國立中興大學電機工程學系

最佳化演算法

Midterm Exam

授課老師: 陳正倫

姓名:蔡政桀

學號:7113064722

日期:2025 年 04 月 30 日

1. Golden Section Search

Consider the function

$$f(x) = x^4 - 10x^3 + 40x^2 - 50x$$

Modify the codes of **golden section search** in the lecture notes to find the value of x that minimizes f over the range of [0, 2]. Locate this value of x to within the range of 0.005.

本題需要使用黃金分割法找出介於 0 到 2 之間的 x，使 $f(x)$ 為最小值，計算流程為:

- 設定搜尋區間 [L, R]，以及誤差範圍。
- 逐步計算兩個內點 x_1 和 x_2，並比較函數值，選擇較小函數值所對應的區間來更新。
- 不斷縮小區間直至誤差滿足停止條件，最終得到最佳解。

最後再將結果可視化，程式如 Code1，執行結果如圖 1，圖 2。

L	R	x1	x2	fx1	fx2	Error
0	2	0.764	1.236	-18.971	-17.241	2
0	1.236	0.472	0.764	-15.691	-18.971	1.236
0.472	1.236	0.764	0.94415	-18.971	-19.172	0.764
0.764	1.236	0.94415	1.0557	-19.172	-18.729	0.472
0.764	1.0557	0.87554	0.94415	-19.238	-19.172	0.2917
0.764	0.94415	0.83261	0.87554	-19.192	-19.238	0.18015
0.83261	0.94415	0.87554	0.90154	-19.238	-19.233	0.11154
0.83261	0.90154	0.85861	0.87554	-19.228	-19.238	0.068934
0.85861	0.90154	0.87554	0.88514	-19.238	-19.239	0.042936
0.87554	0.90154	0.88514	0.89161	-19.239	-19.238	0.025998
0.87554	0.89161	0.88201	0.88514	-19.239	-19.239	0.016067
0.87554	0.88514	0.87867	0.88201	-19.239	-19.239	0.0095966
0.87867	0.88514	0.88201	0.88267	-19.239	-19.239	0.0064702
0.88201	0.88514	0.88267	0.88395	-19.239	-19.239	0.0031264

```
Optimal value of x = 0.882980
Optimal value of f(x) = -19.239179
```

圖 1 問題 1 的求解過程，在第 14 代的誤差開始小於 0.005，x=0.88298，$f(x)$=-19.239179

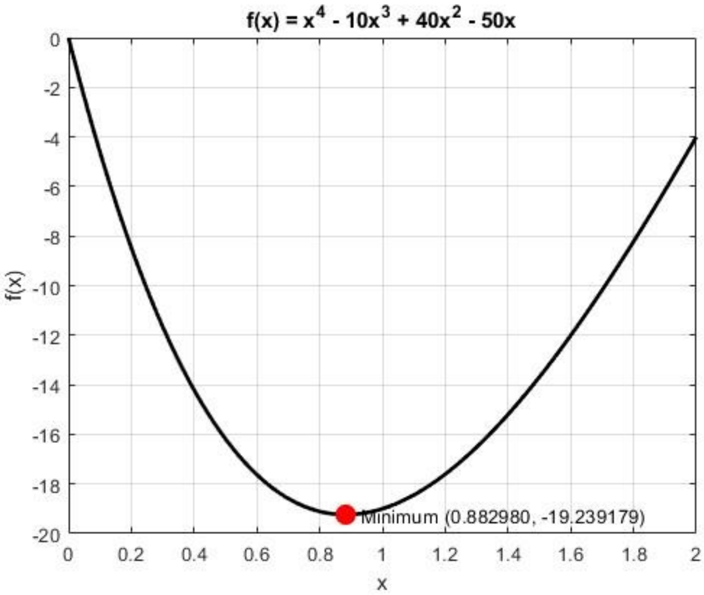

圖 2 問題 1 的方程式與解(紅點)

2. Newton’s method

Modify the codes of **Newton’s method** in the lecture notes to find the intersection of $y_1 = (x-1)^2 - 1$ and $y_2 = \cos(2x)$ in the interval $0 \le x \le 3$.

本題需要用牛頓法來求解y_1、y_2的交點，計算流程為:

- 根據初始猜測值，計算函數的值及其一階導數。
- 使用牛頓公式更新解，直到收斂或達到最大迭代次數。
- 對於每次迭代，檢查是否達到收斂條件。

Code2 為本題程式，執行過程如圖 3，圖 4 將兩方程式繪製後標註交點。

```
Newton's Method starting with x0 = 0.5000
Iteration        x_n          f(x_n)           Error
----------------------------------------------------------
    0        0.50000000  1.29030231  1.88932935e+00
    1        2.38932935  -0.86401482 1.10339848e+00
    2        1.28593087  0.07619739  4.61614174e-02
    3        1.33209228  0.00152588  9.63831611e-04
    4        1.33305611  0.00000072  4.56348513e-07
    5        1.33305657  0.00000000  1.02362563e-13
----------------------------------------------------------
Converged to root x = 1.3330565711 after 6 iterations
Function value at root f(x) = 1.1102230246e-16
First intersection point: x = 1.33305657, y = -0.88907332
>>
```

圖 3 問題 2 的解題過程

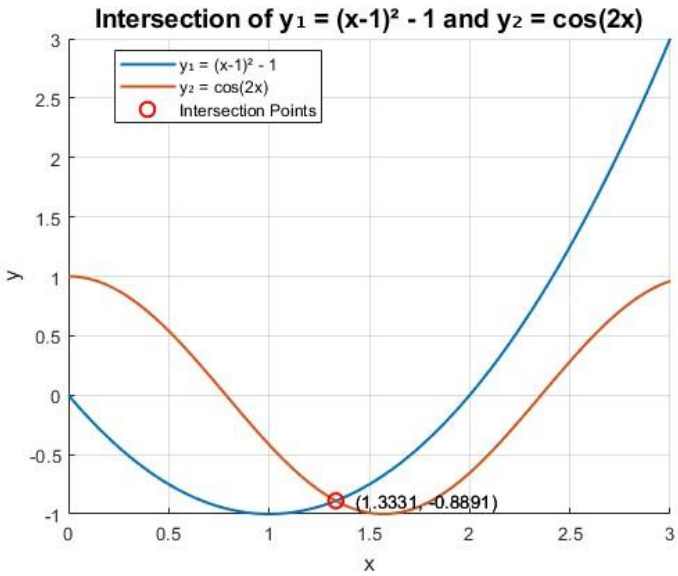

圖 4 問題 2 的解

3. Conjugate Gradient Method

本題要使用共軛梯度法求解最小化問題，計算流程為:

- 設定初始點與目標函數，並計算梯度與黑塞矩陣。
- 利用梯度下降法的思想進行搜索，但每次更新方向是對前一方向的共軛方向進行調整。
- 反覆進行直至滿足誤差容忍度或達到最大迭代次數。

Code3 為本題程式，求解過程如圖 5，初始值為(1,1)，將方程式與解繪圖

後如圖 6、圖 7。

```
Conjugate Gradient Method for minimizing f(x1, x2) = 4x1^2 + x1x2 + 3x2^2 + 2x1 + x2 + 1
-----------------------------------------------------------------------
Iter        x1            x2            f(x)        ||∇f||
-----------------------------------------------------------------------
  0    1.000000     1.000000    12.000000    13.601471
  1   -0.331806     0.031414     0.800720     1.059277
  2   -0.234043    -0.127660     0.702128     0.000000
-----------------------------------------------------------------------
Converged to optimal solution!
Optimal solution: x* = [-0.234043, -0.127660]
Optimal value: f(x*) = 0.702128
Gradient norm at solution: ||∇f(x*)|| = 4.965068e-16

Verification with analytical solution:
Analytical solution: x* = [-0.234043, -0.127660]
Analytical optimal value: f(x*) = 0.702128
```

圖 5 問題 3 求解過程

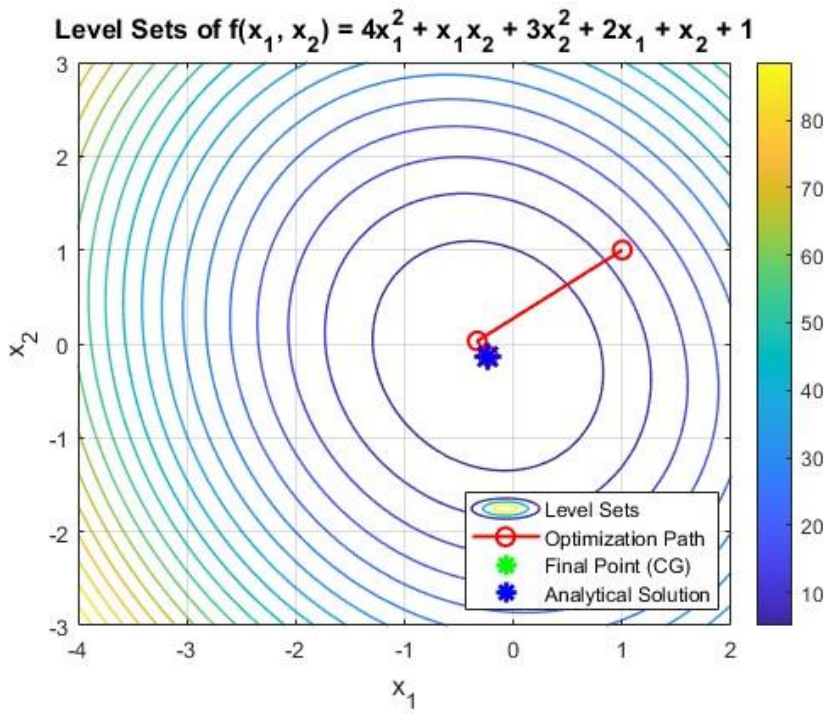

圖 6 問題 3 求解過程可視化(2D)

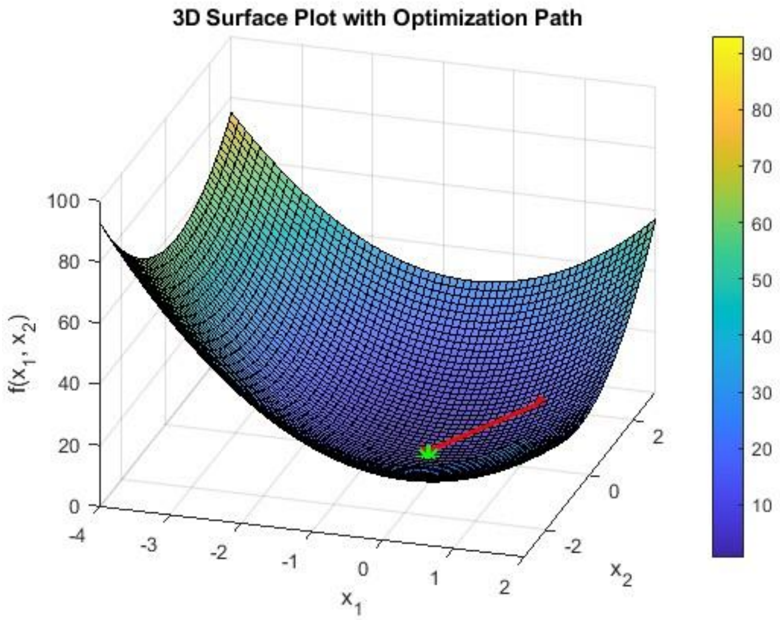

圖 7 問題 3 求解過程可視化(3D)

4. Levenberg-Marquardt Algorithm

Use the following code to generate data pairs (t_i, y_i):

t=0:0.25:10; y=3*sin(2t+0.5)+rand(1,21);

We wish to fit the data pairs with a sinusoid $y = A\sin(\omega t + \phi)$ with proper choices of A, ω, and ϕ. Modify the codes of **Levenberg-Marquardt algorithm** in the lecture notes to solve the nonlinear least square problem.

本題使用 MATLAB 內建的 lsqcurvefit 函數求解，並且使用 optimoptions 來定義求解過程，在測試過程中發現初始猜測值需要定在接近理論值才能擬合，圖 8、圖 9、圖 10 為初始值設為[3.1,2.1,0.3]的計算過程與結果可視化，擬合結果為[3.0691,2.0107,0.5009]，圖 11、圖 12、圖 13 為初始值設為[1,1,1]的計算過程與結果可視化，擬合效果較差，程式如 Code4。

```
                                                    Norm of      First-order
 Iteration  Func-count            Resnorm              step       optimality
     0           4            39.7143                                    349
     1           8            17.6514           0.33444                 16.2
     2          12            16.6608          0.211061                 1.38
     3          16            16.6605        0.00104661              0.00371
     4          20            16.6605       3.02594e-06             3.09e-06

Local minimum possible.

lsqcurvefit 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>

Fitting Results:
Amplitude (A): 3.0691
Angular Frequency (ω): 2.0107
Phase (φ): 0.5009
Final residual norm: 16.660482
Number of iterations: 4
Number of function evaluations: 20
Exitflag: 3

Theoretical function: y = 3*sin(2*t + 0.5) + noise
Fitted function:     y = 3.0691*sin(2.0107*t + 0.5009)

Goodness of Fit Metrics:
R-squared: 0.9134
RMSE: 0.6375
SSE: 16.6605
```

圖 8 問題4求解過程

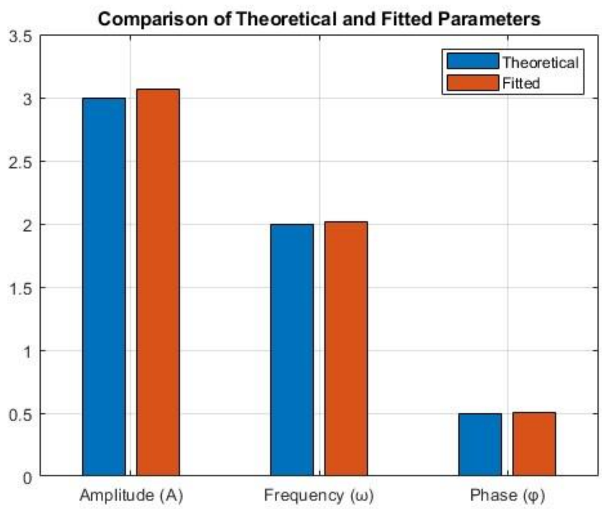

圖 9 理論值與擬合值

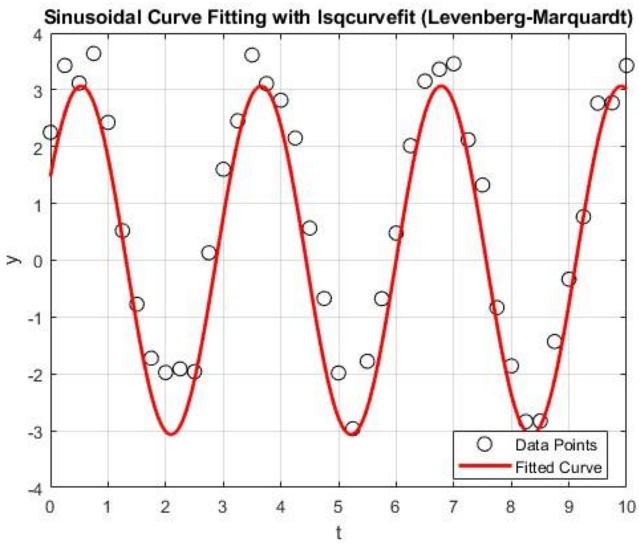

圖 10 加入雜訊的資料與計算出的擬合方程式

```
Fitting Results:
Amplitude (A): 0.7543
Angular Frequency (ω): 1.1805
Phase (φ): 1.5878
Final residual norm: 204.516995
Number of iterations: 33
Number of function evaluations: 136
Exitflag: 3

Theoretical function: y = 3*sin(2*t + 0.5) + noise
Fitted function:     y = 0.7543*sin(1.1805*t + 1.5878)

Goodness of Fit Metrics:
R-squared: -0.0626
RMSE: 2.2334
SSE: 204.5170
```

圖 11 問題 4 求解過程(2)

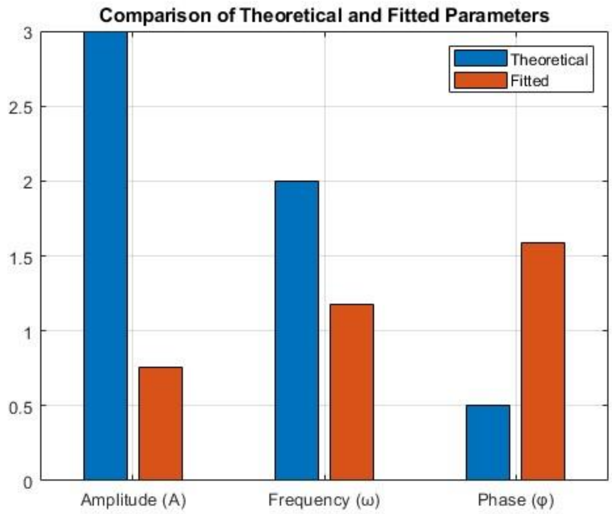

圖 12 理論值與擬合值(2)

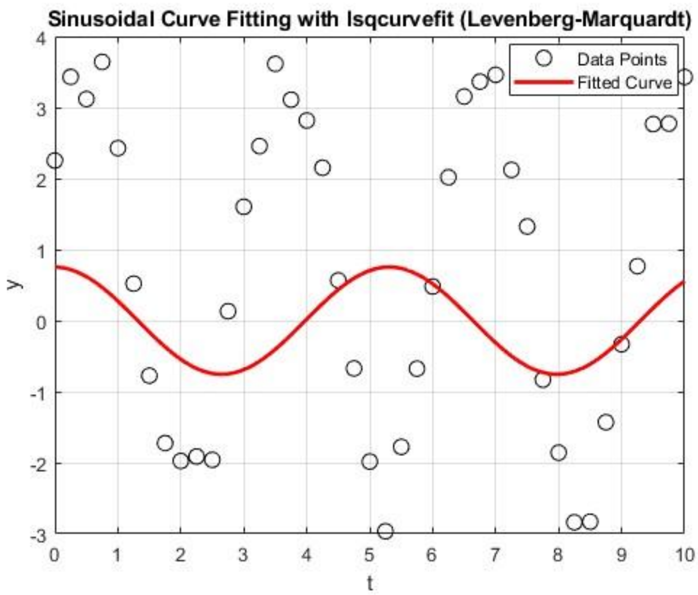

圖 13 加入雜訊的資料與計算出的擬合方程式(2)

5. Solver-Based Approach

Consider the following constrained optimization problem:

$$\text{Minimize} \quad e^{x_1}(4x_1^2 + 4x_1x_2 + 2x_2^2 + 2x_2 + 1)$$

$$\text{Subject to} \quad x_1x_2 - x_1 - x_2 \le -1.5$$

$$-x_1x_2 - 10 \le 0$$

Using solver-based approach in MATLAB optimization toolbox and select **fmincon** slover, write code to find the minimizer of the problem. Supply the solver with analytic derivative/gradient information of both objective and constraint functions to improve the accuracy and efficiency of the results.

本題需要使用 Solver-Based Approach，在 matlab 中利用 optimoptions 來設定解題過程的條件，使用 fmincon 函示來解決最小化問題，程式如 Code5，解題過程如圖 14，可視化結果如圖 15。

```
                                            First-order      Norm of
 Iter F-count            f(x)  Feasibility   optimality         step
    0       1    2.718282e+00    1.500e+00    2.727e+00
    1       5    2.603906e+00    1.205e+00    5.546e+00    4.987e-01
    2       8    2.026452e+00    2.955e-01    1.128e+00    1.383e+00
    3       9    2.818428e+00    0.000e+00    3.858e-01    3.028e-01
    4      11    2.684263e+00    0.000e+00    7.156e-01    2.996e-02
    5      12    2.644445e+00    0.000e+00    3.677e-02    2.140e-02
    6      13    2.559392e+00    0.000e+00    7.950e-03    1.646e-02
    7      14    2.554441e+00    0.000e+00    3.052e-04    1.451e-03
    8      15    2.554227e+00    0.000e+00    1.937e-06    6.606e-05

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
最佳解: x1 = 1.183388, x2 = -1.726458
目標函數值: 2.554227
約束條件 1: -0.000001 (應該 <= 0)
約束條件 2: -7.956930 (應該 <= 0)
```

圖 14 問題 5 執行過程

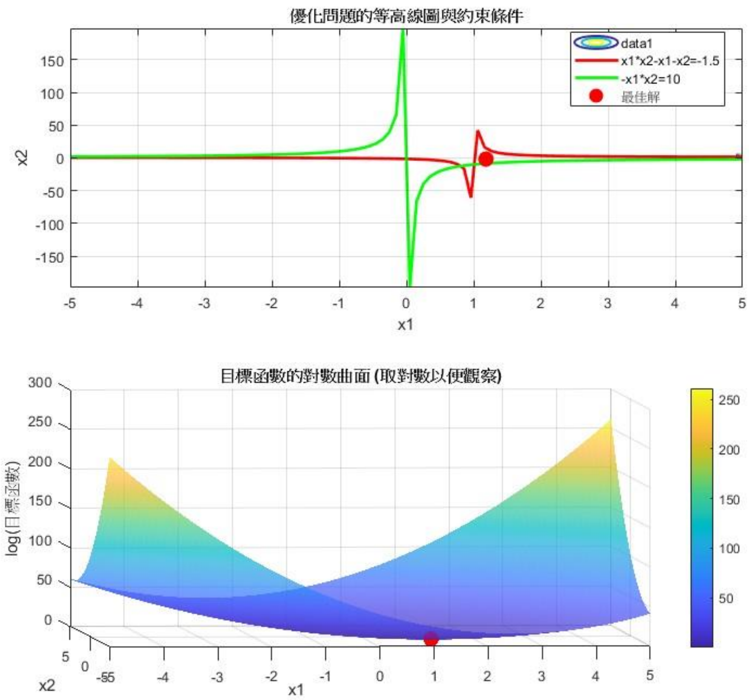

圖 15 問題 5 執行結果

6. Problem-Based & Solver-Based Approach

Consider the following linear program:

$$\text{Maximize} \quad x_1 + 2x_2$$

$$\text{Subject to} \quad \begin{aligned} &-2x_1 + x_2 + x_3 = 2 \\ &-x_1 + 2x_2 + x_4 = 7 \\ &x_1 + x_5 = 3 \\ &x_i \geq 0,\ i = 1,2,3,4,5 \end{aligned}$$

Using both problem-based and solver-based approaches in MATLAB optimization toolbox, wirite code to find the minimizer of the problem.

本題需要分別以兩種方式解題，差別在於 Problem-Based 可以使用 optimvar 和 optimproblem 定義優化變數和問題，Solver-Based Approach 則會用到到像是 fminunc, fmincon, lsqnonlin 等求解器，Code6 為本題程式，圖 16、圖 17 為兩個方法的執行過程與結果。

```
線性規劃問題求解
目標: 最大化 x1 + 2x2
約束條件:
  -2x1 + x2 + x3 = 2
  -x1 + 2x2 + x4 = 7
  x1 + x5 = 3
  xi ≥ 0, i = 1,2,3,4,5

======= 問題導向解法 =======

Solving problem using linprog.

Optimal solution found.

最優解:
x1 = 3.0000
x2 = 5.0000
x3 = 3.0000
x4 = 0.0000
x5 = 0.0000
目標函數值 = 13.0000

檢查約束條件:
約束1: -2x1 + x2 + x3 = 2.0000 (應該等於2)
約束2: -x1 + 2x2 + x4 = 7.0000 (應該等於7)
約束3: x1 + x5 = 3.0000 (應該等於3)
非負約束: 所有變數都非負
```

圖 16 問題導向法求解過程與結果

```
======= 求解器導向解法 =======
Running HiGHS 1.7.0: Copyright (c) 2024 HiGHS under MIT licence terms
Coefficient ranges:
  Matrix [1e+00, 2e+00]
  Cost   [1e+00, 2e+00]
  Bound  [0e+00, 0e+00]
  RHS    [2e+00, 7e+00]
Presolving model
2 rows, 4 cols, 6 nonzeros  0s
0 rows, 0 cols, 0 nonzeros  0s
Presolve : Reductions: rows 0(-3); columns 0(-5); elements 0(-8) - Reduced to empty
Solving the original LP from the solution after postsolve
Model   status      : Optimal
Objective value     : -1.3000000000e+01
HiGHS run time      :           0.00

Optimal solution found.

最優解:
x1 = 3.0000
x2 = 5.0000
x3 = 3.0000
x4 = 0.0000
x5 = 0.0000
目標函數值 = 13.0000

檢查約束條件:
約束1: -2x1 + x2 + x3 = 2.0000 (應該等於2)
約束2: -x1 + 2x2 + x4 = 7.0000 (應該等於7)
約束3: x1 + x5 = 3.0000 (應該等於3)
非負約束: 所有變數都非負
```

圖 17 求解器法求解過程與結果

Code

1. Code1

```
clear all, clc
format short

% Define the Objective function
f = @(x) x.^4 - 10*x.^3 + 40*x.^2 - 50*x;

% Set parameters
L = 0;          % Lower limit of the search range
R = 2;          % Upper limit of the search range
maxerr = 0.005; % Maximum error (stopping criteria)
maxiter = 100;  % Maximum number of iterations (safety parameter)

% Plot the function
t = linspace(L, R, 100);
plot(t, f(t), 'k', 'LineWidth', 2);
title('f(x) = x^4 - 10x^3 + 40x^2 - 50x');
xlabel('x');
ylabel('f(x)');
grid on;

% Golden section search
ratio = 0.618;  % Golden ratio
x2 = L + ratio * (R - L); % Compute x2
x1 = L + R - x2;         % Compute x1
err = R - L;             % Initial Error
iter = 1;                % Set iteration counter initially

fprintf('Iteration\t   L\t\t   R\t\t   x1\t\t   x2\t\t  f(x1)\t\t  f(x2)\t\t
Error\n');
fprintf('------------------------------------------------------------------
------------------------------------------\n');

% Create a storage for results
rsl = [];
```

```
while err > maxerr
    % Compute Error
    err = R - L;

    % Compute function values
    fx1 = f(x1);
    fx2 = f(x2);

    % Display current iteration details
    fprintf('%4d\t\t%8.6f\t%8.6f\t%8.6f\t%8.6f\t%8.6f\t%8.6f\t%8.6f\n', ...
        iter, L, R, x1, x2, fx1, fx2, err);

    % Store results for this iteration
    rsl(iter,:) = [L, R, x1, x2, fx1, fx2, err];

    % Update interval based on comparison of function values
    if fx1 > fx2  % Look for "Minimum"
        L = x1;   % Update L
        x1 = x2;  % Update x1
        x2 = L + ratio * (R - L); % Compute new x2
    elseif fx1 < fx2
        R = x2;   % Update R
        x2 = x1;  % Update x2
        x1 = L + R - x2; % Compute new x1
    elseif fx1 == fx2
        if min(abs(x1), abs(L)) == abs(L)
            R = x2; % Update R
        else
            L = x1; % Update L
        end
        x1 = L + (1 - ratio) * (R - L);
        x2 = L + ratio * (R - L);
    end

    % Check if maximum iterations reached
    if iter == maxiter
        fprintf('Maximum number of iterations (%d) reached.\n', maxiter);
```

```
        break;
    else
        iter = iter + 1; % Update iteration counter
    end
end

% Display the termination condition
if iter < maxiter
    fprintf('Maximum error limit %.6f reached after %d iterations.\n', 
maxerr, iter);
end

% Display results as a table
Variables = {'L', 'R', 'x1', 'x2', 'fx1', 'fx2', 'Error'};
ResultTable = array2table(rsl);
ResultTable.Properties.VariableNames(1:size(ResultTable, 2)) = Variables;
disp(ResultTable);

% Compute & Print Optimal Result
xopt = (L + R) / 2;      % Optimal "x" (mid-point of final L & R)
fopt = f(xopt);          % Optimal value of f(x)
fprintf('\nOptimal value of x = %.6f\n', xopt);
fprintf('Optimal value of f(x) = %.6f\n', fopt);

% Mark the minimum point on the plot
hold on;
plot(xopt, fopt, 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');
text(xopt + 0.05, fopt, ['Minimum (' num2str(xopt, '%.6f') ', ' 
num2str(fopt, '%.6f') ')']);
hold off;
```

2. Code2

```
clear all, clc

% Define the functions and their derivatives
y1 = @(x) (x-1).^2 - 1;     % y₁ = (x-1)² - 1
dy1 = @(x) 2.*(x-1);        % Derivative of y₁

y2 = @(x) cos(2*x);         % y₂ = cos(2x)
```

```
dy2 = @(x) -2*sin(2*x);     % Derivative of y₂

% Set the interval for search
x = 0:0.01:3;

% Plot both functions to visualize the intersection points
figure;
hold on;
plot(x, y1(x), 'LineWidth', 1.5);
plot(x, y2(x), 'LineWidth', 1.5);
hold off;
grid on;
title('Functions y₁ = (x-1)² - 1 and y₂ = cos(2x)', 'FontSize', 14);
xlabel('x', 'FontSize', 12);
ylabel('y', 'FontSize', 12);
legend('y₁ = (x-1)² - 1', 'y₂ = cos(2x)', 'Location', 'best');

% Visual inspection shows there are two intersection points in [0,3]
% Let's find both using Newton's method with different initial guesses

% Find the intersection point
initial_guess1 = 0.5;  % Initial guess based on visual inspection
intersection1 = newtons_method(@(x) y2(x) - y1(x), @(x) dy2(x) - dy1(x),
initial_guess1);

% Display results
fprintf('First intersection point: x = %.8f, y = %.8f\n', intersection1,
y1(intersection1));

% Plot the result with intersection points marked
figure;
hold on;
plot(x, y1(x), 'LineWidth', 1.5);
plot(x, y2(x), 'LineWidth', 1.5);
plot(intersection1, y1(intersection1), 'ro', 'MarkerSize', 8, 'LineWidth',
1.5);
text(intersection1 + 0.1, y1(intersection1), ['(' num2str(intersection1,
'%.4f') ', ' num2str(y1(intersection1), '%.4f') ')']);
```

```
hold off;
grid on;
title('Intersection of y₁ = (x-1)² - 1 and y₂ = cos(2x)', 'FontSize', 14);
xlabel('x', 'FontSize', 12);
ylabel('y', 'FontSize', 12);
legend('y₁ = (x-1)² - 1', 'y₂ = cos(2x)', 'Intersection Points',
'Location', 'best');

% Newton's Method Implementation
function root = newtons_method(f, df, x0)
    % Set parameters
    TOL = 1e-10;       % Tolerance for convergence
    max_iter = 100;    % Maximum number of iterations

    % Initialize
    x_old = x0;
    err = 2*TOL;       % Set initial error to enter the loop
    iter = 0;          % Iteration counter

    % Display header for iteration progress
    fprintf('\nNewton''s Method starting with x0 = %.4f\n', x0);
    fprintf('Iteration\t   x_n\t\t   f(x_n)\t\t   Error\n');
    fprintf('----------------------------------------------------------
\n');

    % Newton's Method iteration
    while (err > TOL) && (iter < max_iter)
        % Compute function value and derivative at current point
        f_val = f(x_old);
        df_val = df(x_old);

        % Check if derivative is close to zero to avoid division by zero
        if abs(df_val) < 1e-10
            warning('Derivative is close to zero. Method may not
converge.');
            df_val = sign(df_val) * 1e-10;  % Assign a small non-zero value
        end
```

```
        % Update estimate of root using Newton's formula
        x_new = x_old - f_val / df_val;

        % Calculate error
        err = abs(x_new - x_old);

        % Display current iteration information
        fprintf('%5d\t\t%10.8f\t%10.8f\t%10.8e\n', iter, x_old, f_val, err);

        % Update for next iteration
        x_old = x_new;
        iter = iter + 1;
    end

    % Check if maximum iterations reached
    if iter >= max_iter
        warning('Maximum number of iterations reached. Solution may not be
accurate.');
    end

    % Display final result
    fprintf('----------------------------------------------------------
\n');
    fprintf('Converged to root x = %.10f after %d iterations\n', x_old,
iter);
    fprintf('Function value at root f(x) = %.10e\n', f(x_old));

    % Return the root
    root = x_old;
end
```

3. Code3

```
format short;
clc; clear all;
syms x1 x2

% Define Objective function
f1 = 4*x1^2 + x1*x2 + 3*x2^2 + 2*x1 + x2 + 1;
fx = matlabFunction(f1, 'Vars', [x1, x2]); % Convert to function
```

```
fobj = @(x) fx(x(1), x(2));

% Plot the level set of the objective function
figure(1);
X = -4:0.1:2;
Y = -3:0.1:3;
[X1, X2] = meshgrid(X, Y);
Z = 4*X1.^2 + X1.*X2 + 3*X2.^2 + 2*X1 + X2 + 1;

% Plot contour
contour(X, Y, Z, 20, 'LineWidth', 1);
xlabel('x_1', 'FontSize', 12);
ylabel('x_2', 'FontSize', 12);
title('Level Sets of f(x_1, x_2) = 4x_1^2 + x_1x_2 + 3x_2^2 + 2x_1 + x_2 +
1', 'FontSize', 12);
colorbar;
grid on;
hold on;

% Compute the gradient of f
gradient_f = gradient(f1, [x1, x2]);
% Convert symbolic gradient to function handle
gradient_x1 = matlabFunction(gradient_f(1), 'Vars', [x1, x2]);
gradient_x2 = matlabFunction(gradient_f(2), 'Vars', [x1, x2]);
% Function to compute gradient at point x
gradf = @(x) [gradient_x1(x(1), x(2)); gradient_x2(x(1), x(2))];

% Compute the Hessian matrix (constant for quadratic function)
H = double(hessian(f1, [x1, x2]));

% Parameters for conjugate gradient method
x0 = [1; 1];     % Initial point (column vector)
maxiter = 10;    % Maximum number of iterations
tol = 1e-6;      % Tolerance for convergence
iter = 1;        % Iteration counter
X_history = x0'; % Save iteration history (as row for plotting)
f_history = fobj(x0); % Save function values
```

```
% Display header
fprintf('Conjugate Gradient Method for minimizing f(x₁, x₂) = 4x₁² + x₁x₂ + 3x₂² + 2x₁ + x₂ + 1\n');
fprintf('----------------------------------------------------------------------\n');
fprintf('Iter\t   x₁\t\t   x₂\t\t   f(x)\t\t ||∇f||\n');
fprintf('----------------------------------------------------------------------\n');
fprintf('%3d\t%10.6f\t%10.6f\t%10.6f\t%10.6f\n', 0, x0(1), x0(2), fobj(x0), norm(gradf(x0)));

% Initial gradient and search direction
g0 = gradf(x0);
d0 = -g0;  % Initial search direction is negative gradient

% Main loop of conjugate gradient method
while (norm(g0) > tol) && (iter <= maxiter)
    % Compute step size (exact line search for quadratic functions)
    alpha = -(g0' * d0) / (d0' * H * d0);

    % Update position
    x_new = x0 + alpha * d0;

    % Compute new gradient
    g_new = gradf(x_new);

    % Compute beta using Fletcher-Reeves formula
    beta = (g_new' * g_new) / (g0' * g0);

    % Update search direction
    d_new = -g_new + beta * d0;

    % Update current point and gradient for next iteration
    x0 = x_new;
    g0 = g_new;
    d0 = d_new;

    % Save history
```

```
    X_history = [X_history; x0'];
    f_history = [f_history; fobj(x0)];

    % Display iteration information
    fprintf('%3d\t%10.6f\t%10.6f\t%10.6f\t%10.6f\n', iter, x0(1), x0(2),
fobj(x0), norm(g0));

    % Update iteration counter
    iter = iter + 1;
end

% Display result
fprintf('-----------------------------------------------------------------
-----\n');
if norm(g0) <= tol
    fprintf('Converged to optimal solution!\n');
else
    fprintf('Maximum iterations reached.\n');
end
fprintf('Optimal solution: x* = [%f, %f]\n', x0(1), x0(2));
fprintf('Optimal value: f(x*) = %f\n', fobj(x0));
fprintf('Gradient norm at solution: ||∇f(x*)|| = %e\n', norm(gradf(x0)));

% Calculate analytical solution for verification
% For quadratic functions, we can compute exact solution by setting
gradient to zero
fprintf('\nVerification with analytical solution:\n');
% Solve 8*x1 + x2 + 2 = 0 and x1 + 6*x2 + 1 = 0
A = [8, 1; 1, 6];
b = [-2; -1];
x_analytical = A\b;
fprintf('Analytical solution: x* = [%f, %f]\n', x_analytical(1),
x_analytical(2));
fprintf('Analytical optimal value: f(x*) = %f\n', fobj(x_analytical));

% Plot the optimization path
plot(X_history(:,1), X_history(:,2), 'ro-', 'LineWidth', 1.5, 'MarkerSize',
8);
```

```
plot(x0(1), x0(2), 'g*', 'LineWidth', 2, 'MarkerSize', 12);
plot(x_analytical(1), x_analytical(2), 'b*', 'LineWidth', 2, 'MarkerSize',
12);
legend('Level Sets', 'Optimization Path', 'Final Point (CG)', 'Analytical
Solution', 'Location', 'best');

% 3D surface plot
figure(2);
surf(X1, X2, Z, 'FaceAlpha', 0.8);
hold on;
for i = 1:size(X_history, 1)
    plot3(X_history(i,1), X_history(i,2), fobj([X_history(i,1);
X_history(i,2)]), 'r.', 'MarkerSize', 20);
    if i < size(X_history, 1)
        plot3([X_history(i,1), X_history(i+1,1)], [X_history(i,2),
X_history(i+1,2)], ...
              [fobj([X_history(i,1); X_history(i,2)]),
fobj([X_history(i+1,1); X_history(i+1,2)])], 'r-', 'LineWidth', 2);
    end
end
plot3(x0(1), x0(2), fobj(x0), 'g*', 'LineWidth', 2, 'MarkerSize', 12);
xlabel('x_1');
ylabel('x_2');
zlabel('f(x_1, x_2)');
title('3D Surface Plot with Optimization Path');
colorbar;
grid on;
view(40, 30);
```

4. Code4

```
format short;
clc; clear all;
syms x1 x2

% Define Objective function
f1 = 4*x1^2 + x1*x2 + 3*x2^2 + 2*x1 + x2 + 1;
fx = matlabFunction(f1, 'Vars', [x1, x2]); % Convert to function
fobj = @(x) fx(x(1), x(2));
```

```matlab
% Plot the level set of the objective function
figure(1);
X = -4:0.1:2;
Y = -3:0.1:3;
[X1, X2] = meshgrid(X, Y);
Z = 4*X1.^2 + X1.*X2 + 3*X2.^2 + 2*X1 + X2 + 1;

% Plot contour
contour(X, Y, Z, 20, 'LineWidth', 1);
xlabel('x_1', 'FontSize', 12);
ylabel('x_2', 'FontSize', 12);
title('Level Sets of f(x_1, x_2) = 4x_1^2 + x_1x_2 + 3x_2^2 + 2x_1 + x_2 +
1', 'FontSize', 12);
colorbar;
grid on;
hold on;

% Compute the gradient of f
gradient_f = gradient(f1, [x1, x2]);
% Convert symbolic gradient to function handle
gradient_x1 = matlabFunction(gradient_f(1), 'Vars', [x1, x2]);
gradient_x2 = matlabFunction(gradient_f(2), 'Vars', [x1, x2]);
% Function to compute gradient at point x
gradf = @(x) [gradient_x1(x(1), x(2)); gradient_x2(x(1), x(2))];

% Compute the Hessian matrix (constant for quadratic function)
H = double(hessian(f1, [x1, x2]));

% Parameters for conjugate gradient method
x0 = [1; 1];     % Initial point (column vector)
maxiter = 10;    % Maximum number of iterations
tol = 1e-6;      % Tolerance for convergence
iter = 1;        % Iteration counter
X_history = x0'; % Save iteration history (as row for plotting)
f_history = fobj(x0); % Save function values

% Display header
```

```
fprintf('Conjugate Gradient Method for minimizing f(x₁, x₂) = 4x₁² + x₁x₂ + 3x₂² + 2x₁ + x₂ + 1\n');
fprintf('----------------------------------------------------------------------------\n');
fprintf('Iter\t   x₁\t\t   x₂\t\t   f(x)\t\t ||∇f||\n');
fprintf('----------------------------------------------------------------------------\n');
fprintf('%3d\t%10.6f\t%10.6f\t%10.6f\t%10.6f\n', 0, x0(1), x0(2), fobj(x0), norm(gradf(x0)));

% Initial gradient and search direction
g0 = gradf(x0);
d0 = -g0;  % Initial search direction is negative gradient

% Main loop of conjugate gradient method
while (norm(g0) > tol) && (iter <= maxiter)
    % Compute step size (exact line search for quadratic functions)
    alpha = -(g0' * d0) / (d0' * H * d0);

    % Update position
    x_new = x0 + alpha * d0;

    % Compute new gradient
    g_new = gradf(x_new);

    % Compute beta using Fletcher-Reeves formula
    beta = (g_new' * g_new) / (g0' * g0);

    % Update search direction
    d_new = -g_new + beta * d0;

    % Update current point and gradient for next iteration
    x0 = x_new;
    g0 = g_new;
    d0 = d_new;

    % Save history
    X_history = [X_history; x0'];
```

```
    f_history = [f_history; fobj(x0)];

    % Display iteration information
    fprintf('%3d\t%10.6f\t%10.6f\t%10.6f\t%10.6f\n', iter, x0(1), x0(2), 
fobj(x0), norm(g0));

    % Update iteration counter
    iter = iter + 1;
end

% Display result
fprintf('------------------------------------------------------------------
-----\n');
if norm(g0) <= tol
    fprintf('Converged to optimal solution!\n');
else
    fprintf('Maximum iterations reached.\n');
end
fprintf('Optimal solution: x* = [%f, %f]\n', x0(1), x0(2));
fprintf('Optimal value: f(x*) = %f\n', fobj(x0));
fprintf('Gradient norm at solution: ||∇f(x*)|| = %e\n', norm(gradf(x0)));

% Calculate analytical solution for verification
% For quadratic functions, we can compute exact solution by setting 
gradient to zero
fprintf('\nVerification with analytical solution:\n');
% Solve 8*x1 + x2 + 2 = 0 and x1 + 6*x2 + 1 = 0
A = [8, 1; 1, 6];
b = [-2; -1];
x_analytical = A\b;
fprintf('Analytical solution: x* = [%f, %f]\n', x_analytical(1), 
x_analytical(2));
fprintf('Analytical optimal value: f(x*) = %f\n', fobj(x_analytical));

% Plot the optimization path
plot(X_history(:,1), X_history(:,2), 'ro-', 'LineWidth', 1.5, 'MarkerSize', 
8);
plot(x0(1), x0(2), 'g*', 'LineWidth', 2, 'MarkerSize', 12);
```

```
plot(x_analytical(1), x_analytical(2), 'b*', 'LineWidth', 2, 'MarkerSize',
12);
legend('Level Sets', 'Optimization Path', 'Final Point (CG)', 'Analytical
Solution', 'Location', 'best');

% 3D surface plot
figure(2);
surf(X1, X2, Z, 'FaceAlpha', 0.8);
hold on;
for i = 1:size(X_history, 1)
    plot3(X_history(i,1), X_history(i,2), fobj([X_history(i,1);
X_history(i,2)]), 'r.', 'MarkerSize', 20);
    if i < size(X_history, 1)
        plot3([X_history(i,1), X_history(i+1,1)], [X_history(i,2),
X_history(i+1,2)], ...
             [fobj([X_history(i,1); X_history(i,2)]),
fobj([X_history(i+1,1); X_history(i+1,2)])], 'r-', 'LineWidth', 2);
    end
end
plot3(x0(1), x0(2), fobj(x0), 'g*', 'LineWidth', 2, 'MarkerSize', 12);
xlabel('x_1');
ylabel('x_2');
zlabel('f(x_1, x_2)');
title('3D Surface Plot with Optimization Path');
colorbar;
grid on;
view(40, 30);
```

5. Code5

```
function main()
    % 初始點
    x0 = [0, 0];

    % 定義 fmincon 的選項
    options = optimoptions('fmincon', 'Display', 'iter', ...
                           'Algorithm', 'interior-point', ...
                           'SpecifyObjectiveGradient', true, ...
                           'SpecifyConstraintGradient', true);
```

```
    % 定義邊界（本題沒有明確邊界，設為空）
    lb = [];
    ub = [];

    % 定義線性約束（本題沒有線性約束，設為空）
    A = [];
    b = [];
    Aeq = [];
    beq = [];

    % 執行 fmincon
    [x_opt, f_opt] = fmincon(@objective_with_grad, x0, A, b, Aeq, beq, lb,
ub, @constraints_with_grad, options);

    % 顯示結果
    fprintf('最佳解: x1 = %.6f, x2 = %.6f\n', x_opt(1), x_opt(2));
    fprintf('目標函數值: %.6f\n', f_opt);

    % 檢查約束
    [c, ceq] = constraints_with_grad(x_opt);
    fprintf('約束條件 1: %.6f (應該 <= 0)\n', c(1));
    fprintf('約束條件 2: %.6f (應該 <= 0)\n', c(2));

    % 可視化最佳解
    visualize_solution(x_opt);
end

function [f, grad] = objective_with_grad(x)
    % 提取變數
    x1 = x(1);
    x2 = x(2);

    % 計算指數部分
    exponent = 4*x1^2 + 4*x1*x2 + 2*x2^2 + 2*x2 + 1;

    % 目標函數
    f = exp(exponent);
```

```
    % 目標函數的梯度
    if nargout > 1
        % 關於 x1 的偏導數
        df_dx1 = f * (8*x1 + 4*x2);

        % 關於 x2 的偏導數
        df_dx2 = f * (4*x1 + 4*x2 + 2);

        grad = [df_dx1; df_dx2];
    end
end

function [c, ceq, gradc, gradceq] = constraints_with_grad(x)
    % 提取變數
    x1 = x(1);
    x2 = x(2);

    % 不等式約束 (c <= 0)
    c = [x1*x2 - x1 - x2 + 1.5;   % 第一個約束: x1*x2 - x1 - x2 <= -1.5
         -x1*x2 - 10];            % 第二個約束: -x1*x2 - 10 <= 0

    % 無等式約束
    ceq = [];

    % 不等式約束的梯度
    if nargout > 2
        % 第一個約束的梯度: d/dx1 = x2-1, d/dx2 = x1-1
        gradc1 = [x2-1; x1-1];

        % 第二個約束的梯度: d/dx1 = -x2, d/dx2 = -x1
        gradc2 = [-x2; -x1];

        gradc = [gradc1, gradc2];
        gradceq = [];
    end
end

function visualize_solution(x_opt)
```

```
    % 建立繪圖網格
    [X1, X2] = meshgrid(linspace(-5, 5, 100), linspace(-5, 5, 100));
    Z = zeros(size(X1));

    % 計算目標函數值
    for i = 1:size(X1, 1)
        for j = 1:size(X1, 2)
            x = [X1(i,j), X2(i,j)];
            [Z(i,j), ~] = objective_with_grad(x);
        end
    end

    % 建立新的圖形
    figure;

    % 繪製等高線圖
    subplot(2,1,1);
    contour(X1, X2, Z, 20, 'LineWidth', 1.5);
    hold on;

    % 繪製約束條件
    x1_range = linspace(-5, 5, 100);
    x2_c1 = (x1_range + 1.5)./(x1_range - 1); % 從 x1*x2 - x1 - x2 = -1.5 解
出 x2
    x2_c2 = -10./x1_range; % 從 -x1*x2 = 10 解出 x2

    % 繪製約束條件線
    plot(x1_range, x2_c1, 'r-', 'LineWidth', 2, 'DisplayName', 'x1*x2-x1-
x2=-1.5');
    plot(x1_range, x2_c2, 'g-', 'LineWidth', 2, 'DisplayName', '-
x1*x2=10');

    % 標示最佳解
    plot(x_opt(1), x_opt(2), 'ro', 'MarkerSize', 10, 'MarkerFaceColor',
'r', 'DisplayName', '最佳解');

    % 添加圖例和標籤
    legend('Location', 'best');
```

```
    xlabel('x1');
    ylabel('x2');
    title('優化問題的等高線圖與約束條件');
    grid on;

    % 繪製 3D 曲面圖
    subplot(2,1,2);
    surf(X1, X2, log(Z), 'EdgeColor', 'none', 'FaceAlpha', 0.7);
    hold on;

    % 在 3D 曲面上標示最佳解
    [f_opt, ~] = objective_with_grad(x_opt);
    plot3(x_opt(1), x_opt(2), log(f_opt), 'ro', 'MarkerSize', 10,
'MarkerFaceColor', 'r');

    % 添加標籤
    xlabel('x1');
    ylabel('x2');
    zlabel('log(目標函數)');
    title('目標函數的對數曲面 (取對數以便觀察)');
    colorbar;
    grid on;

    % 調整圖形
    set(gcf, 'Position', [100, 100, 800, 700]);
end
```

6. Code6

```
function main()
    fprintf('線性規劃問題求解\n');
    fprintf('目標: 最大化 x₁ + 2x₂\n');
    fprintf('約束條件:\n');
    fprintf('  -2x₁ + x₂ + x₃ = 2\n');
    fprintf('  -x₁ + 2x₂ + x₄ = 7\n');
    fprintf('  x₁ + x₅ = 3\n');
    fprintf('  xᵢ ≥ 0, i = 1,2,3,4,5\n\n');

    % 執行問題導向解法
    fprintf('======= 問題導向解法 =======\n');
```

```matlab
    problem_based_approach();

    % 執行求解器導向解法
    fprintf('\n======= 求解器導向解法 =======\n');
    solver_based_approach();
end

% 方法一：問題導向解法
function problem_based_approach()
    % 創建優化問題
    prob = optimproblem('ObjectiveSense', 'maximize');

    % 定義變數 (所有變數非負)
    x = optimvar('x', 5, 'LowerBound', 0);

    % 定義目標函數 (最大化 x₁ + 2x₂)
    prob.Objective = x(1) + 2*x(2);

    % 定義約束條件
    prob.Constraints.con1 = -2*x(1) + x(2) + x(3) == 2;
    prob.Constraints.con2 = -x(1) + 2*x(2) + x(4) == 7;
    prob.Constraints.con3 = x(1) + x(5) == 3;

    % 求解問題
    x0.x = zeros(5, 1); % 初始點
    [sol, fval, exitflag, output] = solve(prob, x0);

    % 顯示結果
    fprintf('最優解:\n');
    for i = 1:5
        fprintf('x%d = %.4f\n', i, sol.x(i));
    end
    fprintf('目標函數值 = %.4f\n', fval);

    % 檢查約束條件是否滿足
    check_constraints(sol.x);
end
```

```
% 方法二：求解器導向解法
function solver_based_approach()
    % 定義目標函數的係數 (最大化 x₁ + 2x₂)
    % 注意：linprog 是最小化問題，所以我們對係數取負值使其成為最大化
    f = [-1; -2; 0; 0; 0];

    % 定義不等式約束 Ax <= b (這裡沒有不等式約束，除了非負約束)
    A = [];
    b = [];

    % 定義等式約束 Aeq*x = beq
    Aeq = [-2, 1, 1, 0, 0;
           -1, 2, 0, 1, 0;
            1, 0, 0, 0, 1];
    beq = [2; 7; 3];

    % 定義變數下限 (所有變數非負)
    lb = zeros(5, 1);

    % 定義變數上限 (無上限)
    ub = [];

    % 定義選項
    options = optimoptions('linprog', 'Display', 'iter');

    % 求解線性規劃問題
    [x, fval, exitflag, output] = linprog(f, A, b, Aeq, beq, lb, ub,
options);

    % 顯示結果
    fprintf('最優解:\n');
    for i = 1:5
        fprintf('x%d = %.4f\n', i, x(i));
    end
    fprintf('目標函數值 = %.4f\n', -fval); % 轉回最大化問題的值

    % 檢查約束條件是否滿足
    check_constraints(x);
```

```matlab
end

% 檢查約束條件是否滿足
function check_constraints(x)
    fprintf('\n 檢查約束條件:\n');

    % 檢查-2x₁ + x₂ + x₃ = 2
    con1_val = -2*x(1) + x(2) + x(3);
    fprintf('約束 1: -2x₁ + x₂ + x₃ = %.4f (應該等於 2)\n', con1_val);

    % 檢查-x₁ + 2x₂ + x₄ = 7
    con2_val = -x(1) + 2*x(2) + x(4);
    fprintf('約束 2: -x₁ + 2x₂ + x₄ = %.4f (應該等於 7)\n', con2_val);

    % 檢查x₁ + x₅ = 3
    con3_val = x(1) + x(5);
    fprintf('約束 3: x₁ + x₅ = %.4f (應該等於 3)\n', con3_val);

    % 檢查非負約束
    all_non_negative = all(x >= 0);
    if all_non_negative
        fprintf('非負約束: 所有變數都非負\n');
    else
        fprintf('非負約束: 不滿足! 某些變數為負\n');
    end
end
```