i, -l_i$;

якщо $g_i < 0$ і $u_i, = \infty$, то $v_i = -1$;

якщо $g_i < 0$ і $l_i, = -\infty$, то $v_i = 1$;

Система нелінійних рівнянь (6.8) є така, що усюди не диференціюється. Властивість, що не диференціюється має місце при умові $v_i = 0$. Такі точки можна обійти шляхом підтримки строгої припустимості, тобто шляхом відповідного обмеження $L < X < U$.

Модифікований крок Ньютона з урахуванням масштабування S_k^N стосовно рівняння (6.8) визначається як розв'язок наступної системи лінійних рівнянь

$$\widehat{M}DS^N = -\ \hat{g}, \tag{6.9}$$

де

$$\hat{g} = D^{-1}g = diag[V^{1/2}]g \quad (6.10)$$

і

$$\widehat{M} = D^{-1}HD^{-1} + diag(g)J^{v}. \quad (6.11)$$

В даному випадку J^v грає роль Якобіана для $|v|$ кожної діагоналі для діагональної матриці J^v рівний або 0, -1, або 1. У випадку, якщо компоненти векторів L і U є кінцевими, то $J^v = diag\ (sign(g))$. В якійсь точці, де $g_i = 0$, величина v_i може бути і що не диференціюється. В таких точках проводиться визначення J_u^V=0. Не лиференційованість даного типу не може бути причиною помилки, оскільки для цих компонентів значення, що приймаються и не є істотними. Далі хоча і величина $|v_i|$ в цих точках може бути і перервною, але функція $|v_i|g_i$, є безперервною. По-друге, для збільшення величини розміру кроку використовуються так звані віддзеркалення. Крок віддзеркалення (одиничний) задається таким чином. Нехай крок P перетинає граничну межу, то далі проводиться аналіз першого P граничної межі, що перетинається. Припустимо, що це є i-ю обмежувальною межею (або i-а верхня межа або і-а нижня межа). Далі для i-ї компоненти приймається крок віддзеркалення $P^R = P$, де $p_1^R = - P_i$.

Нелінійний метод найменших квадратів

Важливим випадком для функції (X) є нелінійна задача на метод найменших квадратів

$$f(x) = \sum_i f_i^2(X) = \frac{1}{2}\left\| \mathrm{F(X)} \right\|_2^2 \quad (6.12)$$

де $F(X)$ є функція з компонент векторів з i-м компонентом $F(X)$, рівним, $f_i(X)$. Для вирішення задач даного типу основний метод який використовується є тим же самим. Проте, структура нелінійної задачі на метод найменших квадратів використовується для підвищення ефективності. Зокрема, напрям апроксимації Ньютона-Гаусса, тобто розв'язок S для

$$\min \left\| JS + F \right\|_2^2, \quad (6.13)$$

де J Якобіан від $F(X)$, що використовується для поліпшення визначення двовимірного підпростору S. Другі похідні компонент функції $f_i(X)$ не використовуються.

На кожній ітерації даний метод спряжених градієнтів з попередньою підготовкою даних використовується для вирішення апроксимації нормалізованих розв'язків, тобто

$$J^T JS = - J^T F$$

хоча нормалізовані рівняння і не сформовані у явному вигляді.

Квадратичне програмування

В даному випадку шукана функція (X) є квадратичним рівнянням

$$q(X)=\frac{1}{2}X^T HX + f^T(X)X. \qquad (6.14)$$

Замість зведення кроку пошуку до (якщо це можливо) кроку з віддзеркаленням, як це було у разі нелінійної мінімізації, на кожній ітерації проводиться кусковий лінійний пошук з віддзеркаленням [28].

Лінійний метод найменших квадратів

В даному випадку шукана функція $f(X)$ має наступний вигляд

$$f(X)=\frac{1}{2}\left\| CX+D \right\|_2^2. \qquad (6.15)$$

В даному алгоритмі генеруються безумовно припустимі ітерації, що збігаються, в граничному випадку, до якогось локального значення. Кожна ітерація включає розв'язок апроксимації лінійної системи великої розмірності (порядку n, де n є розмірність X). Ітераційні матриці мають структуру таку ж, як матриця С. Зокрема, метод спряжених градієнтів з попередньою обробкою даних використовується для вирішення апроксимації нормалізованих рівнянь, тобто

$$C^T CX = -C^T D$$

хоча нормалізовані рівняння в явному вигляді і не формуються.

Замість зведення кроку пошуку до (якщо це можливо) кроку з віддзеркаленням, як це було у разі квадратичної мінімізації, на кожній ітерації проводиться кусковий лінійний пошук з віддзеркаленням [28]. Кінець кінцем, лінійні системи є реалізацією методу Ньютона стосовно розв'язку в рамках оптимальності першого порядку, що приводить до певних строго локальних швидкостей збіжності.

Слід відзначити, що більшість функцій оптимізації за умовчанням використовують алгоритми великої розмірності.

Відкриття графічного інтерфейсу Optimization Tool здійснюється з командного рядка MATLAB

>> optimtool

На екрані з'являється нове вікно (рис. 6.1).

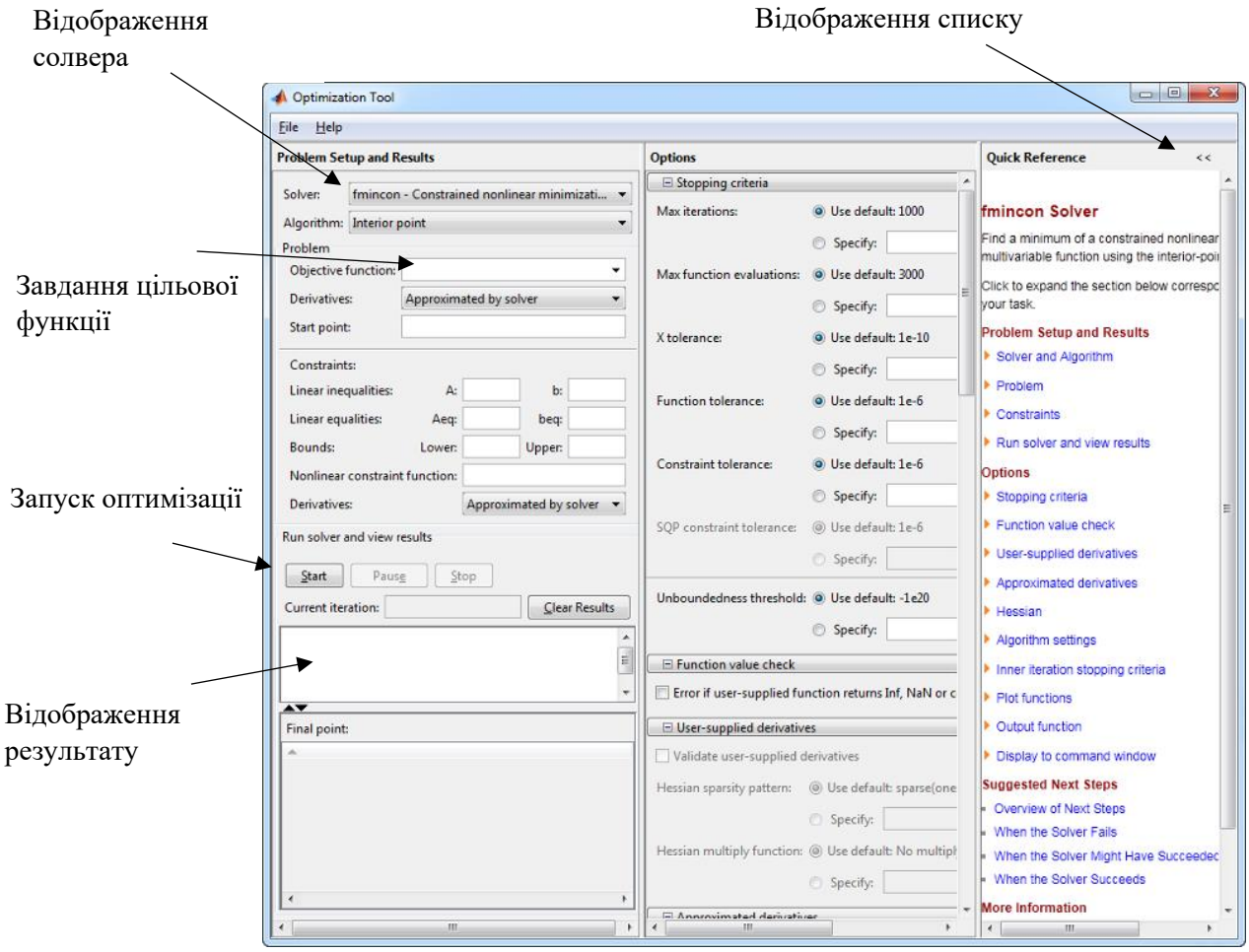


Рис. 6.1. Графічний інтерфейс Optimization Tool

Цільова функція задається у вигляді М файлу. Для більш детальної інформації можна скористатися "відображенням опису".

6.2. Розв’язання систем нелінійних рівнянь

Під розв’язанням деякої нелінійної системи рівнянь $F(X)$ розуміється знаходження такого розв’язку, коли кожне рівняння в даній нелінійній системі буде дорівнювати нулю. Таким чином, маємо n рівнянь та n невідомих і потрібно знайти такі $X \in \Re^n$, що буде справедливе $F(X)=0$, де

$$F(X)=\begin{bmatrix} F_1(X) \\ F_2(X) \\ \vdots \\ F_n(X) \end{bmatrix}. \qquad (6.16)$$

Одним з можливих підходів до розв’язання даної задачі полягає у використанні нелінійного методу найменших квадратів [3,11]. Оскільки ми припустили, що корені системи існують, то вона повинна мати і найменшу нев'язку і тому в даному випадку може бути ефективним метод Ньютона-Гаусса. При цьому на кожній ітерації необхідно вирішувати деяку лінійну задачу за допомогою методу найменших квадратів, і визначати напрям пошуку. Також може бути використано метод Левенберга-Марквардта**.**

Метод ламаних довірчих областей

Інший підхід в розв’язанні лінійної системи рівнянь полягає в знаходження напряму пошуку, а саме, згідно методу Ньютона потрібно знайти такий напрям пошуку D_k, що

$$J(X_k)D_k = -F(X_k)$$
$$X_{k+1} = X_k + D_k$$

де $J(X_k)$ Якобіан розмірністю $n \times n$

$$J(X_k) = \begin{bmatrix} \nabla F_1(X_k)^T \\ \nabla F_2(X_k)^T \\ \vdots \\ \nabla F_n(X_k)^T \end{bmatrix}.$$

У методі Ньютона можуть зустрічатися різні труднощі. Якщо $J(X_k)$ буде сингулярною, то крок в методі Ньютона не можна буде навіть визначити. Крім того обчислювальні витрати для точного визначення кроку за Ньютоном D_k можуть бути надмірно великими. Якщо початква точка була вибрана надмірно далеко від точки розв'язку, то метод Ньютона може взагалі не збігатися.

Застосування методики довірчих областей дозволить поліпшити стійкість розрахунків у випадках, коли початкова точка вибрана дуже далеко від точки розв'язку, і долати неприємності у разі сингулярної матриці $J(X_k)$. Для використання стратегії довірчих областей необхідна якась функція вигоди для того, що б визначити, що точка буде краща або гірше чим X_k. Можлива стратегія вибору може полягати в наступному

$$\min_D f(D) = \frac{1}{2} F(X_k + D)^T F(X_k + D)$$

Але мінімум $f(D)$ не обов'язково повинен бути коренем $F(X)$.

Крок за Ньютоном D_k є коренем

$$M(X_k + D) = F(X_k) + J(X_k) D$$

і оскільки це також є мінімумом і для $m(D)$, де

$$\begin{aligned} &\min_D m(D) = \frac{1}{2} \left\| M(X_k + D) \right\|_2^2 = \frac{1}{2} \left\| F(X_k) + J(X_k) D \right\|_2^2 = \\ &= \frac{1}{2} F(X_k)^T F(X_k) + D^T J(X_k)^T F(X_k) + \frac{1}{2} D^T \left(J(X_k)^T J(X_k) \right) D \end{aligned} \tag{6.17}$$

Тоді $m(D)$ буде більш кращою стратегією вибору для функції вигоди, чим $f(D)$, і звідси під задача довірчої області полягатиме в наступному

$$\min_D \left[\frac{1}{2} F(X_k)^T F(X_k) + D^T J(X_k)^T F(X_k) + \frac{1}{2} D^T \left(J(X_k)^T J(X_k) \right) D \right] \qquad (6.18)$$

За умови, що $\|D\| \leq \Delta$. Така внутрішня під задача може бути успішно вирішена за допомогою стратегії ламаних напрямів. Детальніше про метод довірчих областей можна знайти в роботах Кону [4] і Носеда [5].

Реалізація розв'язання нелінійних рівнянь

Реалізація даного методу розв'язання нелінійних рівнянь складається з трьох частин:

- Реалізація методу Ньютона-Гаусса;
- Реалізація методу Левенбрга-Марквардта;
- Реалізація методу ламаних довірчих областей.

Реалізація методу ламаних довірчих областей

Ключовою особливістю даного алгоритму є використання методики ламаних напрямів Пауелла для розрахунку D з мінімізацією згідно рівнянню (6.18). Детально метод приведений в роботі Пауелла [6].

Крок D будується на основі опуклої комбінації кроку Коші (крок уздовж напряму найшвидшого спуску) і кроку Ньютона-Гауса для $F(X)$. Крок Коші визначається як

$$D_C = -\alpha J(X_k)^T F(X_k),$$

де α вибрано з мінімуму згідно рівнянню (6.17).

Крок Ньютона-Гауса знаходиться з рішення рівняння

$$J(X_k)D_{GN} = -F(X_k)$$

Звідси вибраний крок буде

$$D = D_C + \lambda(D_{GN} - D_C),$$

де λ є найбільше значення на інтервалі [0, 1] таке, що $\|D\| \le \Delta$. Якщо J_k є (приблизно) сингулярним, то D якраз буде напрямом Коші.

Алгоритм ламаних напрямів ефективніший, оскільки для нього на кожній ітерації потрібне тільки одне звернення до розрахунку кроку за методом Ньютона-Гауса. Крім того, він може бути і стійкішим за рахунок використання методу Ньютона-Гауса для лінійного пошуку.

Приклади програм

Розв’язок алгебраїчних рівнянь Ріккаті

Система з одним входом і одним виходом описується рівнянням стану, в якому

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -4 & -9 \end{bmatrix}, \; B = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}.$$

Знайти розв’язок алгебраїчних рівнянь Ріккаті за допомогою стандартних функцій MATLAB.

Знайдемо спочатку чисельний розв’язок алгебраїчних рівнянь Ріккаті за допомогою функції призначеної для розв’язання алгебраїчних і матричних

рівнянь. Для цього запишемо процедуру (М – файл) та збережемо його як "*Riccati_R*"

```
function [F A B Q R N] = Riccati_R(P)
A = [0 1 0; 0 0 1; -2 -4 -9]; % Введення матриці A
B = [0 0 2]'; % Введення матриці B
Q = eye(3); % Введення матриці Q
R = 1/2; % Введення матриці R
N = [0 0 0]'; % Введення матриці N
F = A'*P+P*A-(P*B+N)*inv(R)*(B'*P+N')+Q;
```

Запишемо основну програму (М – файл)

```
P0 = eye(3) % Завдання початкової точки
% Параметри опцій оптимізації
options = optimset('Display','iter','NonlEqnAlgorithm','lm');
% Виклик функції оптимізації
[P,fval,exitflag,output,jacobian] = fsolve(@Riccati_R,P0,options)
% P - ров'язок рівняння
% fval - значення функції в точці розв'язку
% exitflag - флаг виходу алгоритма (інформація про стан алгоритма)
% output - інформація структура інформації про оптимізацію
% jacobian - матриця Якобі
```

Результат виконання програми

P0 =

1	0	0
0	1	0
0	0	1

Iteration	Func-count	Residual	Step-size	Directional derivative	Lambda
0	10	655			
1	23	147.435	0.658	-47.3	6.01476e-025
2	37	97.3999	1.22	-0.0171	217.785
3	51	91.4365	8.54	-0.000308	218.541
4	65	76.1504	1.6	-0.0103	22.9044
5	79	47.0195	2.41	-0.015	8.80039
6	93	18.772	1.4	-0.0539	2.57805
7	107	5.48656	1.57	-0.00771	1.07159
8	121	2.18087	1.53	-0.00139	0.417497
9	135	0.292045	1.99	-0.0759	0.165134
10	148	0.0176133	1.19	-6.26e-005	0.0530548
11	162	0.000317052	1.23	-1.35e-006	0.024209

```
    12         175     1.4488e-006         1.05      -3.26e-010      0.0108653
    13         188    1.69155e-009         1.05      -6.19e-014     0.00529928
    14         201    4.95587e-013         1.01      -4.56e-018     0.00258234
    15         202     1.1434e-016            1      -9.79e-013     0.00128319
Optimization terminated: directional derivative along
 search direction less than TolFun and infinity-norm of
 gradient less than 10*(TolFun+TolX).

P =

    2.2364    1.9915    0.1830
    1.9915    4.4685    0.4343
    0.1830    0.4343    0.0994


fval =

  1.0e-008 *

   -0.9490    0.0733    0.2869
    0.0733   -0.1653    0.0816
    0.2869    0.0816   -0.1637


exitflag =

     1


output =

       iterations: 15
        funcCount: 202
         stepsize: 1
     cgiterations: []
    firstorderopt: []
        algorithm: 'medium-scale: Levenberg-Marquardt, line-search'
          message: [1x147 char]




jacobian =

      0         0   -3.4641         0         0         0   -3.4641         0
0
 1.0000         0   -7.4742         0         0         0         0   -3.4641
0
      0    1.0000   -9.7953         0         0         0         0         0   -
3.4641
 1.0000         0         0         0         0   -3.4641   -7.4742         0
0
      0    1.0000         0    1.0000         0   -7.4742         0   -7.4742
0
      0         0    1.0000         0    1.0000   -9.7953         0         0   -
7.4742
      0         0         0    1.0000         0         0   -9.7953         0   -
3.4641
      0         0         0         0    1.0000         0    1.0000   -9.7953   -
7.4742
      0         0         0         0         0    1.0000         0    1.0000  -
19.5907
```

Тепер розв’яжемо алгебраїчні рівняння Ріккаті за допомогою стандартної функції. Програма (М – файл) наведений нижче

```
A = [0 1 0; 0 0 1; -2 -4 -9]; % Введення матриці A
B = [0 0 2]'; % Введення матриці B
Q = eye(3); % Введення матриці Q
R = 1/2; % Введення матриці R
N = [0 0 0]'; % Введення матриці N
E = eye(3); % Введення матриці E (в нашому випадку I)
[P L G report] = care(A,B,Q,R,N,E) % Розв'язок рівнянь Ріккаті
% P - розв'язок
% L - власні значення замкненої системи
% G - матричний коефіцієнт підсилення
% report - RR норма Фробеніуса відносних залишків
```

Результат виконання програми

```
P =

    2.2364    1.9915    0.1830
    1.9915    4.4685    0.4343
    0.1830    0.4343    0.0994


L =

  -9.0083
  -0.3935 + 0.4793i
  -0.3935 - 0.4793i


G =

    0.7321    1.7371    0.3977


report =
```

```
1.3592e-015
```

Розв’язок нелінійної системи рівнянь

Розв’язати систему рівнянь

$$\begin{cases} x_1^4 - \cos x_1 e^{x_2} = 0 \\ 2x_2 - \sin x_2 e^{x_1} = 0 \end{cases}$$

Знайдемо спочатку чисельний розв’язок системи рівнянь за допомогою функції призначеної для розв’язання алгебраїчних і матричних рівнянь. Для цього запишемо процедуру (M – файл) та збережемо його як “*Eq*”

```
function F = Eq(x)
F = [x(1)^4-cos(x(1))*exp(x(2)); 2*x(2)-sin(x(2))*exp(x(1))];
```

Запишемо основну програму (M – файл)

```
x0 = [1 -1]'; % Завдання початкової точки
% Параметри опцій оптимізації
options = optimset('Display','iter','NonlEqnAlgorithm','gn');
% Виклик функції оптимізації
[x,fval,exitflag,output,jacobian] = fsolve(@Eq,x0,options)
% x - ров'язок рівняння
% fval - значення функції в точці розв'язку
% exitflag - флаг виходу алгоритма (інформація про стан алгоритма)
% output - інформація структура інформації про оптимізацію
% jacobian - матриця Якобі
```

Результат виконання програми

```
                                                    Directional
 Iteration  Func-count     Residual      Step-size      derivative
     0          3         0.724549
```

```
     1          10     0.000355977         1.27         -0.0123
     2          16    4.44751e-007            1       -4.63e-007
     3          22    2.20634e-013            1       -2.85e-013
Optimization terminated: directional derivative along
 search direction less than TolFun and infinity-norm of
 gradient less than 10*(TolFun+TolX).

x =

    0.7739
   -0.6903


fval =

  1.0e-006 *

    0.4318
   -0.1849


exitflag =

     1


output =

       iterations: 4
        funcCount: 22
         stepsize: 1.0009
     cgiterations: []
    firstorderopt: []
        algorithm: 'medium-scale: Gauss-Newton, line-search'
          message: [1x147 char]
```

```
jacobian =

    2.2042   -0.3586
    1.3806    0.3282
```

Тепер отримаємо аналітичний розв’язок за допомогою функції *solve* зпишемо М – файл

```
% Отримання розв'язку і знесення його в структуру S
S = solve('x1^4-cos(x1)*exp(x2) = 0','2*x2-sin(x2)*exp(x1) = 0');
% Вилучення інформації з структури S
x1 = S.x1
x2 = S.x2
```

Результат виконання програми

```
x1 =

.77386194903675472502436430883255



x2 =

-.69027767856405687546646609111260
```

Як можна побачити з вище отриманих розв’язків, що результати майже співпадають.

7. Розв’язання системи нелінійних диференційних рівнянь

Для розв’язання диференційних рівнянь та систем рівнянь, призначені спеціальні функції MATLAB, в обчислювальній математиці їх називають

солвери. MATLAB має достатньо великий набір солверів, заснованих на різноманітних числових методах.

Для розв'язання задачі Коші в MATLAB існує сім солверів: ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb.

Задача Коші для диференційного рівняння полягає в знаходженні функції, що задовольняє диференційному рівнянню довільного порядку

$$\frac{d^n y}{dt^n} = f\left(t, y, \frac{dy}{dt}, ..., \frac{d^{n-1} y}{dt^{n-1}}\right)$$

та початковим умовам $y(t)\Big|_{t=t_0} = y_0, \frac{dy}{dt}\Big|_{t=t_0} = y_1, ..., \frac{d^{n-1} y}{dt^{n-1}}\Big|_{t=t_0} = y_{n-1}.$

Приклади програм

Розв'язок диференційного рівняння

Знайдемо розв'язок диференційного рівняння вигляду

$$\frac{d^2 y}{dt^2} + 2\frac{dy}{dt} + 10y = \cos t + e^{-3t}, \quad y(t)\Big|_{t=0} = 0{,}05, \frac{dy}{dt}\Big|_{t=0} = 0{,}1.$$

Представимо дане диференційне рівняння у вигляді системи диференційних рівнянь першого порядку (приведемо до форми Коші)

$$\begin{cases} y_1' = y_2 \\ y_2' = -2y_2 - 10y_1 + \cos t + e^{-3t}, \end{cases} \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 0{,}05 \\ 0{,}1 \end{bmatrix}.$$

Запишемо програму, що виконує порівняння аналітичного (символьного) розв'язоку "Deff_Rivn"

```
function Deff_Rivn
% Початкові умови
```

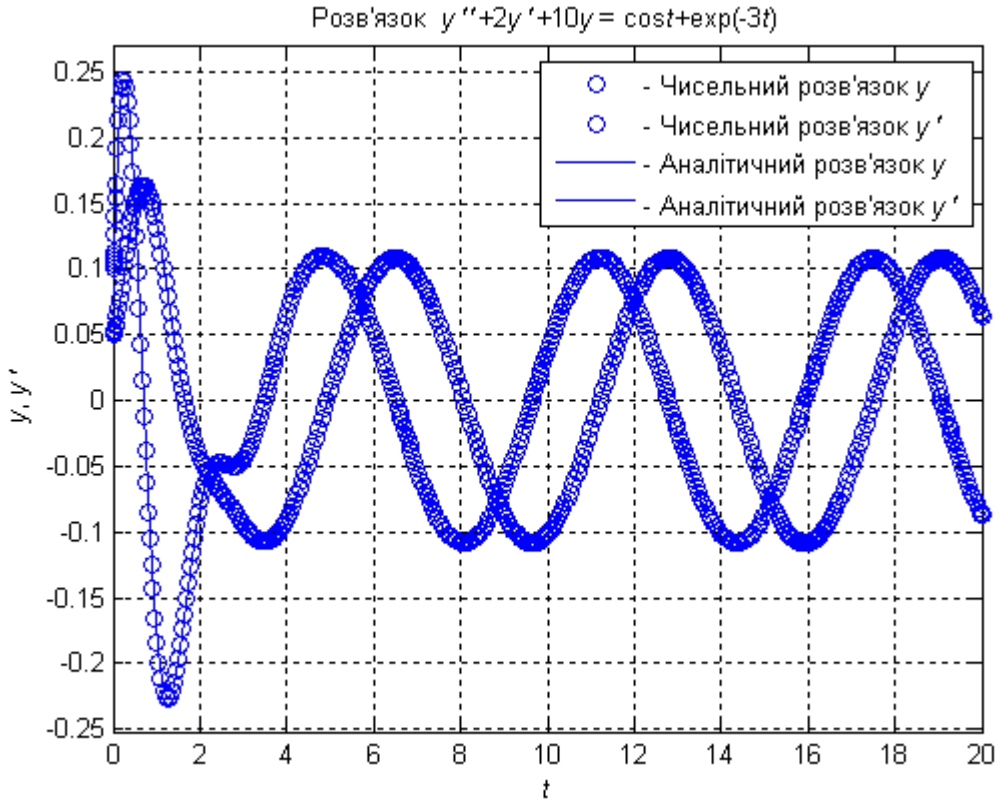
```
Y0 = [0.05 0.1];
% Початковий та кінцевий момент часу
T = [0 20];
% Структура опцій солвера
options = odeset('RelTol',1e-4,'AbsTol',[1e-6 1e-6]);
% Виклик солвера
[T,Y] = ode45(@func,T,Y0,options);
% Аналітичний роз'язок
% Отримання розв'язку і знесення його в структуру S
S = dsolve('Dy1 = y2','Dy2 = -2*y2-10*y1+cos(t)+exp(-3*t)',...
'y1(0) = 0.05','y2(0) = 0.1');
% Вилучення інформації з структури S
y1 = S.y1
y2 = S.y2
% Виведення графіку розв'язку вихідного диференціального рівняння (чисельного
% розв'язку)
plot(T,Y(:,1),'o',T,Y(:,2),'o','Color','blue')
hold on
% Виведення графіку розв'язку вихідного диференціального рівняння (аналітичного
% (символьного) розв'язку)
ezplot(y1,[0 20]), ezplot(y2,[0 20])
% Виведення пояснень на графік
title('Розв''язок  {\ity} \prime\prime+2{\ity} \prime+10{\ity} = ...
cos{\itt}+exp(-3{\itt})')
xlabel('\itt'), ylabel('{\ity}, {\ity} \prime '),
legend('- чисельний розв''язок {\ity}','- чисельний розв''язок {\ity}
\prime',...
    '- аналітичний розв''язок {\ity}', '- аналітичний розв''язок {\ity}
\prime'), grid on
hold off
```

Результат виконання програми

y1 =

*257/4420*exp(-t)*sin(3*t)-*
*587/4420*exp(t)*cos(3*t)+9/85*cos(t)+2/85*sin(t)+1/13*exp(-3*t)*

```
y2 =

376/1105*exp(-t)*sin(3*t)+679/2210*exp(-t)*cos(3*t)-9/85*sin(t)+2/85*cos(t)-
3/13*exp(-3*t)
```



Розв’язання рівнянь Ріккаті

Матричне нелінійне диференціальне рівняння Ріккати використовується для синтезу оптимального закону керування

$$\mathrm{P}' = -P(t)A(t) - A^T P(t)B(t)R^{-1}B^T P(t) - Q(t)$$

де R(t) та Q(t) вагові матриці в критерії якості.

Розглянемо скалярну систему

$$\frac{dx(t)}{dt} = \frac{1}{2}x(t) + u(t), \quad x(t_0) = x_0$$

та функцію вартості

$$J=\frac{1}{2}sx^2\left(t_f\right)+\frac{1}{2}\int_{t_0}^{t_f}\left[2x^2\left(t\right)+u^2\left(t\right)\right]dt.$$

Рівняння Ріккаті матиме вид

$$\frac{dp(t)}{dt}=p(t)+p(t)^2-2,\quad p\left(t_f\right)=s.$$

Розв'язок цього рівняння можна записати у вигляді

$$p(t)=-0{,}5+1{,}5th\left(-1{,}5t+\xi_1\right),$$

або

$$p(t)=-0{,}5+1{,}5cth\left(-1{,}5t+\xi_2\right),$$

де ξ_1 і ξ_2 обираються так, що $p\left(t_f\right)=s$.

Наприклад, якщо $s=0,\ t_f=1$ то $\xi_1=1{,}845$ радіан, що дасть

$$p(t)=-0{,}5+1{,}5th\left(-1{,}5t+1{,}845\right)$$

Тепер порівняємо отриманий розв'язок з аналітичним та чисельним розв'язками отримані за допомогою MATLAB. Наведемо приклад програми, що розв'язує дану задачу "Poriv_Ric".

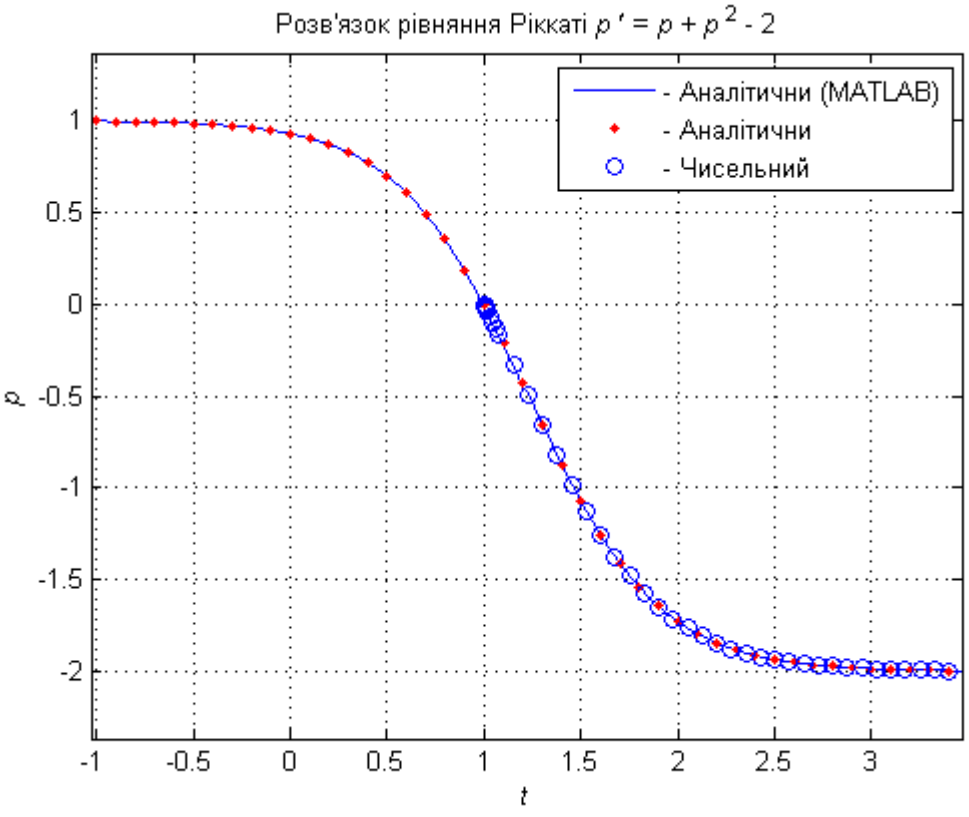
```
function Poriv_Ric
% Введення функції,що була отримана аналітично
t = -1:0.1:5; % диапазон зміни аргументу
p = -0.5+1.5*tanh(-1.5*t+1.845);
% Отримання аналітичного (символьного) розв'язку за допомогою MATLAB
y = dsolve('Dp = p+p^2-2','p(1) = 0','t');
disp('y = ')
pretty(y)
% Отримання чисельного розв'язку
% Виклик солвера
[T P] = ode45('Riccati',[1 4],0);
% Виведення графіку розв'язку рівняння Ріккаті(аналітичного (символьного))
```

```
ezplot(y)
hold on
% Виведення графіку розв'язку рівняння Ріккаті(аналітичного)
plot(t,p,'.','Color','red')
% Виведення графіку розв'язку рівняння Ріккаті(чисельного)
plot(T,P,'o')
title('Розв''язок рівняння Ріккаті {\itp} \prime = {\itp} + {\itp} ^2 - 2')
xlabel('\itt'), ylabel('{\itp}'), grid on
legend('- Аналітичні (MATLAB)','- Аналітичні','- Чисельний')
hold off
```

Результат виконання програми

y =

$$\frac{\frac{\exp(3\ t)}{\exp(3)} - 1}{-1 - 1/2\ \frac{\exp(3\ t)}{\exp(3)}}$$

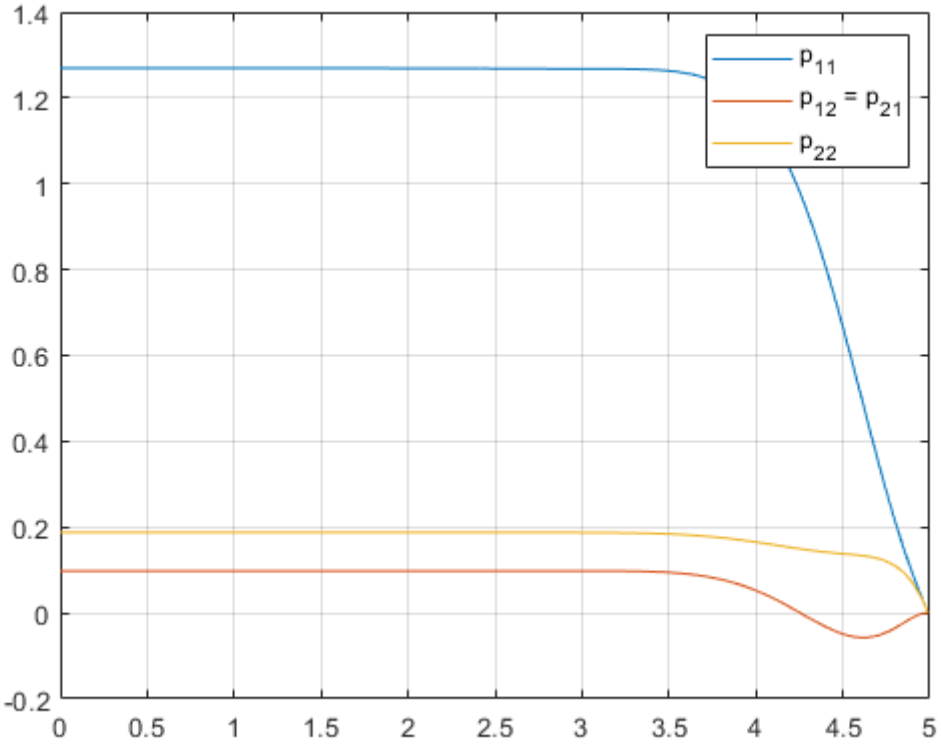


Наведемо приклад програми, що розв’язує дану задачу для системи другого порядку “Ric”.

```
function Ric
    disp('Матриця A:')
    A =[0 1; -5 -3] % Введення матриці A
    disp('Матриця B:')
    B = [0; 1] % Введення матриці B
    % Параметри критерія якості:
    disp('Матриця Q:')
    Q = eye(2) % Формування одиничної матриці
    disp('Матриця R:')
    R = 1/2 % Введення матриці R
    disp('Матриця S:')
    S = zeros(2)
    disp('Час спостереження tf:')
    tf = 5 % Завдання часу спостереження
    disp('Довжина кроку dt:')
    dt = 0.01 % Завдання довжини кроку
    disp('Кількість кроків n:')
    n = tf/dt
    % Визначення розмірності задачі
    s = size(A);
    % Формування P
    P = cell(1,n);

    P{n} = S;

    for i = n:-1:2
        P{i - 1} = P{i} - dt*(-P{i}*A - A'*P{i} + P{i}*B*1/R*B'*P{i} - Q);
    end

    p11 = zeros(1, n);
    p12 = zeros(1, n);
    p22 = zeros(1, n);

    for i = 1:1:n
        p11(i) = P{i}(1,1);
        p12(i) = P{i}(1,2);
        p22(i) = P{i}(2,2);
    end

    t = 0:dt:tf - dt;
    plot(t, p11, t, p12, t, p22), grid on
    legend('p_1_1', 'p_1_2 = p_2_1', 'p_2_2');
end
```

```
Матриця A:

A =

     0     1
    -5    -3
```

```
Матриця B:

B =

     0
     1

Матриця Q:

Q =

     1     0
     0     1

Матриця R:

R =

    0.5000

Матриця S:

S =

     0     0
     0     0

Час спостереження tf:

tf =

     5

Довжина кроку dt:

dt =

    0.0100

Кількість кроків n:

n =
```



Завдання

1. Отримати рівняння Ріккаті та їх розв’язок за допомогою MATLAB для синтезу системи оптимального керування, інтегрального квадратичного критерію якості у вигляді:

$$I=\frac{1}{2}X^{T}(t_f)SX(t_f)+\frac{1}{2}\int_0^{t_f}(X^{T}QX+U^{T}RU)dt$$

і об’єкту

$$X'=AX+BU$$

Варіанти:

1. $A=\begin{bmatrix}0 & 1\\ -40 & -12\end{bmatrix};\quad B=\begin{bmatrix}0\\ 10\end{bmatrix},$
2. $A=\begin{bmatrix}0 & 1\\ -3 & -5\end{bmatrix};\quad B=\begin{bmatrix}0\\ 1\end{bmatrix},$

3. $A=\begin{bmatrix} 0 & 1 \\ -3 & -7 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$,

4. $A=\begin{bmatrix} 0 & 1 \\ -5 & -11 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$,

5. $A=\begin{bmatrix} 0 & 1 \\ -7 & -14 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$,

6. $A=\begin{bmatrix} 0 & 1 \\ -9 & -14 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$,

7. $A=\begin{bmatrix} 0 & 1 \\ -10 & -14 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$,

8. $A=\begin{bmatrix} 0 & 1 \\ -12 & -14 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$,

9. $A=\begin{bmatrix} 0 & 1 \\ -14 & -12 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$,

10. $A=\begin{bmatrix} 0 & 1 \\ -14 & -12 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 5 \end{bmatrix}$,

Комп’ютерний практикум № 7

7. Генетичний алгоритм і прямий пошук

Genetic Algorithm and Direct Search Toolbox містить набір функцій, призначений для розширення Optimization Toolbox і середовища чисельних методів MATLAB і призначений для вирішення спеціального класу задач з алгоритмами прямого пошуку та алгоритмами структурного пошуку. В алгоритмах структурного пошуку проводиться розрахунок якоїсь послідовності точок, направлених на рух у напряму все ближче і ближче до шуканої оптимальної точки.

Генетичний алгоритм

Генетичний алгоритм є методом для вирішення задач оптимізації, який заснований на природному відборі, тобто є якоюсь аналогією біологічного процесу еволюції. Генетичний алгоритм неодноразово модифікує сімейство індивідуальних розв'язків. На кожному кроці генетичного алгоритму

проводиться відбір імовірності якихось індивідуальностей з поточного батьківського покоління і далі проводиться подальше дочірнє покоління. Через послідовний відбір поколінь проходить "еволюція" просування до оптимального розв'язку. Генетичний алгоритм можна застосовувати для різноманітних задач оптимізації, які не завжди вдало підходять для вирішення за допомогою стандартних оптимізаційних алгоритмів, і в першу чергу даний метод використовується при рішенні задач, коли цільова функція є перервною, не диференціюється, стохастичною або дуже нелінійною.

Відкриття генетичного алгоритму здійснюється з командного рядка MATLAB

>> gatool

На екрані з являється нове вікно (графічний інтерфейс) рис. 7.1.

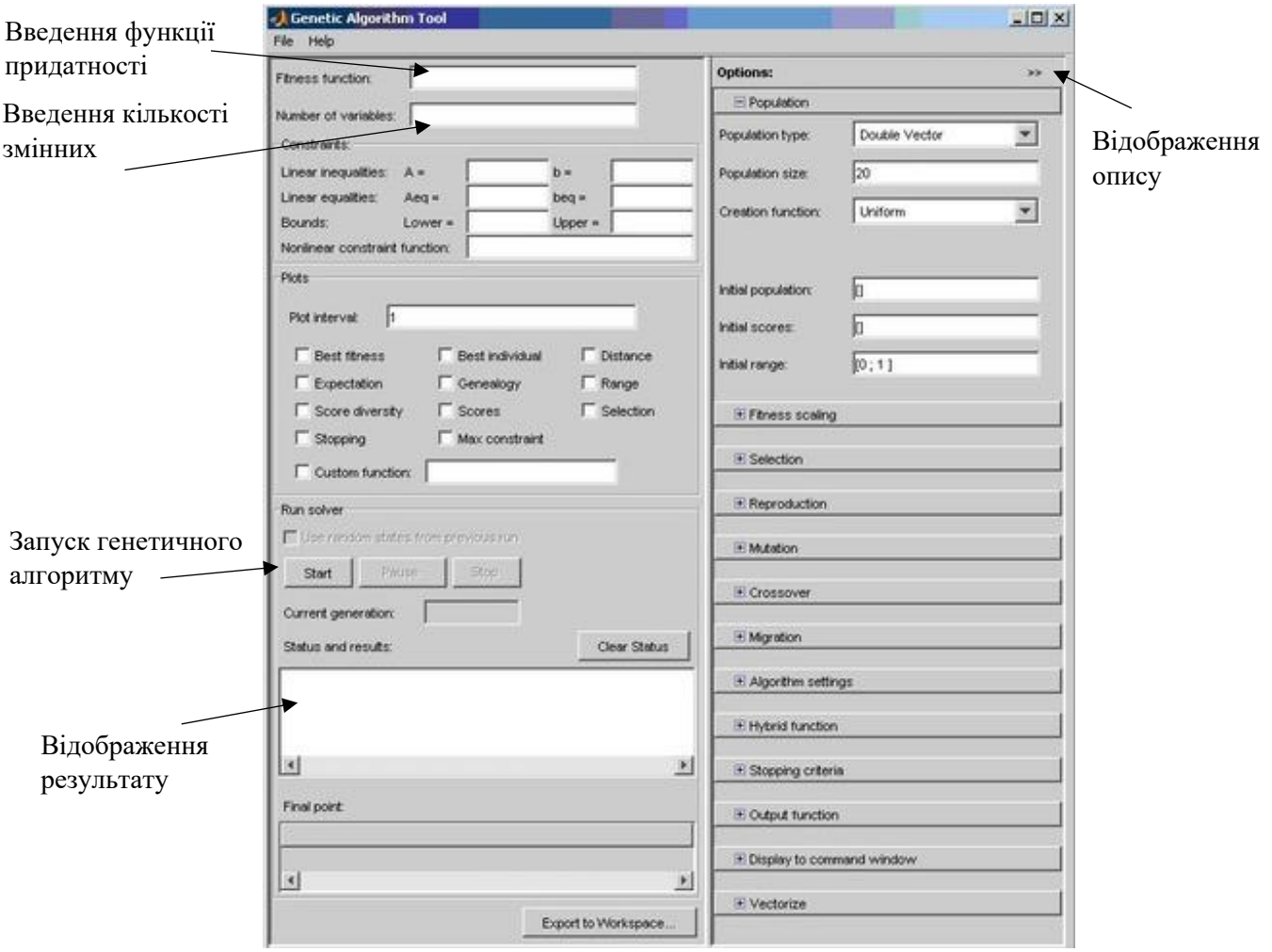


Рис. 7.1. Графічний інтерфейс генетичного алгоритму

Для більш детальної інформації можна скористатися "відображенням опису".

Запуск генетичного алгоритму з командного рядка відбувається наступним чином

```
>> [x fval] = ga(@fitnessfun, nvars)
```

де

вхідні аргументи:

@fitnessfun - функція придатності, що записана у вигляді M - файлу;

reason - причина зупинки алгоритму.

вихідні аргументи:

x - остаточний розв'язок;

fval - значення функції придатності в кінцевій точці.

Для більш детальної інформації див. додаток або довідкову систему MATLAB.

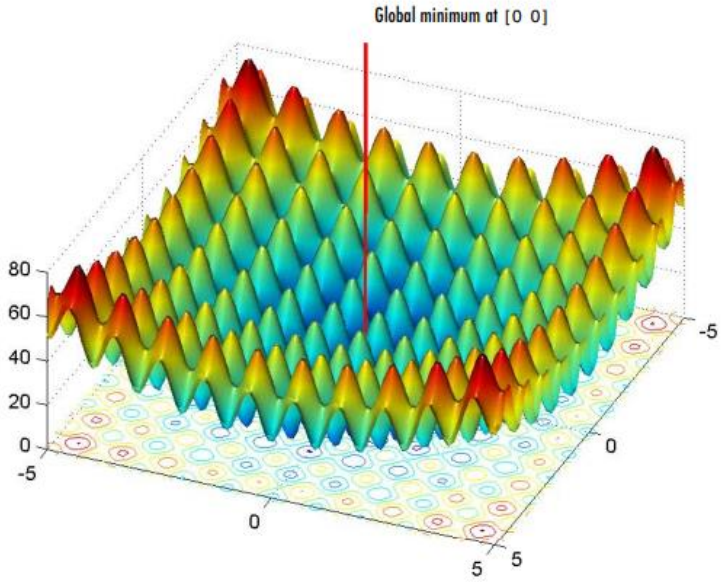
Приклад функції Растрігіна, як функції, яка достатньо часто використовується для тестування генетичних алгоритмів.

У випадку двох незалежних змінних функція Растрігіна вона записується як

$$Ras(X)=20+x_1^2+x_2^2-10\ (\cos 2\pi x_1 + \cos 2\pi x_2.). \qquad (7.1)$$

В пакеті міститься M-файл rastriginsfcn.m, який може використовуватися для розрахунку функції Растрігіна. Далі на малюнку приводиться поверхня та лінії рівня функції Растрігіна, що отримано за допомогою команди

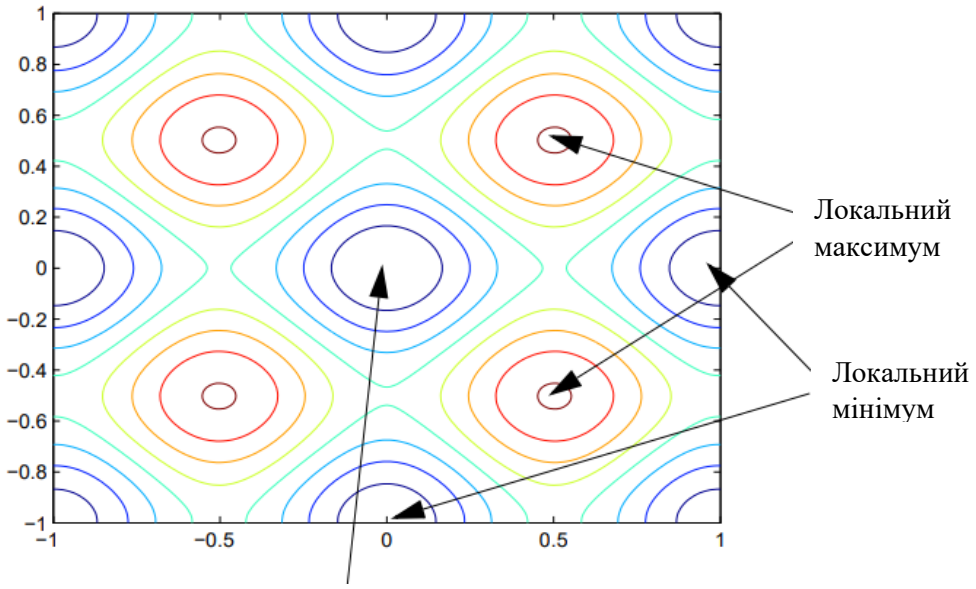
```
>> plotobjective(@rastriginsfcn, [-5 5;5-5])
```



Як випливає з поверхні, функція Растрігина містить безліч локальних мінімумів – "долин". Проте функція має тільки один глобальний мінімум, який знаходиться в точці [0 0], цей мінімум відзначений за допомогою вертикальної червоної прямої і де значення функції дорівнює 0. В будь-якому іншому відмінному від [0 0] локальному мінімуму значення функції Растрігина більше нуля. Чим далі локальний мінімум розташований від глобального мінімуму, тим більше значення функції в даній точці.

Функція Растрігіна часто використовується для тестування генетичного алгоритму, оскільки наявність великої кількості локальних мінімумів створює неподоланні труднощі для використовування стандартних методів пошуку глобального мінімуму на основі використовування градієнтних методів.

Приведений нижче графік рельєфу функції Растрігіна представляє розташування альтернативних максимумів і мінімумів



Пошук мінімуму функції Растрігіна

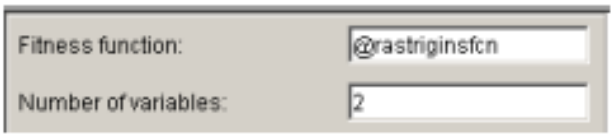
Для знаходження мінімуму слід виконати наступні кроки:

1. Ввести команду gatool в командному рядку, що відкриє Інструментарії генетичного алгоритму.

2. В Інструментарії генетичного алгоритму виконати наступні дії:

- В полі **Fitness function**, ввести @rastriginsfcn.
- В полі **Number variables**, ввести 2, число незалежних змінних для функції Растрігіна.

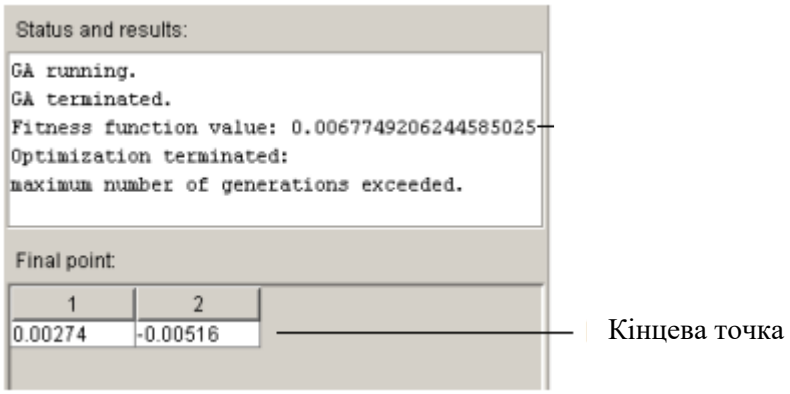
Поля **Fitness function і Number variables** повинні мати наступний вигляд, ик це показано на малюнку нижче



Натиснути мишкою на кнопку **Start** на панелі **Run solver**, як це показано на наступному малюнку.

У міру того, як відбувається виконання даного алгоритму, в полі **Current poneration** відбувається відображення числа поточних генерацій. Можна час від часу припиняти алгоритм, натискуючи на кнопку **Pause**. Після того, як операція виконається, те ім'я кнопки заміниться на **Resume**. Для продовження роботи агоритму з точки останову слід натиснути кнопку **Resume**.

3. Після закінчення алгоритму на панелі **Status and results** з'являться наступні параметри, як це показано нижче



На панелі **Status and results** відображається наступна інформація:

- Остаточне значення функції придатності в точці останову алгоритму;
- Function value: 0,0067749206244585025

Відзначимо, що значення, що відображається, знаходиться дуже близько до дійсного значення функції Растрігіна, рівного 0.

Способи зупинки алгоритму:

- **Exit**: Зупинити алгоритм;
- Перевищення максимального числа поколінь.

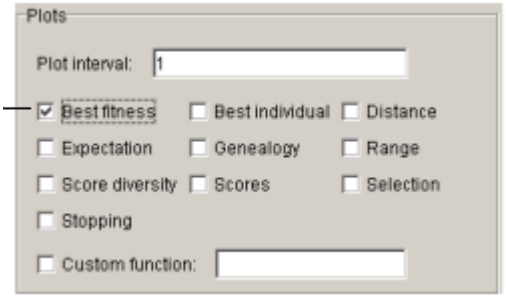
В даному прикладі алгоритм зупиняється після 100 поколінь, що приймається за умовчанням значення опції Generations, де задається максимальне число поколінь для розрахунку по заданому алгоритму.

В даному прикладі остаточна точка останову має значення [0,00274-0,00516].

Примітка: Оскільки при виконанні пошуку в генетичному алгоритмі використовуються стохастичні величини, то даний алгоритм може приводити до дещо відмінних результатів при різних запусках даної програми.

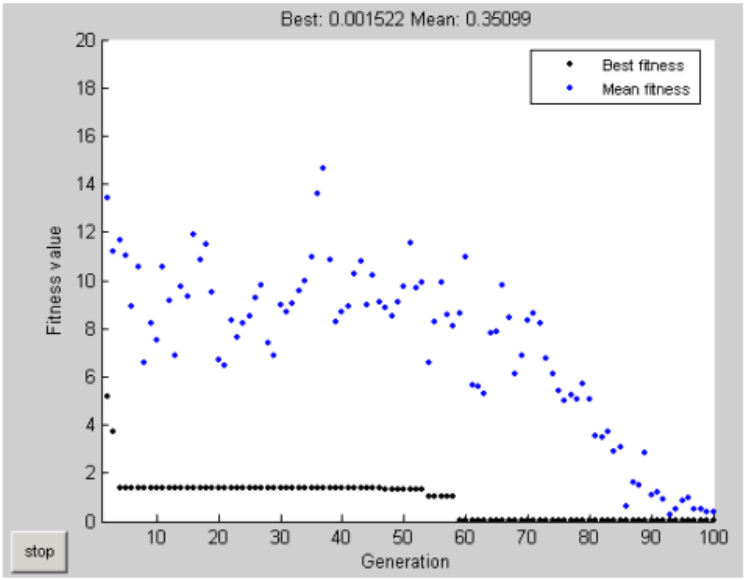
Відображення графіків

За допомогою панелі **Plots** є можливість відображення великої кількості різноманітної інформації про процес виконання генетичного алгоритму. Ця інформація може бути використана при зміні структури опцій з метою поліпшення ефективності роботи алгоритму. Наприклад, що б отримати графік кайкращого і середнього значення функції придатності кожної ітерації слід вибрати блок **Best fitness value**, як це показано далі на малюнку.



Якщо натиснути мишкою на кнопку **Start**, то в Інструментарії генетичного алгоритму відображається графік функції найкращого і середнього значень придатності, як це показано на малюнку нижче.

Точки в нижній частині графіка відносяться до підходящих значень, тоді як точки у верхній частині графіка відносяться до середніх значення функції придатності на кожній ітерації. У верхній частині графіка також приводяться чисельні значення найкращої придатності і середні значення для кожної ітерації.



Як правило, значення найкращої придатності покращуються найбільш помітно на ранніх поколіннях, коли вибрані об'єкти найбільш далекі від оптимального значення. Для останніх поколінь, у міру наближення даного

сімейства до оптимальної точки, покращення функції придатності відбувається більш поволі.

Деяка термінологія для генетичного алгоритму

Функції придатності. Функція придатності це підлягаюча оптимізації функція. У разі стандартних оптимізаційних алгоритмів вона відома як цільова функція. Задачею даного пакету є пошук мінімуму функції придатності.

Функцію придатності можна записати в М файл і передати у вигляді аргументу в основний генетичний алгоритм.

Індивідуум. Під індивідуумом в даному випадку розуміється якась точка, в якій можливий розрахунок функції придатності. Значення функції придатності об'єкту, що індивідуалізувався, якраз і є її кількісний показник. Наприклад, якщо функція придатності має наступний вигляд

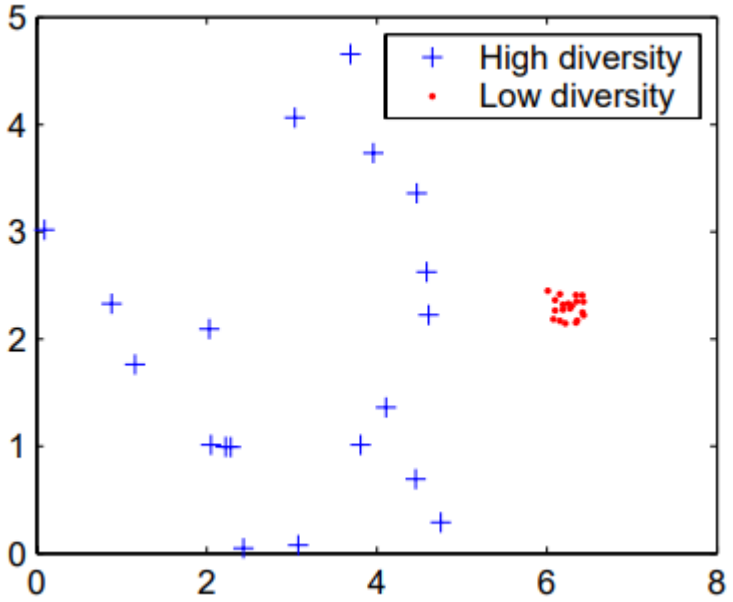
$$f(x_1, x_2, x_3)\ (2x_1 + 1)^2 + (3x_2 + 4)^2 + (x_3 - 2)^2. \qquad (7.2)$$

То вектор (2, 3, 1), чия розмірність рівна числу змінних даної задачі, і є індивідуум. Кількісний показник індивідуума як об'єкту (2, 3, 1) буде $f(2, -3, 1)=51$. Об'єкт індивідуума іноді називається - геном, а компоненти вектора називаються генами.

Сімейства і покоління. Сімейство є масив об'єктів, що індивідуалізувалися. Наприклад, якщо розмір сімейства рівний 100 і число змінних у функції при придатності рівно 3, то дане сімейство є матрицею розміром 100 х 3. Один і той же об'єкт індивідуума може з'являтися в даному сімействі більш, одного разу. Наприклад, об'єкт індивідуума (2, 3, 1) може виявлятися більш ніж в одному рядку масиву.

Для того, що б отримати нове покоління в генетичному алгоритмі на кожній ітерації проводиться якась серія розрахунків на основі поточного сімейства. Кожне успішне сімейство називається новим поколінням.

Розкид. Розкидом називається середня відстань між об'єктами індивідуумів для певного сімейства. Сімейство володіє високим розкидом, якщо середня відстань між індивідуумами є істотно великою, в протилежному випадку буде низький розкид. Далі на малюнку зображено сімейство, що ліворуч володіє високим розкидом, тоді як сімейство праворуч має низький розкид.



Поняття розкиду є істотним для генетичного алгоритму, оскільки через нього проводиться оцінка розміру даного простору.

Значення придатності і значення якнайкращої придатності. Значення придатності об'єкту індивідуума є значення функції придатності для даного об'єкту, що індивідуалізувався. Оскільки даний пакет призначений для пошуку мінімуму функції придатності, те значення якнайкращої придатності для якогось сімейства є якнайменше значення функції придатності для будь-якого індивідуума даного сімейства.

Батьківський і дочірній. Для того, щоб отримати подальше сімейство, в генетичному алгоритмі відбір проводиться певних об'єктів, що індивідуалізувалися, в поточному сімействі, так званому батьківському, який далі використовуються для утворення об'єктів індивідуумів для наступного сімейства, званого дочірнього. Як правило, вибираються ті батьки, які дають більш краще значення функції придатності.

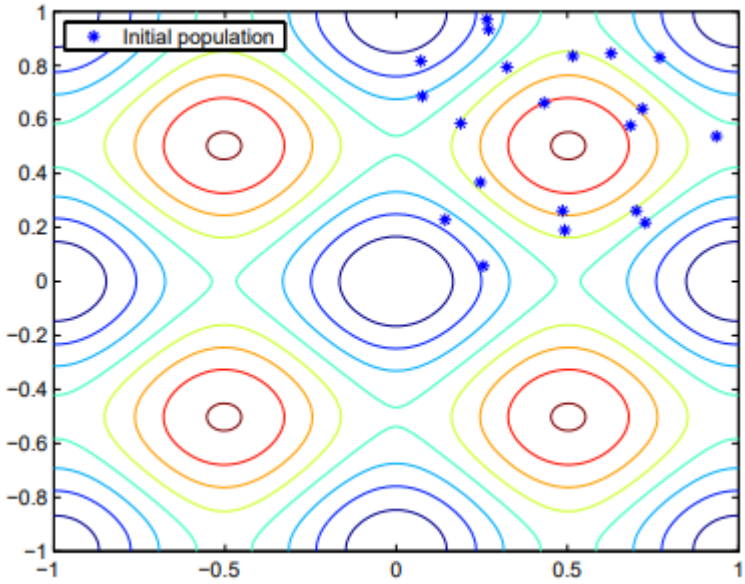
Робота генетичного алгоритму

Генетичний алгоритм працює згідно наступній схемі:

1. Перш за все, в даному алгоритмі для організації початку рахунку створюється довільне початкове сімейство.
2. Далі алгоритм проводить деяку послідовність нових сімейств або поколінь. На кожному окремому кроці алгоритм використовує певних індивідуумів з поточного покоління, для того, що б створити подальше покоління. При формуванні нового покоління в алгоритмі проводяться наступні дії:
   - Наголошується кожний член поточного сімейства за допомогою обчислення відповідного значення придатності.
   - Проводиться маштабування отриманого ряду значень функції придатності, що дозволяє побудувати діапазон значень більш зручний для подальшого використовування.
   - Вибираються батьківські значення на основі значень їх придатності.
   - Частина індивідуумів з батьківського покоління має більш краще значення функції придатності і які надалі вибираються як вибрані (елітні) значення. Ці вибрані (елітні) значення передаються далі вже в подальше покоління.
   - Дочірні значення утворюються або шляхом деяких випадкових змін окремого одного батька мутація, або шляхом комбінації векторних компонентів деякої пари батьків кроссовер.
   - Заміна поточного сімейства на дочірнє з метою формування подальшого покоління.
3. Останов алгоритму проводиться тоді коли виконується будь-який критерій останову.

Вихідне сімейство

На початку алгоритму, перш за все, створюється випадкове початкове сімейство, наприклад, як це показано на наступному малюнку.



В даному прикладі, початкове сімейство містить 20 індивідуумів, що приймається за умовчанням значенням для розміру сімейства в опції **Population**. Слід зазначити, що всі індивідууми лежать у верхньому правому квадранті даного малюнка, тобто їх координати лежать в діапазоні від 0 до 1, оскільки приймаються за умовчанням значення **Initial range** для опції **Population** рівними [0; 1].

Якщо приблизно відоме місцеположення точки мінімуму, то слід вибрати функцію **Initial range,** так що б необхідна точка знаходилася в середині вибраного діапазону. Наприклад, якщо відомо, що для функції Растрігіна точка мінімуму знаходиться в діапазоні [00], то слід вибрати функцію **Initial range** і встановить діапазон [-1; 1]. Проте, як випливає з даного прикладу, генетичний алгоритм може вийти на точку мінімуму навіть з меншими витратами, ніж у разі попереднього вибору за допомогою функції **Initial range.**

Створення наступного покоління

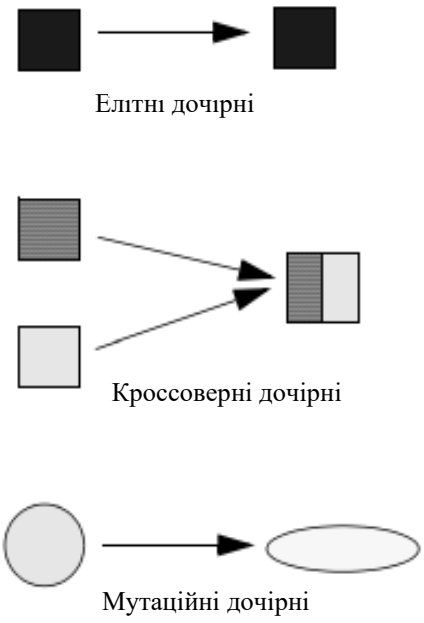
На кожному розрахунковому кроці в генетичному алгоритмі поточне сімейство використовується для створення дочірнього сімейства або подальшого покоління. Даний алгоритм використовує групу індивідуумів з поточного сімейства, званого батьківським, яке передає свої гени компоненти відповідних векторів своїм дочірнім сімействам. Як правило, в даному алгоритмі вибираються саме ті індивідууми, які мають більш краще значення придатності, ніж батьківські. Дану функцію можна вибрати таким чином, що б згідно

алгоритму сформувати значення для батьків в полі **Selection function** в опції **Selection**.

В генетичному алгоритмі для формування подальшого покоління використовуються три типи дочірніх поколінь:

1. Вибрані дочірні це ті індивідууми в поточному сімействі, які мають якнайкращі значення придатності. Ці індивідууми зберігаються для наступного сімейства. автоматично
2. Кроссоверні дочірні утворюються шляхом комбінації векторів з батьківської пари.
3. Мутаційні дочірні утворюються за рахунок введення випадкових варіацій або мутацій з окремого батьківського значення.

На наступній діаграмі схемно приводиться ілюстрація цих трьох типів дочірніх значень.



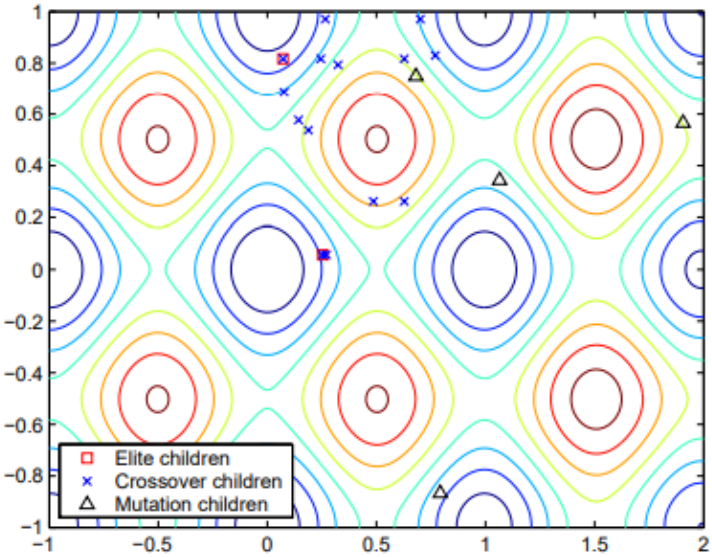
Кроссоверні дочірні значення

Даний алгоритм формує Кроссоверні дочірні значення шляхом комбінування з пари батьківських та з поточного сімейства. Для кожної координати дочірнього вектора, що приймається за умовчанням функція кроссовера випадковим чином вибирає елементи, або гени, для тієї ж самої координати для одного з двох батьківських значень і привласнює їх дочірньому значенню.

Мутаційні дочірні значення

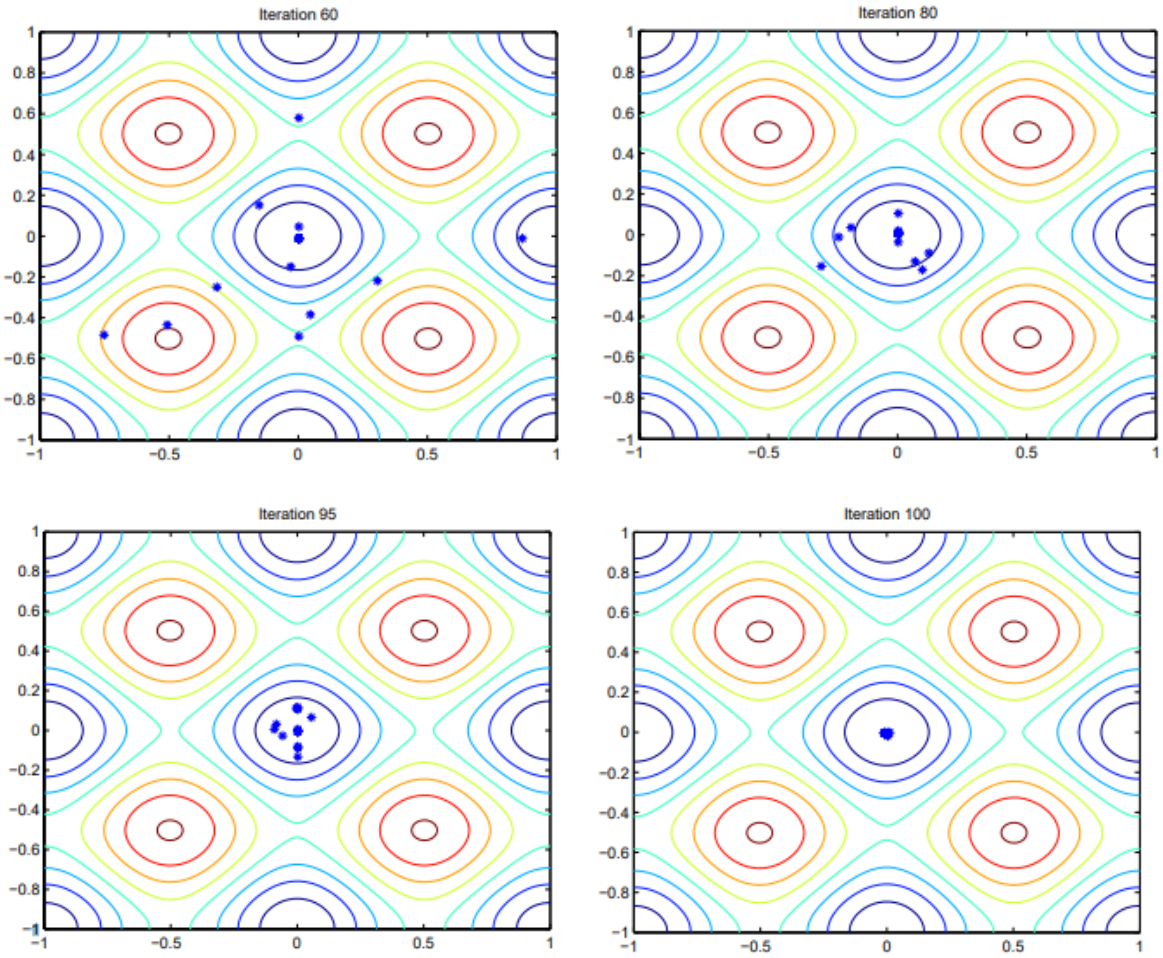
Даний алгоритм формує мутаційні дочірні значення шляхом випадкової вибірки генів з батьківських індивідуумів. За умовчанням приймається, що даний лгоритм додає до батьківського значення вектор імовірності з Гауссового розподілу.

Далі на малюнку приводиться вид дочірнього значення для початкового сімейства, тобто, сімейства другого покоління і позначаються або елітні, або кроссоверні, або мутаційні дочірні значення.



Графік останнього покоління

У міру збільшення числа ітерацій, індивідууми для даного сімейства оходяться все ближче і ближче до точки мінімуму [0 0]. Далі на малюнках приводяться сімейства для ітерацій, відповідно, 60, 80, 95 і 100.



Умови останову алгоритму

Для визначення умов останову в генетичному алгоритмі використовуються наступні п'ять умов:

Generations - алгоритм зупиняється тоді, коли число поколінь досягає якогось заданого значення **Generations.**

Time limit - алгоритм зупиняється по закінченню якогось заданого часу в секундах **Time limit.**

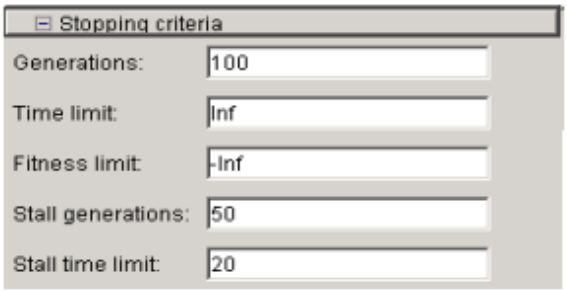
Fitness limit - алгоритм зупиняється тоді, коли значення функції придатності для якнайкращої точки, для поточного сімейства буде менше або рівно **Fitness limit.**

Stall generations - алгоритм зупиняється у випадку, якщо немає поліпшення для цільової функції в послідовності наступних один за одним поколінь завдовжки **Stall generations.**

Stall time limit - алгоритм зупиняється у випадку, якщо немає поліпшення для цільової функції протягом інтервалу часу в секундах рівного **Stall time limit.**

Виконання алгоритму зупиняється тоді, коли зустрічається одна з цих п'яти умов. Значення цих критеріїв можна вибрати в опції **Stopping criteria** в інструментарії Генетичного алгоритму. Що приймаються за умовчанням значення приводяться нижче.

При виконанні генетичного алгоритму на панелі Status проводиться відображення ініціюючої останов критерію.



Опції **Stall time limit** і **Time limit** застерігають алгоритм від дуже довгого часу виконання. Якщо алгоритм зупиняється по виконанню однієї з цих умов, то, природно, результати можна буде покращити при збільшенні значень в опціях **Stall time limit** або **Time limit.**

Метод прямого пошуку або метод безпосереднього пошуку

Метод прямого пошуку використовується для вирішення задач оптимізації, коли відсутня яка-небудь інформація про градієнт цільової функції. На противагу більш традиційним методам оптимізації, в яких використовується інформація про градієнт і про неодноразові похідні при пошуку оптимальної точки, в алгоритмі прямого пошуку проводиться вибір якогось набору точок в околі поточної точки з метою пошуку такої точки, де значення цільової функції буде більш оптимальним, ніж в початковій точці. На кожному кроці даного алгоритму проводиться пошук якогось набору точок, так званої сітки в околі поточної точки - а саме точки, визначеної на попередньому кроці даного алгоритму. Даний алгоритм формує сітку шляхом додавання поточної точки до скалярного множника фіксованого набору векторів. Якщо в даному алгоритмі знаходиться якась точка для даної сітки, яка приводить до поліпшення цільової функція відносно поточної точки, то далі ця нова точка стає вже поточною точкою для подальшого кроку вибраного алгоритму. Метод прямого пошуку

можна використовувати для вирішення таких задач, коли цільова функція не диференціюється або навіть перервна.

Відкриття графічного інтерфейсу методу прямого пошуку здійснюється командного рядка MATLAB.

>> psearchtool

На екрані з'являється нове вікно (графічний інтерфейс) рис.7.2

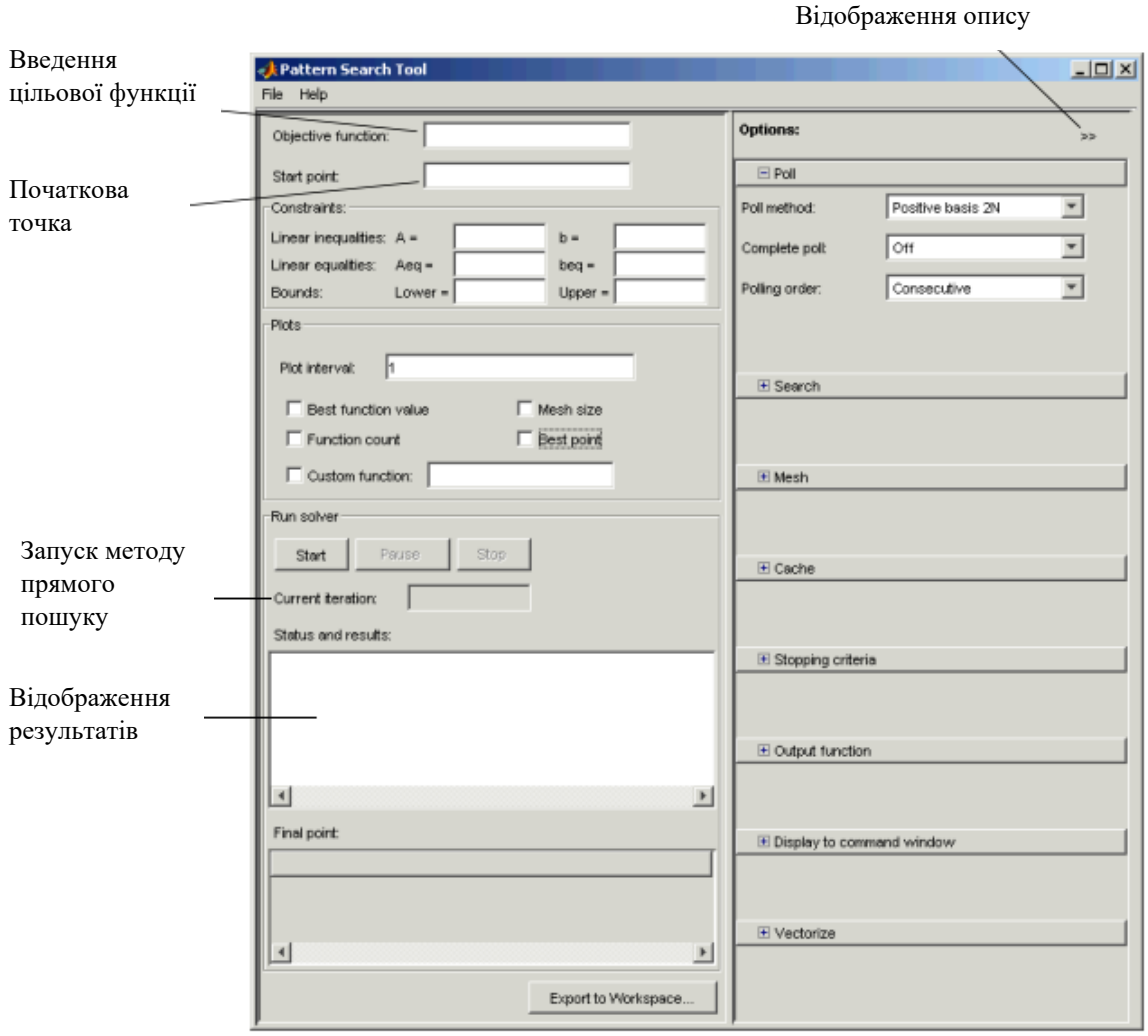


Рис. 7.2. Графічний інтерфейс методу прямого пошуку

Для більш детальної інформації можна скористатися "відображенням опису".

Для запуску виконання команди в методі безпосереднього пошуку для задач без обмежень за допомогою командного рядка слід виконати команду patternsearch з наступним синтаксисом:

```
>> [x fval] = patternsearch(@objfun, x0)
```

де

@objfun - цільова функція, що записана в M файл;

x0 - початкова точка методу прямого пошуку.

Результати обчислень одержуються так:

fval - кінцеве значення цільової функції.

x - точка, де досягнуте кінцеве значення.

Для більш детальної інформації див. Додаток або довідкову систему MATLAB.

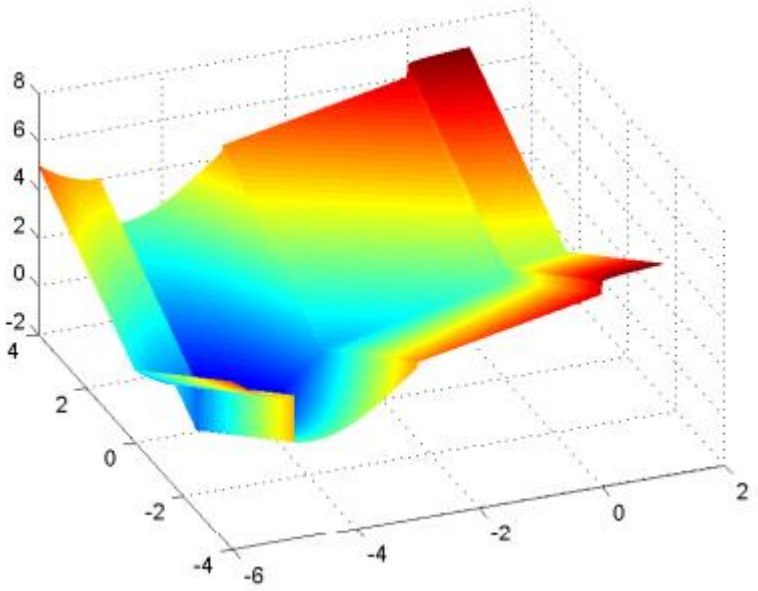
Приклад пошуку мінімуму функції

Як приклад використовується цільова функція **ps_example,** включена в пакет **Genetic Algorithms and Direct Search.** Саму функцію можна проглянути за допомогою команди:

```
>> type ps_example
function f = ps_example(x)
%PS_EXAMPLE objective function for pattern search.
% Copyright 2003-2004 The Mathworks, Inc.
% $Revision: 1.2.4.1 $ $Date: 20/08/2004 19:50:21 $
for i = 1:size(x, 1),
    if x(i,1) <-5
        f(i) (x(i,1)+5)^2 + abs(x(i,2));
    elseif x(i,1) <-3
        f(i)=-2*sin(x(i, 1)) + abs(x(i,2));
    elseif x(i,1) <0
        f(i)=0,5*x(1,1) + 2 + abs(x(1,2));
    elseif x(i,1) = 0
        f(i) = .3*sqrt(x(i,1)) + 5/2 +abs (x(i, 2));
    end
end
```

Далі на малюнку представлена поверхня та ліній рівня цієї функції, що отримані за допомогою команди

```
>> plotobjective(@ps_example, [-6 2:4-4])
```



Для відкриття методу прямого пошуку введемо команду

```
>> psearchtool
```

Відкриється інструментарій **Pattern Search**.

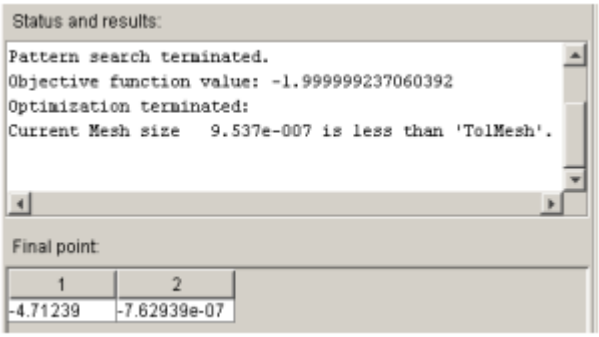
В полі **Objective function** інструментарію **Pattern Search Tool** ввести @ps_example.

В полі **Start point** набрати [2,1 1,7].

Поле панелі **Constraints** можна залишити без змін, оскільки обмеження в даній задачі, відсутні.

Для виконання програми прямого пошуку слід виконати команду Start.

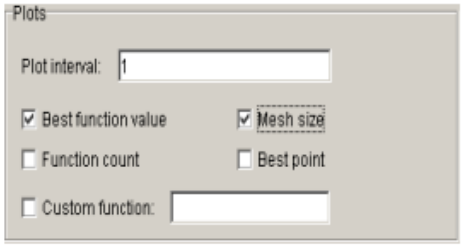
В панелі **Status and Results** будуть відображені результати виконання програми безпосереднього пошуку мінімуму для даної функції.



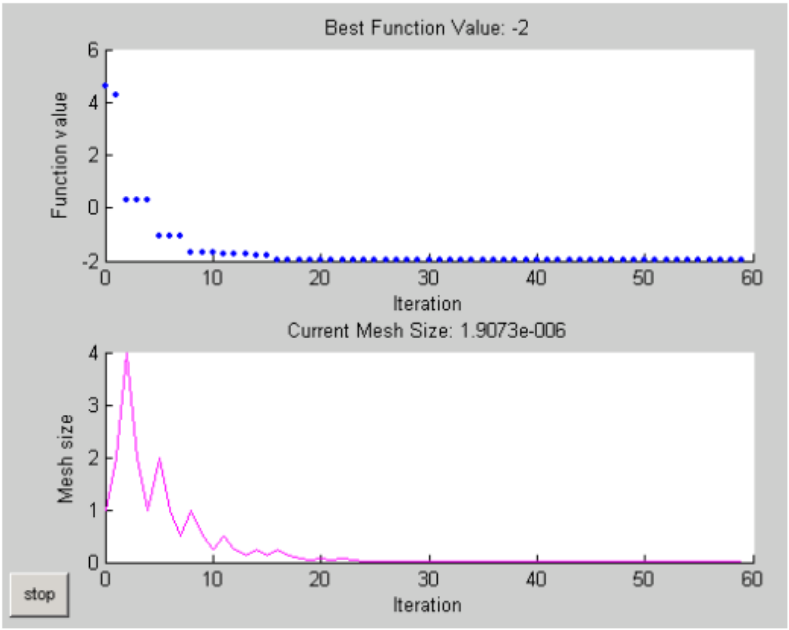
Мінімум функції приблизно рівний -2. На панелі **Final point** відображаються параметри знайденої точки мінімуму.

Графік значень цільової функції і розмірів осередку

Для того, що б проглянути характеристики результатів виконання програми безпосереднього пошуку можна побудувати графік найкращих значень функції і розмірів осередку на кожній ітерації. Для цього на панелі Plots вводяться наступні параметри:



Потім за допомогою кнопки Start запускається програма безпосереднього пошуку. Далі можна буде проглянути наступні графіки.



На верхньому графіку для кожної ітерації представлені значення цільової функції в якнайкращій точці. Як правило, значення цільової функції достатньо швидко поліпшуються на ранніх ітераціях і далі, у міру наближення до оптимального значення, відбувається їх вирівнювання.

На нижньому графіку представлений розмір осередку для кожної ітерації. Розмір осередку збільшується після кожної вдалої ітерації і зменшується після кожної невдалої ітерації.

Термінологія методу прямого пошуку

Шаблони. Шаблон це сукупність векторів, які використовуються в даному алгоритмі для визначення точок пошуку на кожній ітерації. Набір векторів є визначений числом незалежних змінних N в цільовій функції, та додатній набір базису. Зазвичай використовують два додатні набори базису в шаблоні пошукового алгоритму максимальний базис $2N$ векторів, а мінімальний з N+1 векторів.

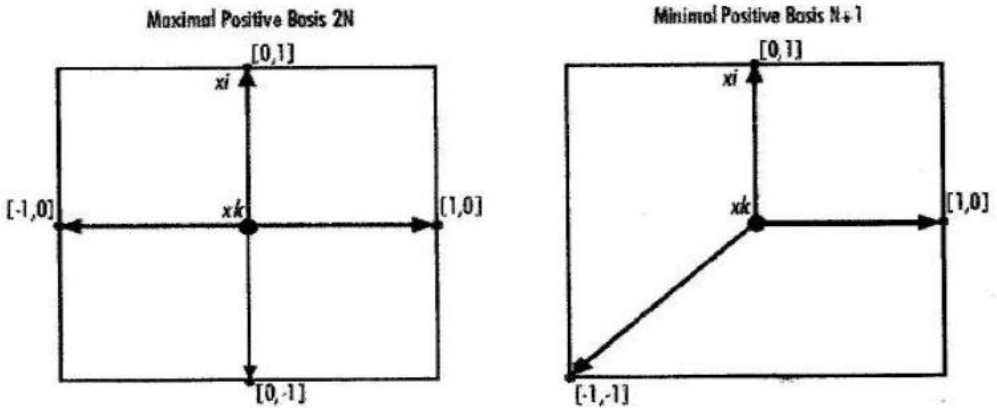
Наприклад, якщо є дві незалежні змінні в задачі оптимізації, то по замовчуванню додатній базис для $2N$ векторів складається з наступного шаблону:

- v_1= [1 0]
- v_2 = [0 1]
- v_3= [-1 0]
- v_4[0 -1]

Додатній базіс для N+1 векторів по замовчуванню складається з наступного шаблону:

- v_1 = [1 0]
- v_2= [0 1]
- v_3 = [-1 -1]

Далі ці вектори представлені на малюнку:



Осередки. На кожному кроці розрахунку в алгоритмі безпосереднього пошуку для точки, яка покращує значення цільової функції, визначається деякий набір точок, так званих осередків. Алгоритм формування осередку буде наступним:

1. Становлячи шаблон вектори множаться на деяку скалярну величину, так званий розмір осередку Д", гессеніан довжиною та напрямком 4, алгоритм рухається від точки хх до точки х. Нова точка може бути записана як попередня плюс крок

$$x_i=x_{i-1}+d_i.$$

Розмір осередку може бути записаний як

$$\Delta^m = \frac{\|x_i - x_{i-1}\|}{\|d_i\|}.$$

Якщо, вектор напрямку нормований, то розмір осередку визначається просто

$$\Delta^m = x_i - x_k.$$

2. Отримані в результаті множення вектори складаються з поточною точкою - точкою з якнайкращим значенням цільової функції, знайденим на попередньому кроці.

Наприклад, приймемо наступні значення величин:

1. Поточна точка [1,6 3,4].
2. Шаблон складається з векторів:

- v_1= [1 0]
- v_2 = [0 1]
- v_3= [-1 0]
- v_4[0 -1]

Поточний розмір осередку дорівнює 4.

Згідно алгоритму вектори шаблона множаться на 4 і потім складаються з поточною точкою, утворюючи, таким чином, наступний осередок:

- [1,6 3,4] + 4 [1 0] = [5,6 3,4];
- [1,6 3,4] + 4 [0 1] = [1,6 7,4];
- [1,6 3,4] + 4[-1 0] = [-2,4 3,4];

- [1,6 3,4] + 4 [0 -1] = [1,6-0,6].

Вектор шаблона, який приймається за точку осередку, називається його напрямом.

Опитування. На кожному кроці згідно прийнятому алгоритму проводиться опитування точок в поточному осередку за допомогою відповідного розрахунку значень цільової функції. У випадку, якщо приймається по умовчанню значення опції **Complete poll** встановлено як **Off**, то згідно алгоритму процесу дослідження точок осередку зупиняється, як тільки буде знайдена якась точка, в якій значення цільової функції буде менше ніж в поточній точці. Якщо останов дійсно має місце, то опит називається успішним і знайдена точка стає вже поточною точкою для наступної ітерації. Помітимо, що згідно алгоритму обчислення для точок осередку і їх цільових функцій проводяться тільки до того моменту, як зустрінеться зупиняюча дане опитування точка. Якщо алгоритм не може знайти точку з більш відповідною цільовою функцією, то таке опитування називається неуспішним і поточна точка залишається, відповідно, тієї ж самої і для наступної ітерації.

Якщо опція **Complete poll** встановлена як **On,** то в даному алгоритмі проводиться розрахунок значень цільової функції для всіх точок даного осередку. Далі в алгоритмі проводиться порівняння по всіх точках поточного осередку з найменшим значенням цільової функції. Якщо в якійсь точці осередку значення цільової функції буде менше, то опитування буде успішним.

Виконання методу безпосереднього пошуку

Успішні пошуки

Метод безпосереднього пошуку починається з деякої заздалегідь вибраної початкової точки хо. В нашому прикладі це буде точка хо= [2,1 1,7].

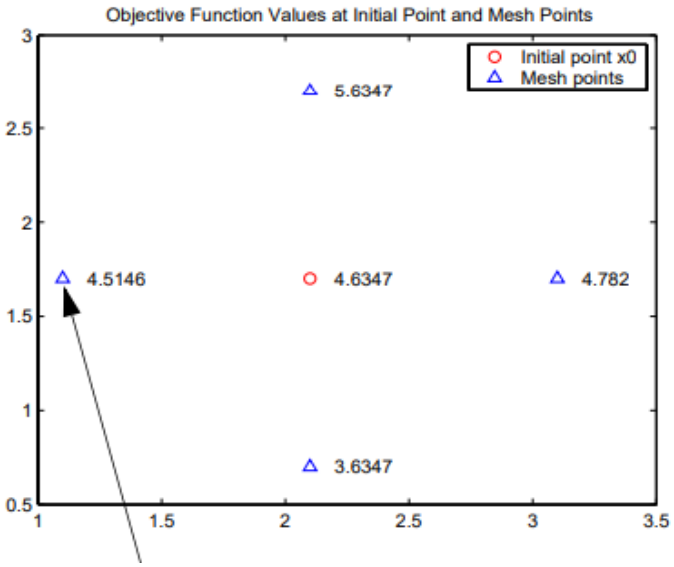
Ітерація 1

На першій ітерації розмір осередку рівний 1 і згідно алгоритму безпосереднього пошуку початкова точка x_0 = [2,1 1,7] складається з векторами шаблона, що дозволяє розрахувати наступний осередок точки:

- [1 0] + x_0[3,1 1,7];

- [0 1] + x_0 [2,1 2,7];
- [-1 0] + x_0 [1,1 1,7];
- [0 -1] + x_0[2,10,7].

Згідно прийнятому вище порядку в алгоритмі проводиться розрахунок цільової функції в нових точках осередку.



Значення цільової функції в початковій точці

Далі на основі розрахунку величин цільової функції в алгоритмі проводиться відбір серед точок осередку до тих пір, поки не буде знайдена така точка, в якій значення цільової функції буде менше ніж 4,6347 для початкової точки x_0. В даному прикладі першою такою точкою буде точка [1,1 1,7], в якій розрахункове значення цільової функції рівно 4,5146, таким чином, вже для першої ітерації опитування буде успішним. Далі алгоритм заносить цю точку в послідовність пошуку як:

- x_1 = [1,1 1,7].

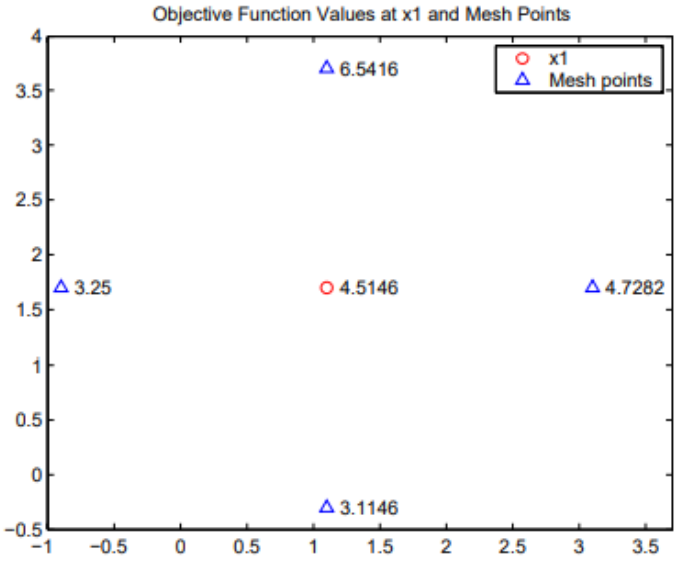
Примітка. За умовчанням приймається, що останов алгоритму безпосереднього пошуку в поточній ітерації відбувається тоді, коли зустрічається така точка осередку, в якому значення цільової функції буде менше ніж в початковій поточній точці. Отже, в даному випадку алгоритм може і не опитувати всі точки осередку. Що б проводити опитування всіх точок осередку необхідно встановити опцію Complete poll як On.

Ітерація 2

Після такого успішного опитування в алгоритмі розмір поточного осередку множиться на 2 (це що приймається за умовчанням коефіцієнт **Expansion factor** в панелі опції **Mesh**). Оскільки початковий розмір осередку прийнятий за 1, то розмір осередку для другої ітерації буде рівний 2. Осередок другої ітерації включає наступні точки:

- 2*[1 0]+x_1= [3,1 1,7];
- 2*[0 1]+x_1 = [1,1 3,7];
- 2*[-1 0]+x_1 = [-0,9 1,7];
- 2*[0 -1]+x_1 = [1,1-0,3].

Далі на малюнку представлена точка x_i і решта точок осередку спільно з відповідними значеннями функції ps_example.



Після цього алгоритм проводить подальше опитування точок осередку до тих пір, поки не буде знайдено значення, менше ніж 4,5146 для відповідної точки x_i. Перша така точка буде знайдена для координат [-0,9 1,7], в якій значення цільової функції рівно 3,25, і, таким чином, ітерація 2 знов вважатиметься успішною. Далі алгоритм заносить цю другу точку в послідовність пошуку як

- x_2 = [-0,9 1,7].

Оскільки опитування було успішним, то згідно алгоритму розмір поточної точки множиться на 2 і для третьої іт ерації розмір осередку вже рівний 4.

Невдалий пошук

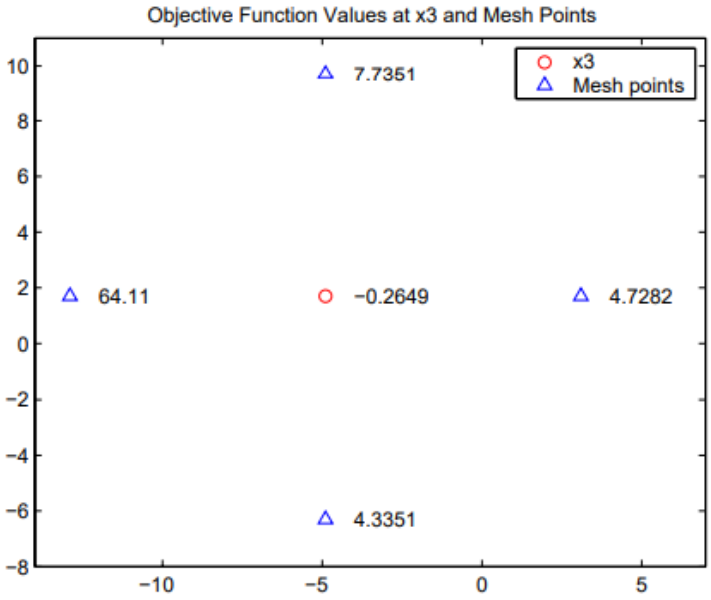
Для четвертої ітерації поточна точка буде рівна

- x_3 [-0,9 1,7].

Розмір осередку буде рівний 8, і такий осередок включатиме точки

- 8*[1 0]+x_3 = [3,1 1,7];
- 8*[-1 0]+ x_3 = [-4,9 1,7];
- 8*[0 1]+ x_3 = [-0,9 5,7];
- 8*[0 -1]+ x_3 = [-0,9-2,3].

Далі на малюнку представлені точки осередку і відповідні значення цільової функції.



Для даної ітерації вже не буде таких точок осередку, які мають значення цільової функції менше ніж в точці xз і, таким чином, опитування буде невдалим. В даному випадку вже відбувається зміна поточної точки для наступної ітерації. Тобто приймається

- $x_4 = x_3$.

Для наступної ітерації розмір поточного осередку множаться на 0,5 (що приймається за умовчанням коефіцієнт Contraction factor в панелі опції Mesh), і, таким чином, розмір осередку для наступної ітерації вже буде рівний 4. Далі опитування проводиться вже для осередку з меншим розміром.

Відображення результатів на кожній ітерації

Результати застосування методу безпосереднього пошуку можна прослідити і для кожної ітерації, в цьому випадку в опції **Display to command window параметр Level display** слід встановити як **Iterative**. Така можливість дозволяє оцінювати ступінь прогресу, що досягається в роботі методу безпосереднього пошуку і у разі потреби вносити відповідні зміни в параметри options.

За допомогою подібних установок в командному рядку проводиться відображення інформації про хід виконання програми безпосереднього пошуку. Після чотирьох ітерацій буде виведена наступна інформація:

Iter	f-count	MeshSize	f(x)	Method
0	1	1	4.635	Start iterations
1	4	2	4.515	Successful Poll
2	7	4	3.25	Successful Poll
3	10	8	-0.2649	Successful Poll
4	14	4	-0.2649	Refine Mesh

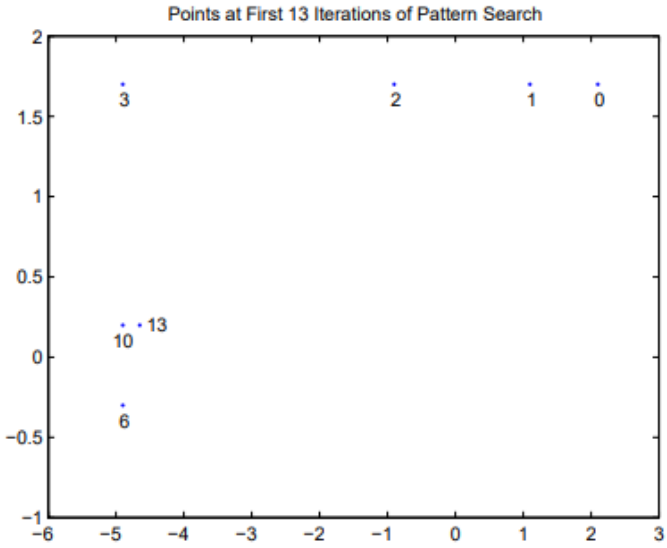
Записи **Successful Poll** під заголовком **Method** указують, що дані ітерації були успішними. Наприклад, можна бачити, що опитування на ітерації є успішним. Відповідно, значення цільової функції в розрахунковій точці на ітерації 2 під заголовком f(x) менше ніж для ітерації 1.

Для ітерації 4 запис **Refine Mesh** в стовпці **Method** повідомляється про те, що опитування було невдалим. Відповідно значення функції для ітерації 4 залишається незмінним після ітерації 3.

Відзначимо, що в методі безпосереднього пошуку відбувається подвоєння розмірів осередку після кожного успішного опитування і зменшення в два рази після кожного невдалого опитування,

Додаткові ітерації

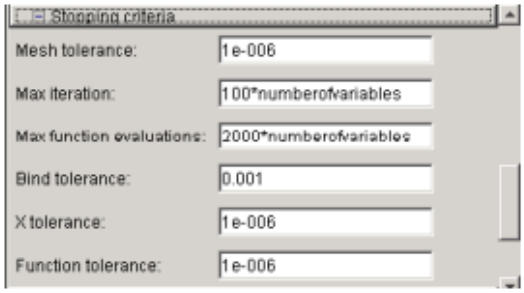
До точки останову в методі безпосереднього пошуку виконуються 88 ітерацій. На малюнку нижче приведені результати розрахунку в певних точках після 13 ітерацій методу безпосереднього пошуку.



Цифра нижче за точку указує на першу ітерацію, в якій зустрічається дана точка. На малюнку приведені тільки номери ітерацій відповідні успішному значенню параметра опитування, оскільки найкраща точка вже не змінюється при неуспішному опитуванні. Наприклад, якнайкраща точка для 4 і 5 ітерацій така ж, як і для 3 ітерації.

Умови останову для методу безпосереднього пошуку

Список критеріїв представлений в розділі **Stopping criteria** інструментарію методу безпосереднього пошуку. Елементи інтерфейсу представлені на малюнку нижче



Процес виконання алгоритму зупиняється, коли виконується одна з наступних умов:

- Розмір осередку буде менше ніж величина в полі **Mesh tolerance**.
- Число виконаних в алгоритмі ітерацій досягне величини в панелі **Max iteration**.
- Загальне число розрахунку виконаних в алгоритмі значень цільової функції досягне величини в панелі **Max function evaluations**.

- Відстань між точкою, знайденою в успішному опиті, і крапкою, знайденому в невдалому опиті буде менше ніж **X tolerance**.
- Зміни в цільовій функції для одного успішного опитування і подальшого успішного опитування будуть менше величини **Function tolerance**.

Значення опції **Bind tolerance**, яка використовується для ідентифікації обмежень в задачах з обмеженнями, не використовується як критерій останову. Алгоритм використовує відношення між розміром осередку Δ^m та параметром опитуванням Δ^p, щоб визначити критерій останову. Для додатного базису N+1 параметр опитування $N\sqrt{\Delta^m}$і для додатного базису 2N параметр опитування $\sqrt{\Delta^m}$.Зупинка критерію відбувається за умови $\Delta^m \leq$ **Mesh tolerance.**

Застосування генетичного алгоритму та методу безпосереднього пошуку для оптимізації за наявності нелінійних обмежень

Для вирішення задач за наявності нелінійних обмежень в генетичному алгоритмі та методі безпосереднього пошуку використовується метод безпосереднього пошуку з доповненням Лагранжа (ALPS). Задача оптимізації з використанням генетичному алгоритму та методу безпосереднього пошуку з доповненням Лагранжа (ALPS) формулюється таким чином

$$\min f(X), \tag{7.4}$$

за обмежень

$C_i(X) \leq 0, \quad i=1...m;$

$C_i(X) = 0, \quad i=m+1...m;$

$AX \leq B;$

$AeqX = Beq;$

$LB \leq X \leq UB,$

де $C(X)$ є системою обмежень у вигляді нелінійних рівності і нерівностей; m-число нелінійних обмежень у вигляді нерівностей; т, загальне число нелінійних обмежень.

В генетичному алгоритмі та методі безпосереднього пошуку з доповненням Лагранжа (ALPS) робиться спроба вирішити задачу оптимізації за наявності нелінійних обмежень, лінійних обмежень і граничних значень. Ставиться

гессеніан і знайти комбінацію цільової функції і функції нелінійних обмежень з використанням методу Лагранжа і штрафних параметрів. Далі проводиться наближена мінімізація отриманої послідовності із задач оптимізації при використовуванні алгоритму методу безпосередньої мінімізації з урахуванням можливих лінійних обмежень і граничних значень.

Така задача формулюється наступним чином

$$\Theta(X,\lambda,S,\rho) = f(X) - \sum_{i=1}^{m} \lambda_i s_i \lg\left(S_i - C_i(X)\right) + \sum_{i=m+1}^{m_i} \lambda_i C_i(X) + \frac{p}{2} \sum_{i=m+1}^{m_i} C_i(X)^2 \qquad (7.5)$$

де λ_i - множники Лагранжа; s_i, елементи вектора S, які являються додатнім зсувом і ρ - додатний штрафний параметр. Алгоритм починає працювати таким чином, що початкове значення використовується у вигляді штрафного параметра (**InitialPenalty**).

За допомогою алгоритму безпосередньої оптимізації проводиться мінімізація деякої послідовності з задач оптимізації, що можна розглядати як деяке наближення початкової задачі. Після завершення мінімізації отриманої гессеніани, досягнення заданої точності і виконанні умови реалізованості, проводиться уточнення розрахункових множників Лагранжа. В невдалому випадку, проводиться збільшення штрафного параметра на штрафний коефіцієнт (**PenaltyFactor**). Далі формулюється вже нова гессеніана для початкової задачі мінімізації. Повтор вказаних дій проводиться до тих пір, поки не буде досягнутий будь-який критерій останову.

7.1. Приклад програми

Пошук мінімума нелінійної задачі з обмеженнями з застосуванням **генетичного алгоритму**

Знайти значення X, що мінімізує

$f(X) = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6$, з урахуванням обмежень:

$$x_1^2 x_4 + \sin\left(x_4 - x_5\right) 2\sqrt{2} = 0;$$
$$x_2 - x_3^4 x_4^2 - 8 - \sqrt{2} = 0.$$

```
[x,fval] = ga(@fun,5,[],[],[],[],[],[],@con)

function f = fun(x)
    f = (x(1) - 1)^2 + (x(1) - x(2))^2 + (x(3) - 1)^2 + (x(4) - 1)^4 + (x(5) - 1)^6;
end

function [c,ceq] = con(x)
    c = [];
    ceq = [x(1)^2*x(4) + 2*sqrt(2)*sin(x(4) - x(5)),...
        x(2) - x(3)^4*x(4)^2 - 8 - sqrt(2)];
end
```

```
Optimization terminated: stall generations limit exceeded
 but constraints are not satisfied.

x =

   -1.2600    9.9746    1.1097    0.6116   -2.8833


fval =

   3.5606e+03
```

Завдання

Знайти мінімум функції.

1. $f(x)=4x_1-x_2^2-12;$

$g_1(x)=10x_1-x_1^2+10x_2-x_2^2-34\geq 0;$
$g_2(x)=x_1\geq 0;$
$g_3(x)=x_2\geq 0;$
$x^{(0)}=\begin{bmatrix}2 & 4\end{bmatrix}^T;$

2. $f(x)=100\left(x_2-x_1^2\right)^2+\left(1-x_1\right)^2;$

$g_1(x)=x_1+1\geq 0;$
$g_2(x)=1-x_2\geq 0;$
$g_3(x)=4x_2-x_1-1\geq 0;$
$g_4(x)=1-0.5x_1-x_2\geq 0;$

3. $f(x)=3x_1^2-2x_2;$

$g_1(x)=2x_1+x_2\geq 4;$
$g_2(x)=x_1^2+x_2^2\leq 40;$

4. $f(x)=3\left(x_2-4\right)^2-2x_1;$

$$g_1(x) = 10 - x_1^2 - x_2^2 \geq 0;$$
$$g_2(x) = 9 - x_1^2 - (x_2 - 4)^2 \geq 0;$$
$$0 \leq x_1 \leq 4;$$
$$0 \leq x_2 \leq 4.$$

Комп'ютерний практикум № 8

8. Дослідження оптимальної системи керування з застосуванням варіаційного числення

Відома постановка задачі оптимального керування динамічними системами, що полягає в знаходженні траєкторії $U(t)$ $\{ t_0 < t < t_f \}$, яка приводить стан системи $X(t)$ за допомогою мінімізації функціоналу вартості або якості

$$I = G(X(t), t)\Big|_{t=t_0}^{t=t_f} + \int_{t=t_0}^{t=t_f} F(X(t), U(t), t)\, dt \qquad (8.1)$$

з заданими початковими умовами $X(t) = X_0$ в кінцеву умову $N[X(t_f), t_f] = 0$ оптимальним способом. З точки зору використання є два альтернативних варіанти отримання оптимальних результатів.

У випадку оптимального керування одержуємо оптимальну траєкторію керування, як постійну функцію часу, та в часовому проміжку оптимізації для реального процесу. Очевидно, що ця оптимальна траєкторія залежить від заданого початкового стану X_0. З-за дії можливих перешкод або не точності моделі, на які не зважають при оптимізації, буде перебігати стан $X(t)$, що відрізняється від очікуваного, отриманого по моделі $X(t)$. При цьому також можливе відхилення кінцевих умов.

В випадку оптимального регулювання з необхідних умов випливає рішення так званої синтез-проблеми оптимального закону регулювання

$$U(t) = R^*[X(t), t]$$

Ця формула використовується для прямих розрахунків керуючого впливу $U(t)$ по виміряним величинам станів $X(t)$. Істотна властивість оптимального регулювання випливає з того, що закон регулювання є незалежним відносно можливих початкових станів $\boldsymbol{X_0}$. За відсутності несподіваних перешкод або неточностей моделі оптимальне керування та оптимальне регулювання для заданих початкових станів $\boldsymbol{X_0}$ дає однакові результати. Установлено, що при заданій постановці проблеми використання оптимального закону регулювання з-за своєї універсальності та легкості обчислень з точки зору практичного

використання є більш бажаним, ніж застосування оптимальної траєкторії керування.

Важливим спеціальним випадком оптимального керування динамічними системами є лінійно-квадратична (ЛК) оптимізація (ЛКО), яка пов'язана з лінійними рівняннями станів та квадратичними функціоналами якості.

Розглянемо задачу оптимального керування, припустимо, що потрібно мінімізувати функцію вартості (при фіксованому t_f)

$$I = \frac{1}{2} \int_{t=t_0}^{t=t_f} [X^T(t)Q(t)X(t) + U^T(t)R(t)U(t)]\, dt \tag{8.2}$$

для узагальненої системи з змінними в часі параметрами і яка задається рівнянням

$$X' = A(t)X(t) + B(t)U(t) \tag{8.3}$$

при умові, що $X(t_0)=X_0$ – вектор початкового стану.

Об’єднаємо функцію вартості та обмеження у формі диференційного рівняння за допомогою множника Лагранжа. Тоді

$$I = \int_{t_0}^{t_f} \left\{ \frac{1}{2} X^T(t)Q(t)X(t) + \frac{1}{2} U^T(t)R(t)U(t) + \lambda^T(\mathrm{t})[A(t)X(t) + B(t)U(t) - X'] \right\} dt \tag{8.4}$$

Точний характер використаної функції вартості залежить від вигляду конкретної задачі, що розв`язується. Тому вагові матриці R(t) та Q(t) звичайно обираються з фізичних міркувань. Крім того, без втрати спільності припускається, що R(t) та Q(t) – симетричні. Вектор керування U(t) розглядаємо так само, якби це був вектор стану. Далі скористаємось рівняннями Ейлера-Лагранжа:

$$\frac{\partial \Phi}{\partial X} - \frac{d}{dt}\left(\frac{\partial \Phi}{\partial X'}\right) = 0, \qquad \frac{\partial \Phi}{\partial U} - \frac{d}{dt}\left(\frac{\partial \Phi}{\partial U'}\right) = 0, \tag{8.5}$$

де

$$\Phi = \frac{1}{2} X^T(t)Q(t)X(t) + \frac{1}{2} U^T(t)R(t)U(t) + \lambda^T(\mathrm{t})[A(t)X(t) + B(t)U(t) - X']$$

Тому що

$$\frac{\partial \Phi}{\partial \mathrm{X}} = Q(t)X(t) + A(t)\lambda(\mathrm{t}),\ \ \frac{\partial \Phi}{\partial \mathrm{X}'} = -\lambda(\mathrm{t})$$

$$\frac{\partial \Phi}{\partial \mathrm{U}} = \mathrm{R(t)U}(t) + B(t)\lambda(\mathrm{t}),\ \ \frac{\partial \Phi}{\partial \mathrm{U}'} = 0$$

то рівняння Ейлера-Лагранжа для цієї задачі мають вигляд:

$$\lambda' = -A^T(t)\lambda(t) - \mathrm{Q}(t)\mathrm{X}(t), \tag{8.6}$$

$$U(t) = -R^{-1}(t)\mathrm{B}^T\lambda(\mathrm{t}), \tag{8.7}$$

Через те, що $X(t_f)$ не визначений, умова трансверсальності в момент досягнення

$$\left.\frac{\partial \Phi(t)}{\partial X}\right|_{t_f} = I\left(t_f\right) = 0, \tag{8.8}$$

Відмітимо, що рівняння для оптимального керування можливе при умові, що існує матриця, обернена матриці $R(t)$

Для отримання невід′ємної другої похідної необхідно, щоб R(t) та Q(t) були невід`ємно визначені. Таким чином, зрозуміло, що R(t) повинна бути додатно визначеною.

Стан системи $X(t_0)$ заданий при $t=t_0$, тоді як спряжений вектор λ(t) визначений в момент досягнення $\lambda(t_f)=0$. Таким чином , перш ніж визначати оптимальне керування, необхідно вирішити двоточкову крайову задачу (ДТКЗ).

З′ясуємо, чи можливо перетворити керування (9.6) по замкненому контуру U(t)=K(t)X(t), тобто знайдемо оптимальний закон регулювання. Припустимо, що рішення для спряженого випадку

$$\lambda(t) = P(t)X(t) \tag{8.9}$$

Підставляючи (9.8) в (9.2) та (9.6), отримаємо

$$X' = A(t)X(t) - B(t)R^{-1}B^T(t)P(t)X(t) \tag{8.10}$$

Крім того, з (9.8) та (9.5) випливає

$$\lambda' = P'X(t) + P(t)X' = Q(t)X(t) - A(t)P(t)X(t) \tag{8.11}$$

Об`єднуючи (9.9)та (9.10)одержуємо

$$\lambda' = P'X(t) + P(t)X' = Q(t)X(t) - A(t)P(t)X(t) \quad (8.11)$$

$$[P' + P(t)A(t) + A^T P(t)B(t)R^{-1}B^T P(t) + Q(t)]X(t) = 0 \quad (8.12)$$

Оскільки ця рівність повинна виконуватись для ненульових x(t), то множник [.], що стоїть перед X(t), повинен дорівнювати нулю. Таким чином матриця P, що є симетричною матрицею розмірності (n x n) і яка містить в собі n(n+1)/2 різних членів, повинна задовольняти матричному рівнянню Ріккаті:

$$\mathrm{P}' = -P(t)A(t) - A^T P(t)B(t)R^{-1}B^T P(t) - Q(t) \quad (8.13)$$

при граничній умові $P(t_f)$=0

Таким чином, розв`язання матричного рівняння Ріккаті можливо проводити в зворотному часі, від t_f до t_0, побудувавши матрицю

$$K(t) = -P(t)B^T P(t) \quad (8.14)$$

і потім одержавши керування по замкненому контуру

$$U(t) = K(t)X(t) \quad (8.15)$$

Приклад 8.1:

Синтезувати систему оптимального керування об`єктом

$$T_2 X'' + T_1 X' + X = kU, X(t_0)$$

з критерієм

$$I = \frac{1}{2}\int_0^{t_f} (x^2 + rU^2)dt$$

$$\begin{cases} x'_1 = x_2 \\ x'_2 = -\frac{1}{T_2}x_1 - \frac{T_1}{T_2}x_2 + \frac{k}{T_2}U \end{cases}$$

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{1}{T_2} & -\frac{T_1}{T_2} \end{bmatrix}; B = \begin{bmatrix} 0 \\ \frac{k}{T_2} \end{bmatrix}.$$

Далі використаємо рівняння Ейлера-Лагранжа:

$$\Phi = \frac{1}{2}x_1^2 + \frac{1}{2}rU^2 + \lambda_1(x_2 - x_1') + \lambda_2\left(-\frac{1}{T_2}x_1 - \frac{T_1}{T_2}x_2 + \frac{k}{T_2}U - x_2'\right)$$

$$\frac{\partial\Phi}{\partial \mathrm{x}_1} = x_1 - \frac{1}{T_2}\lambda_2, \qquad \frac{\partial\Phi}{\partial \mathrm{x}_1'} = -\lambda_1, \qquad \lambda_1' = \frac{1}{T_2}\lambda_2 - x_1,$$

$$\frac{\partial\Phi}{\partial \mathrm{x}_2} = \lambda_1 - \frac{T_1}{T_2}\lambda_2, \qquad \frac{\partial\Phi}{\partial \mathrm{x}_2'} = -\lambda_2, \qquad \lambda_2' = -\lambda_1 + \frac{T_1}{T_2}\lambda_2,$$

або

$$\lambda' = A^T\lambda(\mathrm{t}) - \mathrm{Q(t)X(t)}$$

тоді

$$\lambda' = \begin{bmatrix}\lambda_1' \\ \lambda_2'\end{bmatrix} = -\begin{bmatrix}0 & \frac{1}{T_2} \\ 1 & \frac{T_1}{T_2}\end{bmatrix}\begin{bmatrix}\lambda_1 \\ \lambda_2\end{bmatrix} - \begin{bmatrix}1 & 0 \\ 0 & 0\end{bmatrix}\begin{bmatrix}\mathrm{x}_1 \\ \mathrm{x}_2\end{bmatrix}$$

$$\begin{cases}\lambda_1' = \frac{1}{T_2}\lambda_2 - x_2, \ \frac{\partial\Phi}{\partial U} = rU + \lambda_2\frac{k}{T_2} = 0, \\ \lambda_2' = -\lambda_1 + \frac{T_1}{T_2}\lambda_2, \ U = -\frac{1}{r}\frac{k}{T_2}\lambda_2.\end{cases}$$

Для оптимального закону керування

$$U = -\frac{1}{r}\frac{k}{T_2}(p_{12}x_1 + p_{22}x_2)$$

Матричне рівняння Ріккаті

$$P' = -PA - A^TP + PBR^{-1}B^TP - Q$$

запишемо в вигляді

$$\begin{bmatrix}p_{11}' & p_{12}' \\ p_{21}' & p_{22}'\end{bmatrix} = -\begin{bmatrix}p_{11} & p_{12} \\ p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}0 & 1 \\ -\frac{1}{T_2} & -\frac{T_1}{T_2}\end{bmatrix} - \begin{bmatrix}0 & 1 \\ -\frac{1}{T_2} & -\frac{T_1}{T_2}\end{bmatrix}^T\begin{bmatrix}p_{11} & p_{12} \\ p_{21} & p_{22}\end{bmatrix}$$

$$+\begin{bmatrix}p_{11} & p_{12} \\ p_{21} & p_{22}\end{bmatrix}\begin{bmatrix}0 \\ \frac{k}{T_2}\end{bmatrix}r^{-1}\begin{bmatrix}0 \\ \frac{k}{T_2}\end{bmatrix}^T\begin{bmatrix}p_{11} & p_{12} \\ p_{21} & p_{22}\end{bmatrix} - \begin{bmatrix}1 & 0 \\ 0 & 0\end{bmatrix}, \qquad r = 1$$

$$\begin{cases} p'_{11} = \frac{2}{T_2} p_{12} + \left(\frac{k}{T_2}\right)^2 p_{12}^2 - 1, \;\; p_{11}(t_f) = 0 \\ p'_{12} = -p_{11} + \left(\frac{T_1}{T_2}\right) p_{12} + \frac{1}{T_2} p_{22} + \left(\frac{k}{T_2}\right)^2 p_{12} p_{22}, \qquad p_{12}(t_f) = 0 \\ p'_{22} = \frac{2T_1}{T_2} p_{22} + + \left(\frac{k}{T_2}\right)^2 p_{22}^2 - 2p_{12}, \;\; p_{22}(t_f) = 0 \end{cases}$$

$$U^*(t) = -\frac{1}{r}\frac{k}{T_2}(p_{12}x_1 + p_{22}x_2)$$

Постановка задачі оптимального керування динамічними системами має два альтернативних варіанти отримання оптимальних результатів.

У випадку оптимального керування зі зворотнім зв'язком з необхідних умов рішення так званої синтез-проблеми оптимального закону керування. Істотна властивість оптимального регулювання випливає з того, що закон керування є незалежним відносно можливих початкових станів X_0.

Оптимальний лінійний закон керування

$U(t) = -k(t)X(t)$,

де $k(t) = R^{-1}B^T P$ – матриця коефіцієнтів підсилення, P – симетрична матриця коефіцієнтів Ріккаті.

Приклад 9.2

Розглянемо приклад для системи з одним входом і одним виходом третього порядку, що описується рівняннями у просторі стану:

$$\frac{dX}{dt} = A(X - Xd) + B(U - Ud),\; X(0) = \begin{bmatrix} -0{,}1 & 0{,}1 & 0{,}5 \end{bmatrix}^T,$$

де $A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -10 & -60 & -41 \end{bmatrix}$; $B = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$,

і квадратичний критерій якості

$$J = \int_0^\infty \Big((X - Xd)^T Q(X - Xd) + (U - Ud)^T R(U - Ud) \Big) dt,$$

де $Xd = \begin{bmatrix} 0{,}3 & 0{,}2 & 0{,}5 \end{bmatrix}^T$; $Ud = 0{,}3$; $Q = I$; $R = 1/2$.

Наведемо приклад програми для розв'язку цієї задачі з назвою *"Lin_Reg"*.

```
function Lin_Reg
disp('Матриця A:')
A = [0 1 0; 0 0 1; -10 -60 -41] % Введення матриці A
disp('Матриця B:')
B = [0; 0; 2] % Введення матриці B
% Параметри критерія якості:
disp('Матриця Q:')
Q = eye(3) % Формування одиничної матриці
disp('Матриця R:')
R = 1/2 % Введення матриці R
disp('Матриця N:')
N = [] % Введення матриці N
disp('Вектор початкових умов X0:')
x0 = [-0.1 0.1 0.3]' % Завдання вектора початкових умов
disp('Вектор заданних значень відхилень Xd:')
xd = [0.3 0.2 0.5]' % Завдання вектора заданних значень відхилень
disp('Вектор заданних значень відхилень  керування Ud:')
ud = 0.3 % Завдання відхилень значень керування
disp('Час спостереження tf:')
tf = 30 % Завдання часу спостереження
disp('Довжина кроку dt:')
dt = 0.01 % Завдання довжини кроку
disp('Кількість кроків n:')
n = tf/dt
[K P E] = lqr(A,B,Q,R,N); % Синтез регулятора
disp('Матриця зворотного зв''язку K:'); K
disp('Розв''язок рівняння Ріккаті P:'); P
disp('Власні значення замкненої системи E:'); E
% Визначення розмірності задачі
SA = size(A); SA = SA(1); SB = size(B); SB = SB(2);
% Формування векторів x u
x = zeros(SA,n); u = zeros(SB,n-1);
% Формування початкового стану
x(:,1) = x0;
% Знаходження оптимального керування u та змінних стану x
for i=1:n-1,
    u(:,i) = ud-K*(x(:,i)-xd);
```

```
    x(:,i+1) = (A*(x(:,i)-xd)+B*(u(:,i)-ud))*dt+x(:,i);
end
% Побудова динаміки змінних стану
plot(0:dt:tf-dt,x), grid
title('Динаміка руху змінних стану'); xlabel('Час, t');
ylabel('Змінні стану, X(t)'); legend('x_1','x_2','x_3')
% Побудова вектора керування
figure(2)
plot(dt:dt:tf-dt,u), grid
title('Динаміка зміни керування'); xlabel('Час, t'); ylabel('Керування,
U(t)'); legend('u')
```

Результат виконання програми

Матриця A:

A =

0	1	0
0	0	1
-10	-60	-41

Матриця B:

B =

0
0
2

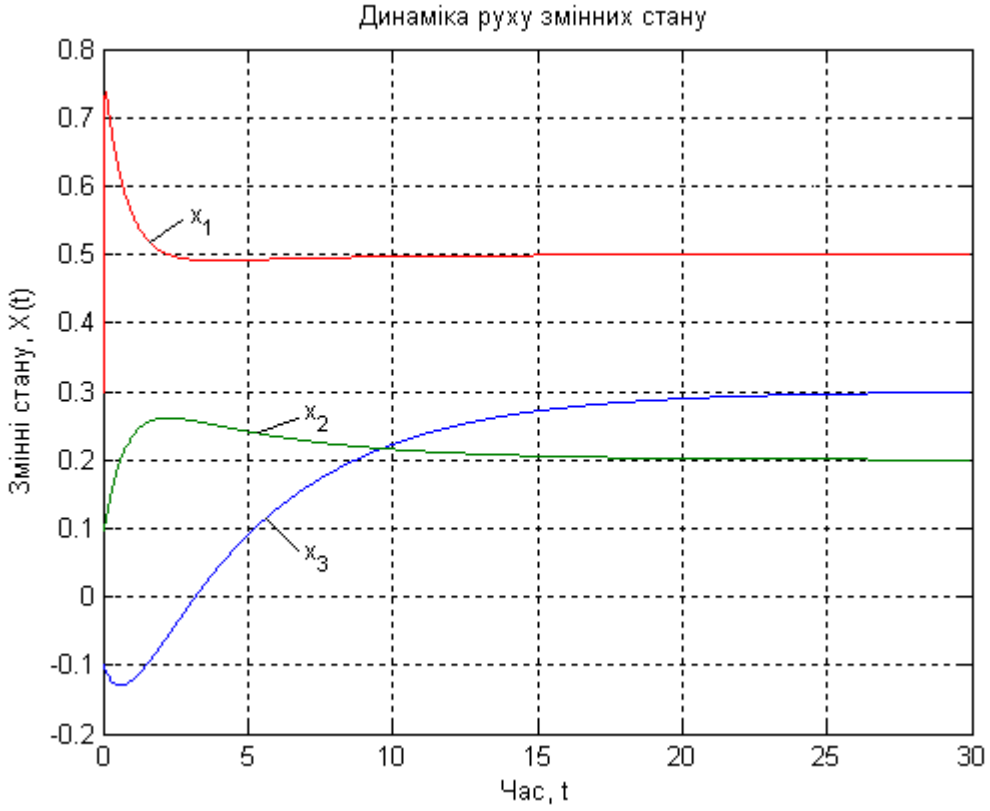
Матриця Q:

Q =

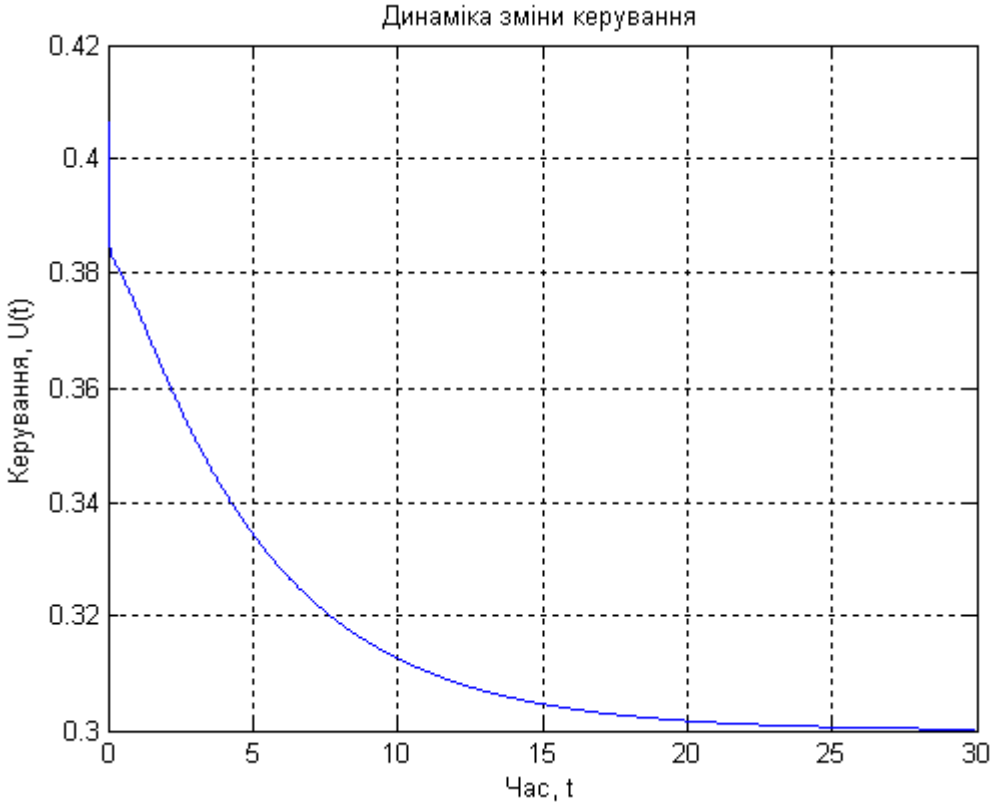
1	0	0
0	1	0
0	0	1

```
Матриця R:

R =

    0.5000

Матриця N:

N =

     []

Вектор початкових умов X0:

x0 =

   -0.1000
    0.1000
    0.3000

Вектор заданих значень відхилень Xd:

xd =

    0.3000
    0.2000
    0.5000

Вектор заданих значень відхилень  керування Ud:

ud =
    0.3000

Час спостереження tf:

tf =
    30
```

```
Довжина кроку dt:
dt =
    0.0100

Кількість кроків n:

n =
        3000

Матриця зворотного з'язку:

K =
    0.1962    0.1760    0.0530

Розв'язок рівняння Ріккаті:

P =

    3.3996    2.1483    0.0490
    2.1483    2.5551    0.0440
    0.0490    0.0440    0.0133

Власні значення замкненої системи:

E =

  -39.5881
   -1.3188
   -0.1990
```



а)



б)

Рис. 8.1. Динаміка руху змінних стану а) і керування б)

Досліджували якість отриманого регулятора для наступних значеннях параметрів: $X(0)=[-0{,}1 \quad 0{,}1 \quad 0{,}5]^T$; $Xd=[0{,}3 \quad 0{,}2 \quad 0{,}5]^T$; $Ud=0{,}3$; $Q=I$; $R=1/2$; $N=0$. Коефіцієнт підсилення K отриманий за допомогою розв'язку алгебраїчного рівняння Ріккаті. Виходи оптимального регулятора, показані на рис. 8.1. а). Оптимальне керування, представлене на рис. 8.1. б), забезпечує мінімальне значення статичної похибки для Xd, Ud за умови, що $t_f \to \infty$, причому R вибрано так щоб для $u(t)$ виконувалася фізична умова невід'ємності. Таким чином, якість отриманого регулятора суттєво залежить від вагових коефіцієнтів Q,R,N.

Завдання

1. Синтезувати систему оптимального керування об`єктом

X'=AX+BU

Варіанти:

а) $A=\begin{bmatrix} 0 & 1 \\ -5 & -3 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

б) $A=\begin{bmatrix} 0 & 1 \\ -4 & -2 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$,

в) $A=\begin{bmatrix} 0 & 1 \\ -10 & -3 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 2 \end{bmatrix}$,

г) $A=\begin{bmatrix} 0 & 1 \\ -7 & -3 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 2 \end{bmatrix}$,

д) $A=\begin{bmatrix} 0 & 1 \\ -15 & -6 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 2 \end{bmatrix}$,

е) $A=\begin{bmatrix} 0 & 1 \\ -20 & -8 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 2 \end{bmatrix}$,

ж) $A=\begin{bmatrix} 0 & 1 \\ -18 & -6 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 2 \end{bmatrix}$,

з) $A=\begin{bmatrix} 0 & 1 \\ -26 & -10 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 5 \end{bmatrix}$,

к) $A=\begin{bmatrix} 0 & 1 \\ -30 & -10 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 10 \end{bmatrix}$,

л) $A=\begin{bmatrix} 0 & 1 \\ -36 & -10 \end{bmatrix}$; $B=\begin{bmatrix} 0 \\ 10 \end{bmatrix}$,

з критерієм у вигляді:

$$\mathbf{I}=\frac{1}{2}\mathbf{X}^T(t_f)\mathbf{S}\mathbf{X}(t_f)+\frac{1}{2}\int_0^{t_f}(\mathbf{X}^T\mathbf{Q}\mathbf{X}+\mathbf{U}^T\mathbf{R}\mathbf{U})dt$$

використавши варіаційне числення.

2. Наведіть основні математичні співвідношення, що визначають алгоритм оптимального керування.

3. Для визначення оптимального керування знайти початкову траєкторію та розв`язати спряжені рівняння в зворотному часі з граничними умовами.

4. Знайти закон оптимального керування **U**(t) та оптимальну траєкторію **X**(t). За результатами розв`язання задачі на ЕОМ побудувати графіки **U**опт(t), **X**(t).

5. Для визначення оптимального закону регулювання розв`язати в оберненому часі рівняння Ріккаті. За результатами розв`язання задачі на ЕОМ побудувати графіки **U**опт(t), **X**(t), **P**(t).

6. Порівняти отримані значення показника якості для оптимального програмного керування та оптимальної системи зі зворотним зв’язком.

7. Сформулювати основні труднощі, що виникають при розв`язанні задачі оптимального керування.

Контрольні запитання

1. Основні положення варіаційного числення.

2. Які комбінації граничних умов можливі ? Умови трансверсальності.

3. Достатні умови існування екстремума.

4. Методи рішення задач з обмеженнями в формі нерівностей.

ДОДАТОК

Синтаксис та опис використання функцій

Функція fminunc

$$\min_x f(x)$$

де *x* є вектор змінних, *f(x)* – цільова функція.

Опис та синтаксис:

fminunc знаходить мінімум скалярної функції декількох змінних, починаючи з деякої початкової точки.

- **x= fminunc(fun,x0)** починає із точки x0 і знаходить локальний мінімум від x для описаної в fun функції. x0 може бути скаляром, вектором або матрицею.
- **x= fminunc(fun,x0, option)** мінімізує з параметрами опції оптимізації.
- [**x,fval] = fminunc(…)** повертає в fval значення цільової функції fun як розв'язок від x.
- **[x,fval,exitflag] = fminunc(…)** повертає значення exitflag, що містить вихідні умови fminunc.
- **[x,fval,exitflag,output] = fminunc(…)** повертає вихідну структуру з гессеніацією fun як розв'язок від x.
- **[x,fval,exitflag,output,grad] = fminunc(…) повертає** в grad значення градієнта fun як розв'язок від x.
- **[x,fval,exitflag,output,hessian] = fminunc(…)** повертає в hessian значення матриці Гессе fun як розв'язок від x.

Функція lsqnonlin

$$\min_x f(x) = f_1(x)^2 + f_2(x)^2 + f_3(x)^2 + \cdots + f_m(x)^2$$

де *L* константа.

Опис та синтаксис:

lsqnonlin вирішує нелінійну задачу методом найменших квадратів, включаючи задачу нелінійного підбора даних.

Для розрахунку значень *f(x)*(“суми квадратів”), *lsqnonlin* вимагає завдання користувачем функції вигляду

$$F(x) = [f_1(x)^2\ f_3(x)^2\ f_3(x)^2 \dots f_m(x)^2\ ]^T.$$

Задача оптимізації може бути заново сформульована як

$$\min_x \frac{1}{2}\|F(x)\|_2^2 = \frac{1}{2}\sum_i f_i(x)^2,$$

де x є вектор змінних, F(x) є функція, що повертає значення вектора.

- **x = lsqnonlin(fun,x0)** починає із точки x0 і знаходить мінімум суми квадратів описаної в fun функції. fun повинна повертати вектор значень, а не суму квадратів значень. (У доданому алгоритмі fun(x) неявно підсумовується й зводиться у квадрат).
- **x = lsqnonlin(fun,x0,lb,ub)** визначає набір нижніх і верхніх меж для проектованих змінних x, так що розв’язок завжди перебуває в діапазоні lb <= x <= ub.
- **x = lsqnonlin(fun,x0,lb,ub,options)** проводить оптимізацію з визначеними параметрами оптимізації. Якщо обмежень не має, то передає порожню матрицю в lb й ub.
- **[x,resnorm]= lsqnonlin(…)** повертає значення квадрата норми нев’язки від x sum(fun(x).^2).
- **[x,resnorm,residual]= lsqnonlin(…)** повертає значення нев’язки fun(x), як розв’язок від x.
- **[x,resnorm,residual,exitflag]= lsqnonlin(…)** повертає значення exitflag, що містить опис вихідних умов lsqnonlin.
- **[x,resnorm,residual,exitflag,output]= lsqnonlin(…)** повертає структурний вихід з інформацією про оптимізацію.
- **[x,resnorm,residual,exitflag,output,lambda] = lsqnonlin(…)** повертає структуру lambda з полями, що містять множники Лагранжа у вигляді розв’язку від x.

- **[x,resnorm,residual,exitflag,output,lambda,jacobian]=lsqnonlin(…)** повертає Якобіан від fun як розв’язок від x.

Функція Fmincon

$$\min_x f(x),$$

за умов

$c(x) \le 0,$

$ceq(x) = 0,$

$Ax \le b,$

$Aeq\ x = beq,$

$lb \le x \le ub,$

де *x, b, beq, lb ub* – вектори, *A* і *Aeq* -матриця , *c(x)* і *ceq(x)* є функції, *f(x)* – цільова функція. *F(x), c(x)* і *ceq(x)* можуть бути нелінійними функціями.

Опис та синтаксис:

fmincon знаходить мінімум цільової функції декількох змінних з обмеженнями починаючи з початкового наближення.

- **x=fmincon(fun, x0,A,b)** починає з точки x0 знаходить мінімум функції fun за умови виконання лінійних нерівностей A*x <=b, x0 може бути скаляром, вектором або матрицею.
- **x=fmincon(fun, x0,A,b,Aeq,beq)** мінімізує fun за умови виконання лінійних рівностей Aeq*x = beq, а так само A*x <=b. Встановлюється A = [] й b=[] у випадках відсутності нерівностей.
- **x=fmincon(fun, x0,A,b,Aeq,beq,lb,ub)** визначає набір нижніх і верхніх обмежень на конструйовані змінні x так, що розв’язок завжди перебуває в діапазоні lb <= x <= ub. Встановлюється Aeq=[] і beq=[] у випадку відсутності рівностей.
- **x=fmincon(fun, x0,A,b,Aeq,beq,lb,ub,unicon)** мінімізує визначені в nonlcon нелінійні нерівності c(x) або рівностей ceq(x) такому оптимуму, що c(x) <=0 й ceq(x) = 0. Встановлюється lb=[] та ub=[] у випадку відсутності обмежень.

- **x=fmincon(fun,x0,A,b,Aeq,beq,Ib,ub,nonicon,options)** проводить мінімізацію з оптимізаційними параметрами, заданими в структурі опцій.
- **[x,fval] = fmincon(...)** повертає значення цільової функції fun як розв'язок від x.
- **[x,fval,exitflag] = fmincon(...)** повертає значення exitflag, що містить опис вихідних умов fmincon.
- **[x,fval,exitflag,output] = fmincon(...)** повертає структурний вихід з інформацією про оптимізацію.
- **[x,fval,exitflag,output,lambda] = fmincon(...)** повертає структуру lambda з полями, що містять множники Лагранжа у вигляді розв'язку від x.
- **[x,fval,exitflag,output,lambda,grad] = fmincon(...)** повертає значення градієнта від fun у вигляді розв'язку від x
- **[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(...)** повертає значення матриці Гессе від fun у вигляді розв'язку від x.

Функція fgoalattain

$\min_{x,\gamma} \gamma,$

за умов

$F(x) - weight \cdot \gamma \leq goal,$

$c\ (x) \leq 0,$

$ceq(x) = 0,$

$Ax \leq b,$

$Aeq\ x = beq,$

$lb \leq x \leq ub,$

де *x*, *weight* (вага), *goal* (ціль), *b*, *beq*, *lb*, *ub* – вектори, *A*, *Aeq* -матриця , *c(x)* і *ceq(x)* є функції, *F(x)* – цільова функція. *F(x), c(x) і ceq(x)* можуть бути нелінійними функціями.

Опис та синтаксис:

fgoalattain вирішує задачу досягнення цілі, яка є одним з формулювань задач для багатокритеріальної оптимізації.

- **x=fgoalattain(fun,x0,goal, weight)** намагається створити цільову функцію, задається як fun і приймаючу певні цільові значення goal шляхом зміни x, починаючи з x0, та вагою weight.
- **x=fgoalattain(fun,x0,goal, weight,A,b)** вирішує задачу досягнення цілі, відповідну системі лінійних нерівностей Ax
- **x = fgoalattain(fun,x0,goal, weight,A,b,Aeq,beq)** вирішує задачу досягнення цілі, відповідну так само системі лінійної рівності Aeq*x= beq. Встановлює визначені A-[] і [], якщо нерівності відсутні.
- **x =fgoalattain(fun,x0,goal, weight,A,b,Aeq beq.lb,ub)** визначає набір нижніх і верхніх обмежень на конструйовані змінні x, так що б шуканий розв'язок знаходилося в діапазоні 1.
- **x = fgoalattain(fun,x0,goal,weight,A,b,Aeq.beq.lb,ub, nonlcon)** підганяє зада- чу досягнення цілі під умови нелінійних нерівностей c(x) або нелінійної рівності ceq(x), що задаються в nonicon. fgoalattain проводить таку оптимізацію, щоб c(x)< 0 і ceq(x)= 0. Встановлює Ib=[] та/або ub-[], якщо немає обмежень.
- **x = fgoalattain(fun,x0,goal, weight,A,B,Aeq,beq.lb,ub, nonlcon,options)** проводить мінімізацію з оптимізаційними параметрами, заданими в структурі опцій.
- [**x,fval] = fgoalattain(...)** повертає значення цільових функцій, обчислю ваних для fun при розрахунку x.
- **[x,fval, attainfactor] = fgoalattain(...)** повертає коефіцієнт досягнення при розрахунку x
- **[x,fval, attainfactor,exitlag] = fgoalattain(...)** повертає значення exitflag, яке характеризує умови задачі fgoalattain.
- **[x,fval, attainfactor,exitflag.output] = fgoalattain(...)** повертає структурний вихід, який містить інформацію про процес оптимізації.
- **[x,fval,attainfactor,exitflag.output,lambda]= fgoalattain(...)** повертає структуру lambda з полями, що містять множники Лагранжа у вигляді розв'язку від x

Функція fseminf

$\min_x f(x)$,

за умови

$c(x) \le 0$,

$ceq(x) = 0$,

$Ax \le b$,

$Aeq\ x = beq$,

$lb \le x \le ub$,

$K_1(x,\omega_1) \le 0$,

$K_2(x,\omega_2) \le 0$,

$\vdots$

$K_n(x,\omega_n) \le 0$,

де *x, b,beq, lb, ub* – вектори, *A, Aeq* -матриця , *c(x)* і *ceq(x)* і $K_i(x,\omega_i)$ є вектор-функції, *f(x)* – цільова функція. *f(x), c(x) і ceq(x)* можуть бути нелінійними функціями. Вектори (або матриці) $K_i(x,\omega_i) \le 0$ є неперервні як від *x*, так і від додаткового набору змінних $\omega_1, \omega_2, \dots, \omega_n$. Змінні $\omega_1, \omega_2, \dots, \omega_n$ є векторами.

Опис та синтаксис:

fseminf знаходить мінімум напівнескінченної цільової функції з обмеженнями декількох змінних, починаючи з початкової точки.

Завдання полягає в тім, що б знайти такий мінімум від *f*(*x*), що б обмеження включали всі можливі значення $\omega_i \in \mathcal{R}^1$ (або $\omega_i \in \mathcal{R}^2$). Оскільки неможливо обчислити всі можливі значення $K_i(x,\omega_i)$, то область повинна бути взята достатня, що б можна було розраховувати підходящі дискретні набори значень.

- **x = fseminf(fun,x0,ntheta,seminfcon)** починає з точки x0 і знаходить мінімум функції fun з напівнескінченними обмеженнями ntheta, визначеними в seminfcon.
- **x = fseminf(fun,x0,ntheta,seminfcon,A,b)** аналогічно намагається задовольнити лінійним нерівностям A*x <= b .

- **x = fseminf(fun,x0,ntheta,seminfcon,A,b,Aeqbeq)** мінімізує об'єкт із урахуванням лінійних рівностей. При відсутності нерівностей встановлюється A=[] й b=[].
- **x = fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq.lb,ub)** визначає набір нижніх і верхніх обмежень на сконструйовані змінні x такі, що розв'язок завжди перебуває в діапазоні lb <= x <= ub.
- **x = fseminf(fun, x0, ntheta, seminfcon, A, b, Aeq. beq, lb, ub, options)** проводить мінімізацію з обліком структурних оптимізаційних параметрів.
- **[x,fval] = fsemin (...)** повертає значення цільової функції fun як розв'язок від x.
- **[x,fval,exitflag] = fseminf(...)** повертає значення exitflag, що містить опис вихідних умов fseminf.
- **[x,fval,exitflag.output] = fseminf(...)** повертає структурний вихід з інформацією про оптимізацію.
- **[x,fval,exitflag.output,lambda] = fseminf(...)** повертає структуру lambda з полями, що містять множники Лагранжа у вигляді розв'язку від x.

Функція fminimax

$$\min_{x} \max_{\{F_i\}} \{F_i(x)\}$$

такі, що

$c(x) \leq 0$,

ceq(x) =0,

$Ax \leq b$,

$Aeq\ x = beq$,

lb ≤x ≤ ub,

де *x, b, beq*, *lb* і *ub* є вектори; *A* й *Aeq* є матриці, *c(x), ceq(x)* і *F(x)* є функції, що повертають вектори. *F(x), c(x)* і *ceq(x)* можуть бути нелінійними функціями.

Опис та синтаксис:

fminimax мінімізує найгірші значення системи функцій багатьох змінних, починаючи з початкової оцінки. Ці значення можуть бути з обмеженнями.

- **x = fminimax(fun,x0)** починає з точки x0 і знаходить розв'язок мінімакса від x для функції fun.
- **x = fminimax(fun,x0,A,b)** вирішує задачу мінімакса з умовою лінійних нерівностей A*x <=b.
- **x = fminimax(fun,x,A,b, Aeq,beq)** вирішує задачу мінімакса з умовою лінійних рівностей Aeq*x= beq. Встановлюється A=[] й b=[] у випадку відсутності рівностей.
- **x = fminimax(fun,x,A,b,Aeq,beq,lb,ub)** визначає набір нижньої й верхньої меж для розрахункових параметрів так, що розв'язок завжди перебуває в діапазоні lb <= x <= ub.
- **x = fminimax(fun,x0,A,b, Aeq,beq,lb,ub, nonlcon)** накладає на задачу мінімакса обмеження типу нерівностей c(x) або рівностей ceq(x), що задають в nonlcon. fminimax мінімізує таким чином, що c(x) <= 0 й ceq(x) = 0. Встановлюється lb=[] та ub=[] у випадку відсутності границь.
- **x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub, nonlcon, options)** проводить оптимізацію з урахуванням параметрів опцій оптимізації.
- **[x,fval] = fminimax(...)** повертає значення цільової функції як розв'язок від x.
- **[x,fval,maxfval] = fminimax(...)** повертає максимальне значення функції як розв'язок від x.
- **[x,fval,maxfval,exitflag] = fminimax(...)** повертає значення exitlag, що містить вихідні умови для fminimax.
- **[x,fval,maxfval,exitflag.output] = fminimax(...)** повертає структурний вихід з інформацією про оптимізацію.
- **[x,fval,maxfval,exitflag,output,lambda] = fminimax(...)** повертає структурну lambda поля якої містять множники Лагранжа, як розв'язок від x.

Опис вихідної умови exitflag:

- **exitflag** > 0-функція збігається до розв'язку від x;
- **exitflag** = 0 - максимальне число оцінок функції або ітерацій було перевищено;
- **exitflag** < 0 - функція не збігається до даного розв'язку.

Структура, що містить інформацію про оптимізацію **output**:

- **iterations** - число виконаних ітерацій;
- **funeCount** - число розрахунків функцій;
- **algorithm** - алгоритм, що використовується.
- **cgiterations** - число PCG ітерацій (тільки для великомасштабного алгоритму);
- **stepsize** - остаточно прийнятий розмір кроку (тільки для середньомасштабного алгоритму);
- **firstorderopt** - вимірювання оптимальності першого порядку: норма градієнта як розв'язок від x.

Структура, що містить множники Лагранжа як розв'язок від x (***розділеному по типах умов***). **Поля структури lambda**:

- **lower** - нижня межа lb;
- **upper** - верхня межа ub;
- **ineqlin** обмеження типу лінійна нерівність;
- **eqlin** - обмеження типу лінійна рівність;
- **ineqnonlin**- обмеження типу нелінійної нерівності;
- **eqnonlin**- обмеження типу нелінійної рівності;
- **maxfval** - максимальне значення функції в точці x, тобто maxfval = max{fun(x)}.

Створення або редагування структури параметрів опцій оптимізації

Опис та синтаксис:

- **options = optimset('param1',value1,'param2', value2,...)** створює структуру параметрів опцій оптимізації, options у якій специфіковані параметри (param) які мають специфіковані значення. Не

специфікований параметр встановлюється як [] (параметри зі значенням [] указують що, коли опції передаються в оптимізаційну функцію, то для даних параметрів використовуються прийняті за замовчуванням значення). Для того, щоб однозначно визначити ім'я параметра, досить тільки набрати визначальні початкові символи. Така операція не допускається для імен параметрів.

- **optimset** без вхідних і вихідних параметрів відображає повний список параметрів з їхніми дозволеними значеннями.
- **options = optimset** (без вхідних аргументів) створює опції з опціонних структур, де всі поля встановлюються як [П.
- **options = optimset(optimfun)** створює опції з опціонних структур з усіма іменами параметрів і прийнятими за замовчуванням значеннями, що ставляться до оптимізаційної функції optimfun.
- **options = optimset(oldopts,'param1', valuel....)** створює копію для oldopts, модифікуючи специфіковані параметри за допомогою специфікованих значень.
- **options - optimset(oldopts,newopts)** сполучає існуючу опціонну структуру oldopts з новою опціонною структурою newopts. Усякий параметр в newopts з недопустимими значеннями перезаписується на відповідні старі параметри в oldopts.

Параметри:

У списку нижче, значення в {} указують на прийняті за замовчуванням значення, при цьому частина параметрів має різні прийняті за замовчуванням значення для різних функцій оптимізації й тому в { } ніяких значень не представлено.

Також можна переглянути параметри оптимізації й прийняті за замовчуванням значення, якщо набрати optimset на командному рядку.

Параметри оптимізації, використовуються в обох великомаштабному й середньомаштабному (стандартному) алгоритмах:

Diagnostics	'on' | {'off'}
Display	'off'| 'iter' | 'final' | 'notify'
GradObj	'on' | {'off'}
Jacobian	'on' | {'off'}
LargeScale	'on' | {'off'}
MaxFunEvals	Додатне ціле
Maxlter	Додатне ціле
TolCon	Додатний скаляр
TolFun	Додатний скаляр
Tol	Додатний скаляр

Параметри оптимізації, використовувані тільки у великомаштабному алгоритмі:

Hessian	'on' | {'off'}
HessMult	function | {[]}
HessPattern	Розріджена матриця | {розріджена матриця}
Jacob Mult	function | {[]}
JacobPattern	Розріджена матриця | {розріджена матриця}
MaxPCGIter	Додатне ціле | {більше 1 і не перевищує (n/2)}, де n число елементів x0, тобто для початкової точки
PrecondBandwidth	Додатне ціле | {0} |laf
TolPCG	Додатній скаляр | (0.1)
Typical	Вектор від будь-яких значень

Параметри оптимізації, використовувані тільки в середньомаштабному (стандартному) алгоритмі:

DerivativeCheck	'on' | {'off'}
DiffMaxChange	Додатний скаляр | {1e-1}
DiffMinChange	Додатний скаляр | {1e-8}
GoalsExactAchieve	Додатне скалярне ціле | {0}
GradConstr	'on' | {'off'}
HessUpdate	{'bfgs'}|'dfp' | 'steepdesc'

LevenbergMarquardt	'on' | {'off'}
LineSearchType	'cubicpoly' | {'quadcubic'}
MertFunction	'singleobj' | {multiobj'}
MinAbsMax	Додатній скалярне ціле | {0}

Параметри опцій оптимізації

Дана таблиця описує поля структури параметрів оптимізації. Наведені значення можуть бути встановлені за допомогою команди optimset.

Таблиця. Параметри опцій оптимізації.

Ім’я параметра	Опис	Використовується у функціях
DerivativeCheck	Порівняння введених користувачем значенням похідних (градієнти або якобіани) з кінцево-різницевими значеннями.	fgoalattain, fmincon, fminimax, fminunc, fseminf, lsquonlin.
Diagnostics	Вивід діагностичної інформації про мінімізовану або розв'язувану функції.	Всі функції.
DiffMaxChange	Максимальна зміна змінних кінцево- різницевих градієнтів.	fgoalattain, fmincon, fminimax, fminunc, fseminf, lsquonlin.
DiffMinChange	Мінімальна зміна змінних кінцево-різницевих градієнтів.	fgoalattain, fmincon, fminimax, fminunc, fseminf, lsqcurvefit, lsqnonlin.
Display	Рівень відображення. 'off' - відображення не виводиться; 'iter' - відображення виводиться на кожній ітерації; 'final' відображення тільки вихідної інформації, 'notify' -відображення виводиться тільки у випадку, якщо функція не збігається.	Всі функції.
GoalsExactAchieve	Визначає число цілей goals, які досягнуті "точно" (не ставиться до того, що перевизначено перевизначено або не досягнуто).	fgoalattain.
GradConstr	Градієнт для визначених користувачем нелінійних обмежень.	fgoalattain, fmincon, fminimax.
GradObj	Градієнт для визначених користувачем цільових функцій.	fgoalattain, fmincon, fminimax, fminunc, fseminf.
Hessian	У випадку установки 'on' в fmincon використовується задана користувачем матриця Гессе (при використанні HessMult) для цільової функції. У випадку 'off' функція використовує	fmincon, fminunc.

	апроксимацію матриці Гессе за допомогою кінцевих різниць.	
HessMult	Визначена користувачем функція для множення на матрицю Гессе.	fmincon, fminunc.
HessPattern	Розріджений шаблон матриці Гессе для кінцевих різниць. Розмірність матриці n × n, де n- число елементів в точці x0 (початкова точка).	fmincon, fminunc.
HessUpdate	Удосконалена квазіньютонівська схема.	fminunc.
Jacobian	При установці 'on', використовується заданий користувачем Якобіан, для цільової функції. При установці 'off', функція апроксимує допомогою кінцевих різниць.	lsqnonlin.
JacobMalt	Задана користувачем функція для множення на Якобіан.	lsqcurvefit, lsqnonlin.
JacobPattern	Розріджений шаблон Якобіана для кінцевих різниць. Розмірність матриці m×n, де число значень у першому аргументі, що повертає функція fun, - число елементів в точці x0 (початковій точці).	lsqcurvefit, lsqnonlin.
LargeScale	Якщо можливо, то використовується алгоритм великої розмірності.	lsqcurvefit, lsqnonlin.
LevenbergMarquardt	Вибір алгоритму Левенберга-Макуарда замість Гаусса-Ньютона.	fmincon, fminunc, lsqnonlin.
LineSearchType	Вибір алгоритму лінійного пошуку.	lsqnonlin.
MaxFunEvals	Максимальне число допустимих значень функції.	fminunc, lsqnonlin.
MaxIter	Максимальне число допустимих ітерацій.	fgoalattain, fmincon, fminimax, fminunc, fseminf, lsqnonlin.
MaxPCGIter	Максимальне число допустимих PCG ітерацій.	Всі функції.
MeritFunction	Використається цільова функція досягнення, fminimax (багатоцільова), а для fgoalattain (одно цільова).	fmincon, fminimax, fminunc, fseminf, lsqnonlin.
MinAbsMax	Число функцій $F(x)$, що підлягають мінімізації в найгіршому випадку для абсолютних значень.	fmincon, fminunc, lsqnonlin.
PrecondBandWidth	Верхня смуга для попередньої обробки для PCG.	fmincon, fminunc, Isqnonlin.
TolCon	Кінцеве припустиме відхилення по порушенню умов обмеження.	fgoalattain, fmincon, fiminimax, fseminf.

TolFun	Кінцеве припустиме відхилення за значенням функції.	fgoalattain, fmincon, fminimax, fminunc, fseminf, lsqnonlin.
TolPCG	Кінцеве припустиме число ітерацій PCG.	fmincon, fminunc, Isqnonlin.
Tol	Кінцеве припустиме відхилення по x.	Всі функції
Typical	Типові значення x. Довжина вектора дорівнює числу елементів в x0 (початковій точці).	fmincon, fminunc, Isqnonlin.

Функція вивода

Поле Outputfcn структури опцій визначає деяку функцію з М-файлу, до якої на кожній ітерації відбувається звертання з програми оптимізації. Як правило, функцію вивода використовують для малювання реперних точок на кожній ітерації або для деякого відображення даних з розрахункового алгоритма. Для встановлення функції вивода необхідні наступні процедури:

1. Написати функцію вивода у вигляді М-файла або підпрограми.
2. За допомогою команди optimset встановлюється значення параметра Outputfen у вигляді деякого покажчика функцій, тобто перед ім'ям функції повинен стояти знак. Наприклад, якщо функція виводу outfun.m, то відповідна команда буде

options=optimset('OutputFcn', @outfun);

яка встановлює параметр OutputFcn як покажчик функції outfun.

3. Викликати функцію оптимізації з відповідними опціями в якості вихідних аргументів.

Структура функції вивода

Відповідно з визначенням, командний рядок функції вивода має наступну форму

stop = outfun(x, optimalues, state, varargin);

де

- x - є вибрана точка, яка розраховується на кожній ітерації відповідно прийнятого в даній задачі алгоритма.

- optimValues - є деяка структура, що включає в себе дані з поточної ітерації.
- state - є поточний стан алгоритма.
- varargin - включає в себе вхідні аргументи від інших залежних задач, які можуть бути передані в функцію outfun.
- stop - є деякий прапорець, приймаючий значення "Істина" або "Хибність" в залежності від того, потрібно або непотрібно виходити з програми оптимізації.

Функція оптимізації на кожній ітерації передає відповідні значення вхідних аргументів в програму outfun.

Відповідні вихідні аргументи

Деякі з заданих в optimValues полей відповідають вихідним аргументам функції оптимізації. Після виконання останньої ітерації для заданного алгоритма оптимізації, в значення кожного такого поля заноситься відповідні вихідні аргументи. Наприклад, параметр optimValues.fval відповідає вихідному аргументу fval. Таким чином, при зверненні до функції fmincon та при наявності вихідної функції й повертаємого параметра fval, остаточне значення параметра optimValues.fval буде дорівнювати fval. В стовпчику "Опис" наведений нижче таблиці приводиться опис полей, які мають відповідні вихідні аргументи.

Відображення командного рядка

Значення деяких полів параметра optimValues відображаються командному рядку у випадку звертання до функції оптимізації із встановленим в опціонним параметром відображення Display як 'iter', Вивід *ітераційного відображення*. Наприклад, параметр optimValues.fval виводить в стовпчик f(x). Далі в таблиці приводяться значення полів, які можуть відображатися в командному рядку.

Поле оптимальних значень (optimValues.field)	**Опис**	**Провертається з функції**	**Відображення командного рядка**
cgiterations	Число ітерацій спряжених градієнтів на кожній ітерації. Остаточне	fmincon, Isqnonlin.	PCG - iteration

	значення дорівнює результату дії функції оптимізації output.cgiterations.		
constrviolation	Порушення максимального обмеження.	fgoalattn, fmincon, fminimax, fseminf.	constrviolation
degenerate	Визначення наявності виродження. Деяка точка вважається виродженою, якщо в даній точці частинна похідна дорівнює нулю.	fmincon, Isqnonlin.	degenerate
directionalderivative	Похідна по напрямку в напрямку пошуку.	fgoalattain, fimincom, fminimax, fminunc, fseminf, Isqnonlin.	directionalderivative
firstorderopt	Оптимальність першого порядку (залежить від виду алгоритму). Остаточне значення дорівнює значенню output.firstorderopt за виводом функції оптимізації.	fgoalattain, fimincom, fminimax, fminunc, fseminf, Isqnonlin.	firstorderopt
funcount	Загальне число розрахунків функції. Остаточне значення дорівнює значенню output.funCount за виводом функції оптимізації.	fgoalattain, fimincom, fminimax, fminunc, fseminf, Isqnonlin.	funcount
fval	Значення функції в поточній точці. Остаточне значення дорівнює вихідному значенню функції оптимізації fval.	fgoalattain, fimincom, fminimax, fminunc, fseminf, Isqnonlin.	f(x)
gradient	Поточне значення градієнта цільової функції - або його аналітичне значення, якщо воно задано, або його кінцево-різницевою апроксимацією. Остаточне значення дорівнює вихідному значенню функції оптимізації grad.	fgoalattain, fimincon, fminimax, fminunc, fseminf, lsqnonlin.	gradient
iteration	Число ітерацій – початкове значення дорівнює 0. Остаточне значення дорівнює	fgoalattain, fminbnd, fimincon, fminimax, fiminsearch,	Iteration

	вихідному значенню функції оптимізації outpuy.iterations.	fminunc, fseminf, lsqnonlin.	
lambda	Значення множників Лагранжа в точці розв'язку x. lambda Є структурою, де кожне поле відведене для різних типів обмежень. Остаточне значення дорівнює вихідному значенню функції оптимізації lambda.	fgoalattain, fmincon, fiminimax, fseminf Isqnonlin.	lambda
positivedefinite	Дорівнює нулю, якщо в алгоритмі буде визначене від'ємне значення кривизни при розрахунку кроку Ньютона, на першому протилежному випадку	fmincon, Isqcurvefit, Isqnonlin.	positivedefinite
procedure	Процедурне посилання.	fgoalattain, fmincon, fminimax, fseminf.	procedure
ratio	Відношення зміни в даній цільовій функції до зміни в квадратичній апроксимації.	fmincon, Isqnonlin.	ratio
residual	Квадрат норми нев'язки.	Isqnonlin.	residual
searchdirection	Напрямок пошуку.	fgoalattain, fmincon, fminimax, fminunc, fseminf, lsqnonlin.	searchdirection
stepsize	Поточний розмір кроку. Остаточне значення дорівнює вихідному значенню функції оптимізації options.stepsize.	fgoalattain, fmincon, fminimax, fminunc, fseminf, lsqnonlin.	stepsize
trustregionradius	Радіус довірчої області.	fmincon, lsqnonlin.	trustregionradius

Виродженість. Значення поля виродженості, які служать мірою ступені виродженості для поточного стана процесу оптимізації в деякій точці x і визначається наступним чином. По-перше, визначимо вектор r, що має таку ж розмірність, як і x, при цьому r(i) є мінімальна відстань від точки x(i) до *i*-ої *складової верхньої й нижньої межі, відповідно*, lb та ub

r = min(abs(ub-x, x-lb))

Тоді ступінь виродженості буде являти собою мінімальну компоненту вектора r+abs(grad), де grad є градієнт даної цільової функції. Ступінь виродженості дорівнює 0, якщо є деякий індекс *i*, для якого справедливі обидва вирази

grad(i) = 0

x(i) дорівнює *i*-й компоненті або верхній (або нижній межі).

Стан алгоритма

Нижче в таблиці наведено список можливих значень стану

Стан	**Опис**
'init'	Алгоритм знаходиться в стані ініціалізації перед першою ітерацією.
'interrupt'	Алгоритм знаходиться в області деякого обчислюваного достатньо довгого процесу. В цьому стані звертання до функції вивода може викликати переривання поточної ітерації оптимізації. В цей час, значення величин x та optim Values мають ті ж значення, що й при останньому зверненні до функції Вивода при стані 'iter'
'iter'	Алгоритм знаходиться в кінцевій стадії деякої ітерації.
'done'	Алгоритм знаходиться в кінцевій стадії після виконання останньої ітерації.

Наведені нижче коди являють собою ілюстрацію того, як функція вивода може використовувати типи стану для опису того, як задачі виконуються в даній поточній ітерації.

```
switch state
case 'iter'
% викликає корегування відображення на графіку або в
зовнішньому виді, залежності від необхідності
case 'interrupt'
% імовірно, що в даному випадку немає ніякої дії, перевірка
умови,
% необхідності вихода з оптимізації
case 'init'
% Звертання до графічного відображення або до будь-якої
іншої зміни зовнішнього стану
case 'done'
% обнулення графіків, або зовнішнього стану
```

```
otherwise
end
```

Прапорець зупинки. Зупинка по вихідним аргументам виконується за допомогою деякого прапорця приймаючого значення "істина" або "хибність". Наведемо нижче приклади, що представляють типові способи використання прапора зупинки.

Зупинка операції оптимізації на основі даних в параметрі optimValues. Вихідна функція може зупинити процес оптимізації при виконанні будь-якої ітерації в залежності від поточних значень параметра optimValues. Наприклад, наступний набір програмних кодів присвоює параметру зупинки значення істина у випадку, якщо похідна по напрямку приймає деяке значення, меньше чим 0,01.

```
function stop = outfun(x, optimValues)
stop = false;
%перевірка дійсності, похідна по напрямку менше, чим 0,01.
if optimValues.directionalderivative < .01
stop = true;
end
```

Зупинка оптимізації по вводу з GUI. Якщо необхідно створити GUI (графічний інтерфейс користувача) для виконання операції оптимізації, то, можливо, зробити зупинку з функції вивода при натисканні користувачем кнопки Stop на панелі GUI. Настуні коди демонструють як це можна зробити. Припустимо, що кнопка Stop звертається до підпрограми зберігання значення <істина в поле optimstop для виклику за допомогою hObject структури вказників.

```
function stop = outfun(x)
stop = false;
%перевірка, по користувач вимагає зупинку оптимізації
stop = getappdata(hObject,'optinstop');
```

Функція ga

ga знаходить мінімум функції придатності за допомогою генетичного алгоритму в командному рядку.

Опис та синтаксис:

- **x - ga(fitnessfun, nvars)** знаходить мінімум функції придатності за допомогою генетичного алгоритму, де fitnessfun функція придатності, nvars - число незалежних змінних для функції придатності.
- **x = ga(fitnessfun, nvars, options)** знаходить мінімум функції придатності за допомогою генетичного алгоритму, з урахуванням параметрів опцій генетичного алгоритму.
- **x = ga(problem)** знаходить мінімум функції придатності за допомогою генетичного алгоритму, що має структуру з трьох полів: **fitnessfen, nvars, options**.
- **[x, fval] = ga(...)** повертає значення функції придатності від x.
- **[x, fval, reason] = ga(...)** повертає причину зупинки алгоритму.
- **[x, fval, reason, output] = ga(...)** повертає структуру, що містить висновок від кожної генерації і іншої інформації про продуктивність алгоритму.

Вихідна структура містить наступні поля:

randstate - генератор випадкового числа MATLAB (тільки перед запуском алгоритму).

randnstate - нормальний генератор випадкового числа MATLAB (тільки перед запуском алгоритму).

generations - число обчислених генерацій.

funccount - число оцінок функції придатності.

message - повідомлення (причина припинення алгоритму).

- **[x, fval, reason, output, population] = ga(...)** повертає заключне сімейство.
- **[x, fval, reason, output, population, scores] = ga(…)** повертає відмітки заключного сімейства.

Створення або редагування структури опцій генетичного алгоритму

Опис та синтаксис:

- **options = gaoptimset**(без вхідних аргументів) створює структуру опцій де всі параметри мають початкові значення.

- **options = gaoptimset('param1', value1,'param2', value2...)** створює структуру опцій і встановлює значення 'param1' до value1, 'param2' до value2, і так далі.
- **options = gaoptimset(oldopts,'param1', valuel...)** створює копію oldopts, змінюючи конкретизовані параметри з конкретизованими значеннями.
- **options = gaoptimset(oldopts,newopts)** об'єднує існуючу структуру опцій, oldopts, з новою структурою опцій, newopts. Будь-які параметри в newopts з непорожніми значеннями переписують відповідні старі параметри в oldopts.

Наступна таблиця складає список опцій, які можна встановити за допомогою наступною gaoptimset. Значення в { } означає значення за умовчанням. Параметри оптимізації можна проглянути за допомогою команди gaoptimset.

Опція	Опис	Значення
CreationFen	Створення початкової популяції.	{@gacreationuniform}
Crossover Fraction	Частинка покоління в наступній генерації, (дочірні елітні не включені в функцію, що створює crossover).	Додатне ціле | (0.8)
CrossoverFen	Керуюча функція, що використовує алгоритм, щоб створити дочірній crossover.	@crossoverheuristic{@crossoverscattered} @crossoverintermediate @crossoversinglepoint @crossovertwopoint
EliteCount	Додатне ціле число, що визначає, скільки індивідуумів в поточному поколінні, гарантовано, витримають до наступного покоління.	Додатне ціле | {2}
FitnessLimit	Функція придатності досягла значення FitnessLimit - зупинка алгоритму.	Скалярний|{-Inf}
FitnessScalingFen	Масштабує значення функції придатності.	@fitscalinggoldberg{@fitscalingrank} @fitscalingprop @fitscalingtop
Generations	Додатне ціле число, що визначає максимальне число	Додатне ціле число |{100}

	повторень перед зупинкою алгоритму.	
PopInitRange	Додатне ціле число, що визначає максимальне число повторень перед зупинкою алгоритму.	Матриця або вектор | [0,1]
PopulationType	Матриця або вектор, що конкретизує ряд індивідуумів в початковому поколінні.	'bitstring' | 'custom' | {'doubleVector')
HybridFen	Опис типа даних поколінь.	{[]}
Initial Population	Функції, що продовжує оптимізацію після закінчення ga.	Додатне ціле | {[]}
InitialScores	Початкове покоління.	Вектор стовпчик | {[]}
MigrationDirection	Початкові відмітки.	'both' | ('forward")
MigrationFraction	Число між 0 і 1 конкретизація частини індивідуумів в кожному підпоколінні, яке мігрує до іншого підпокоління.	Скаляр | {0.2}
MigrationInterval	Додатне ціле число, що конкретизує число генерацій, які мають місце між міграціями індивідуумів між підпоколіннями.	Додатне ціле число | {20}
MutationFen	Мутаційні дочірні значення.	@mutationuniform {@mutationgaussian)
OutputFens	Масив значень функцій, що да, заходить на кожній ітерації.	Масив | {[]}
OutputInterval	Додатне ціле число, що конкретизує число генерацій між послідовними викликами до вихідних функцій.	Додатне ціле число | {1}
PlotFens	Масив значень функцій, що обчислені алгоритмом.	@gaplotbestf @gaplotbestgenome @gaplotdistance @gaplotexpectation @gaplotgeneology @gaplotselection @gaplotrange @gaplotscorediversity @gaplotscores @gaplotstopping | ([])
PlotInterval	Додатне ціле число, що конкретизує число генерацій між послідовними викликами до функцій плану.	Додатне ціле число | {1}
PopulationSize	Розмір покоління.	Додатне ціле число | {20}

SelectionFen	Функції, що вибирають дочірні кроссоверні і	@selectiongoldberg @selectionrandom {@selectionstochunif} @selectionroulette @selectiontournament
StallLimitG	Зупинка алгоритму, якщо немає поліпшення в функції придатності для послідовних генерацій StallLimitG.	Додатне ціле число | {50}
Stall Limits	Зупинка алгоритму, якщо немає поліпшення в функції придатності протягом секунд StallLimits.	Додатне число | {20}
TimeLimit	Алгоритм зупиняється після запуску протягом секунд TimeLimit.	Додатне число | {30}
Vectorized	Визначає, чи є функції векторизованими.	'on' | ('off'}

Функція patternsearch

patternsearch знаходить мінімум цільової функції, використовуючи прямий пошук.

- **x = patternsearch(@fun, x0)** знаходить мінімум цільової функції, де fun - цільова функція; x0 - початковою точка.

$$\min_x f(x)$$

- **x = patternsearch(@fun, x0, A, b)** знаходить мінімум цільової функції з урахуванням лінійних обмежень типу нерівність

$$Ax \leq b,$$

де *A* - матриця розміром $m \times n$ вектор довжиною *m* (якщо не має то A = [] і b= []).

- **x = patternsearch(@fun, x0, A, b, Aeq, beq)** знаходить мінімум цільової функції з урахуванням лінійних обмежень типу нерівність та рівність

$$Ax \leq b,$$

$$Aeq\ x = beq,$$

де *Aeq* - матриця розміром $r \times n$; *beq* вектор довжиною *r* (якщо не має то *Aeq*=[] і *beq* = []).

- **x = patternsearch(@fun, x0, A, b, Aeq, beq, Ib, ub)** знаходить мінімум цільової функції з урахуванням лінійних обмежень типу нерівність та рівність та меж

$$Ax \leq b,$$

$$Aeq\ x = beq,$$

$$\text{lb} \leq \text{x} \leq \text{ub},$$

де lb ub верхня та нижня межа відповідно (якщо не задані то приймаються *inf* та - *Inf* відповідно).

- **x = patternsearch(@fun, x0, A, b, Aeq, beq, b, ub, options)** знаходить мінімум цільової функції з урахуванням структури опцій.
- **x = patternsearch(problem)** знаходить мінімум цільової функції з усіма вище наведеними полями.
- **[x, fval] = patternsearch(...)** повертає значення цільової функції як розв'язок від x.
- **[x, fval, exitflag] = patternsearch(...)** повертає значення exitflag, що описує вихідні умови patternsearch.

exitflag > 0 - функція збігається до розв'язку від x;

exitflag = 0 - максимальне число оцінок функції або ітерацій було перевищено;

exitflag < 0 - функція не збігається до даного розв'язку.

- **[x, fval, exitflag, output] = patternsearch(@fun, x0 ...)** повертає вихідну структуру з інформацією про оптимізацію.

  Вихідна структура містить наступні поля:

function - цільова функція;

problemtype - зв'язані обмеження або лінійні обмеження;

pollmethod - метод проведення голосування;

searchmethod - пошуковий метод;

iteration - ітерації;

funccount - число оцінок функції;

meshsize - розмірність x;

message - повідомлення (причина зупинки алгоритму).

Створення або редагування структури опцій методу прямого пошуку

Опис та синтаксис:

- **options = gaoptimset** (без вхідних аргументів) створює структуру опцій де всі параметри мають початкові значення.

- **options=psoptimset('param1', value1,'param2', value2...)** створює структуру опцій і встановлює значення 'param1' до valuel, 'param2' до value2, і так далі.
- **options psoptimset(oldopts,'param1', value1...)** створює копію oldopts, змінюючи конкретизовані параметри з конкретизованими значеннями.
- **options = psoptimset(oldopts,newopts)** об'єднує існуючу структуру опцій, oldopts, з новою структурою опцій, newopts. Будь-які параметри в newopts з непорожніми значеннями переписують відповідні старі параметри в oldopts.

Наступна таблиця складає список. опцій, які можна встановити за допомогою наступною gaoptimset. Значення в { } означають значення за умовчанням. Параметри оптимізації можна проглянути за допомогою команди psoptimset.

Опції	**Опис**	**Значення**
Cache	Встановлення 'on', patternsearch має історію отриманих точок - це проводить голосування - і не проводить голосування точок, знаходяться поблизу до них в подальших ітераціях. Використовується, якщо patternsearch рухається поволі, щоб зменшити xfc обчислення цільової функції.	'on' | ('off')
CacheSize	Розмір історії.	Додатне число |{1e4}
CacheTol	Визначає, як близько поточна точка сітки має бути близькою до точки, що мається в історії patternsearch.	Додатне число |{1e-10}
Complete Poll	Повне опитування кожної ітерації.	'on' | ('off')
CompleteSearch	Повне опитування під час пошуку.	'on' | ('off')
Display	Рівень відображення.	'off |'iter' | 'notify | 'diagnose' | {final")

InitialMeshSize	Початковий шаблон розміру методу безпосереднього пошуку.	'off |'iter'	'notify	'diagnose'	{final'')
MaxFunEvals	Максимальне число розрахунків цільової функції.	Додатне число	{1.0}
Maxlter	Максимальне число ітерацій.	Додатне ціле число		{2000*numberOfVariables}
MaxMeshSize	Максимальний розмір розв'язку.	Додатне ціле число	{100*numberOfVariables}
MeshAccelerator	Прискорення конвергенції поблизу мінімуму.	Додатне число	{Inf}
MeshContraction	Коефіцієнт стиснення. Використовується, при невдалій ітерації.	'on'	{'off')
MeshExpansion	Коефіцієнт розширення. Використовується, при успішній ітерації.	Додатне число	{0.5}
OutputFcn	Визначає визначену користувачем цільову функцію на кожній ітерації.	Додатне число |{0}
PlotFen	Визначає графіки виводу для кожного пошуку.	@psoutputhistory {_none}
PlotInterval	Визначає номер ітерації між запитами до графічних функцій.	@psplotbestf	@psplotmeshsize	@psplotfuncount	{[]}
PollingOrder	Порядок напрямків опиту в пошуку зразка.	Додатне ціле число
PollMethod	Опитування стратегії, що використовується в пошуку зразка.	'Random'	'Success'	{'Consecutive'}
ScaleMesh	Автоматичне масштабування змінних.	{'PositiveBasis2N'}|'PositiveBasisNp1'
SearchMethod	Тип пошуку, що використовується в пошуку зразка.	('on')	'off
TolBind	Обов'язковий доступ.	Додатне число |{1e-3}
TolCon	Похибка обчислення обмеження.	Додатне число |{1e-6}
TolFun	Похибка обчислення цільової функції.	Додатне число |{1e-6}
TolMesh	Похибка обчислення розміру сітки.	Додатне число	{1e-6}

TolX	Похибка обчислення змінної.	Додатне число |{1e-6}
Vectorized	Визначає, чи є функції векторизованими.	'on' | ('off')

Графічне відображення інформації

MATLAB має широкі можливості для графічного відображення векторів і матриць, а також створення коментарів та друку зображення.

Функції графічного відображення інформації автоматично відкривають нове вікно зображення, якщо до цього його не було на екрані. Якщо ж вікно існує, то воно використовується за замовчуванням.

Створення графіка

Функція *plot* має різноманітні форми, пов'язані з вихідними параметрами, наприклад *plot*(*y*) створює кусково-неперервний графік залежності елементів y від їх індексів. Якщо задати два вектори в якості аргументів, *plot*(*x*,*y*) створює графік залежності *y* від *x*.

Функція *ezplot*(*f*,[*xmin, xmax, ymin, ymax*]) створює графік функції, без формування користувачем відповідних векторів.

Наприклад, для створення графіка значень функції sin(*x*), $x \in [0,2\pi]$, запишемо наступне

```
x = 0:0.01:2*pi;
y = sin(x);
plot(x,y), grid on
```

Функції mesh, surface, meshc, surface, contour

MATLAB визначає поверхню як *z* координати точок над координатною сіткою площини *x* - *y*, використовуючи прямі лінії для з'єднання сусідніх точок. Функції *mesh* та *surface* відображують поверхню у тривимірному просторі. При цьому *mesh* створює каркасну поверхню, де кольорові лінії сполучають дані точки, а функція *surface* разом з лініями відображує в кольорі саму поверхню. Функції *meshc, surfac* відображують ще й лінії рівня під поверхнею, а функція *contour* відображує лінії рівня.

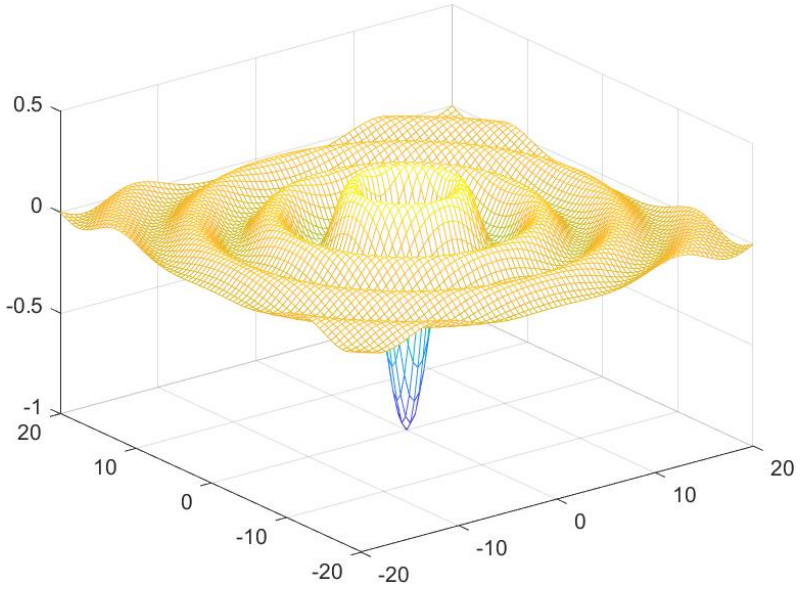
Візуалізація функцій двох змінних

Для відображення функцій двох змінних, $z = f(x,y)$, створюється матриці X та Y, які складаються з рядків та стовпчиків, що повторюються відповідно, перед використанням функції. Потім використовуються ці матриці для обчислення та відображення функції. Функція *meshgrid* перетворює область визначення, задану через один вектор або два вектори x та y, в матриці X та Y для використання при обчисленні функцій двох змінних. Рядки матриці X дублюють вектор x, а стовпчики Y – вектор y.

Розглянемо декілька прикладів:

Побудова поверхні функцій $z = -\frac{\sin\left(\sqrt{x^2+y^2}\right)}{\sqrt{x^2+y^2}}$ (двовимірний випадок -*sinc*(*r*))

```
[X,Y] = meshgrid(-20:0.5:20);
R= sqrt(X.^2+Y.^2)+eps;
Z= -sin(R)./R;
mesh(X,Y,Z);
```

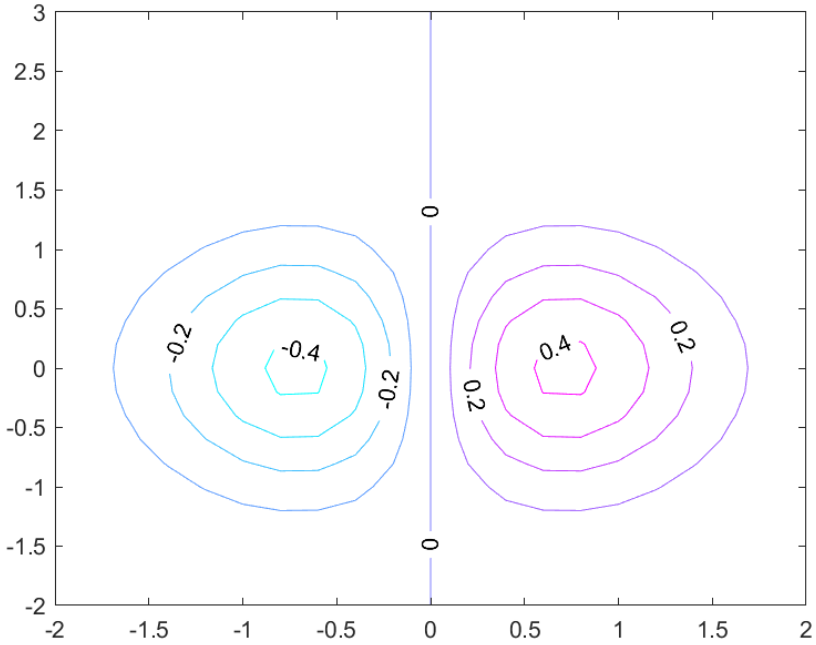


В цьому прикладі R - це відстань від початку координат, якому відповідає центр матриці. Додавання *eps* дозволяє уникнути невизначеності 0/0 в початку координат.

Побудова ліній рівня функції $z = xe^{-x^2-y^2}$, при-2 ≤ x ≤ 2 та -2 ≤ y ≤ 3

```
[X,Y] = meshgrid(-2:0.2:2,-2:.2:3);
Z= X.*exp(-X.^2-Y.^2);
[C,h] = contour(X,Y,Z);
```

```
set(h, 'ShowText', 'on', 'TextStep' , get(h,
'Levelstep')*2)
colormap cool
```



Розглянемо ще деякі функції які не потребують попереднього формування матриць:

ezmeshc(x,y,z,[min,max]) відображує поверхню та лінії рівня функції двох змінних;

ezmesh(x,y,z,[min,max]) відображує поверхню функції двох змінних;

ezcontour(f) відображує лінії рівня функції двох змінних.

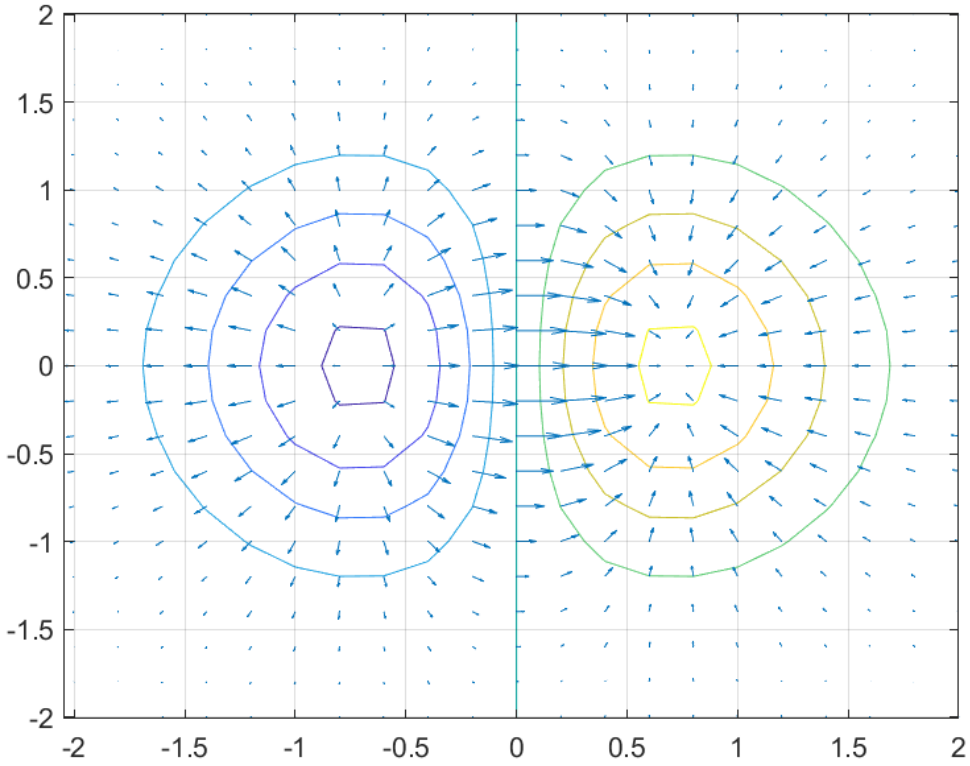
Побудова ліній рівня та векторного поля градієнта

Функція *gradient*(z) чисельно розраховує градієнт;

Функція *quiver*(*X*,*Y*,*DX*,*DY*) формує та виводить на екран поле градієнтів функції у вигляді стрілок для кожної пари елементів масивів *X* та *Y*, а пари елементів *DX* та *DY* використовуються для напрямку і розміру стрілки.

Розглянемо приклад для функції $z = xe^{-x^2-y^2}$

```
x = -2:0.2:2;
[x,y] = meshgrid(x);
z = x.*exp(-x.^2 -y.^2);
[px,py] = gradient(z);
contour(x,y,z), hold on, quiver(x,y,px,py), grid on,
hold off
```



Математичний зміст градієнту складається в задані в кожній точці (x,y) напрямку на площині, в якому функція z зростає найбільш швидше.

Література

1. The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098. Optimization Toolbox User's Guide. © COPYRIGHT 1990-2006 by The Math Works, Inc.

2. The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098. Genetic Algorithm and Direct Search Toolbox User's Guide. © COPYRIGHT 2004-2006 by The Math Works, Inc.

3. Biggs M. C. Constrained Minimization Using Recursive Quadratic Programming // Towards Global Optimization (L.C.W. Dixon and G.P. Szergo, eds.). - North- Holland. 1975. - pp. 341-349.

4. Brayton R.K., Director S.W., Hachtel G.D., and Vidigal L. A New Algorithm for Statistical Circuit Design Based on Quasi-Newton Methods and Function Splitting // IEEE Transactions on Circuits and Systems. - 1979. Vol. Sept. CAS-26. - pp. 784-794.

5. Broyden C.G. The Convergence of a Class of Double-rank Minimization Algorithms // J. Inst. Maths. Applics.-1970.- Vol. 6. pp. 76-90.

6. Censor Y. Pareto Optimality in Multiobjective Problems // Appl. Math. Optimiz. - 1977. Vol. 4. pp 41-59.

7. Da Cunha N.O. and Polak E. Constrained Minimization Under Vector-valued Criteria in Finite Dimensional Spaces // J. Math. Anal. Appl.-1967. - Vol. 19, - pp 103-124.

8. Davidon W. C. Variable Metric Method for Minimization // A.E.C. Research and Development Report. - 1959.

9. Fleming P. J. Application of Multiobjective Optimization to Compensator Design for SISO Control Systems // Electronics Letters. - 1986. Vol. 22 No. 5. - pp. 258-59.

10. Fleming P. J. Computer-Aided Control System Design of Regulators using a Multiobjective Optimization Approach // Proc. IFAC Control Applications of Nonlinear Prog. and Optim.-1985. Capri, Italy. - pp. 47-52.

11. Fletcher R. A. New Approach to Variable Metric Algorithms // Computer Journal. -- 1970. Vol. 13. pp. 317-322.

Зміст