國立中興大學電機工程學系

最佳化演算法

**Midterm Exam**

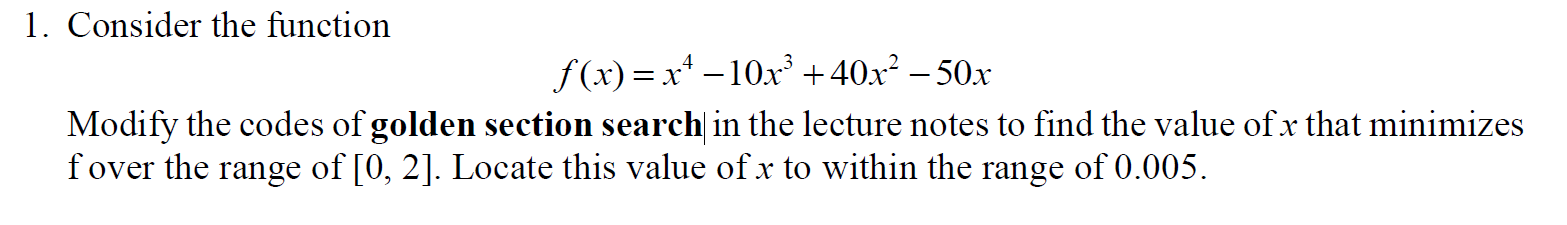
授課老師: 陳正倫

姓名:蔡政桀

學號:7113064722

日期:2025年04月30日

1. Golden Section Search



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x1​和x2，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。

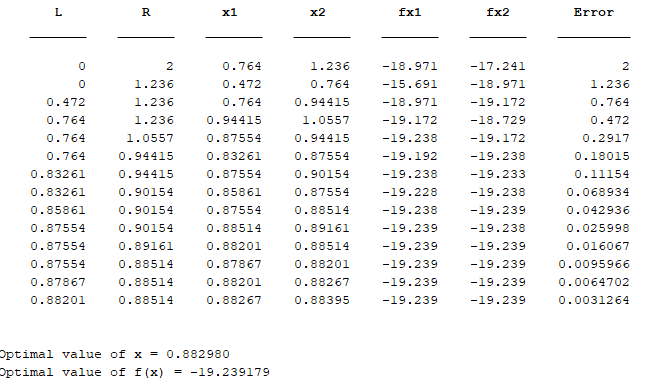


圖 1 問題1的求解過程，在第14代的誤差開始小於0.005，*x*=0.88298，*f(x)=*-19.239179

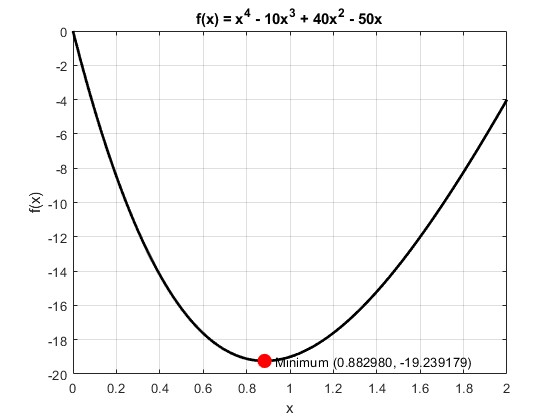
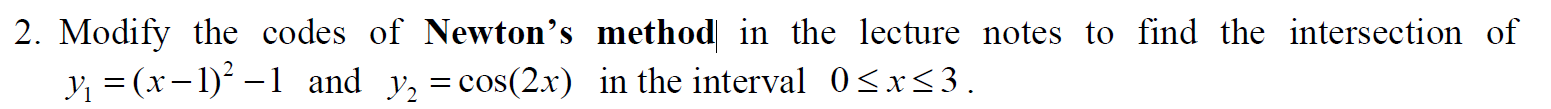


圖 2 問題1的方程式與解(紅點)

1. Newton’s method



本題需要用牛頓法來求解的交點，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。

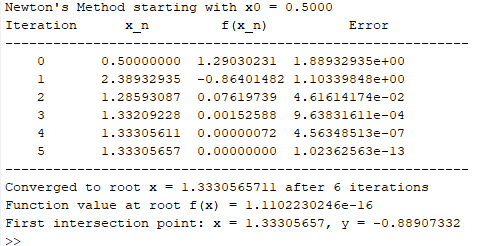


圖 3 問題2的解題過程

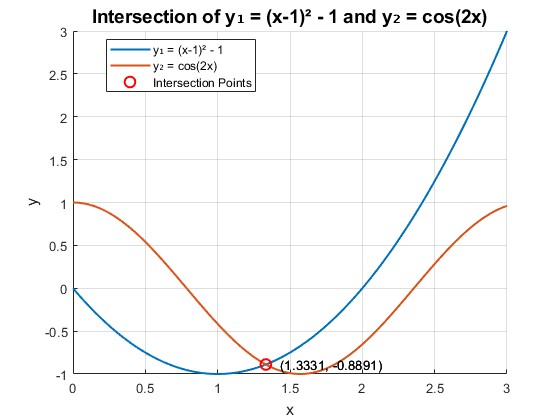


圖 4 問題2的解

1. 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。

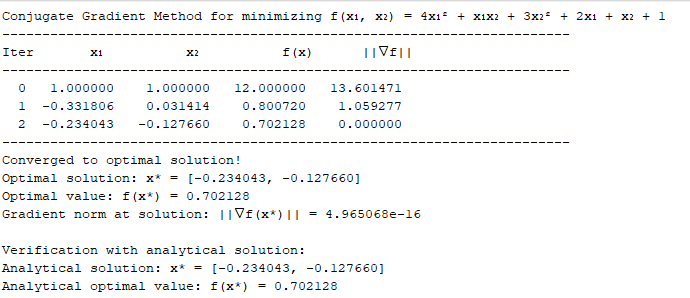


圖 5 問題3求解過程

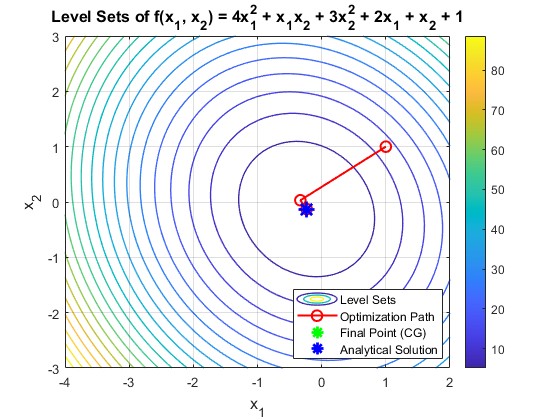


圖 6 問題3求解過程可視化(2D)

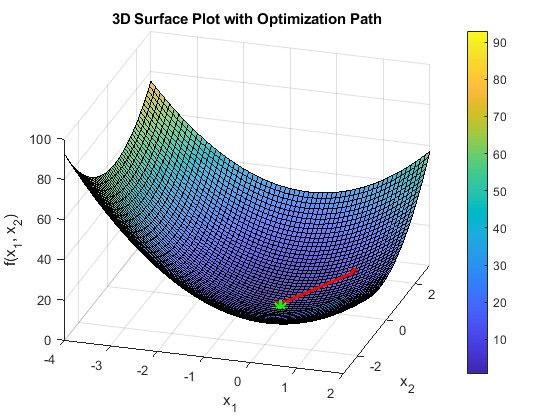
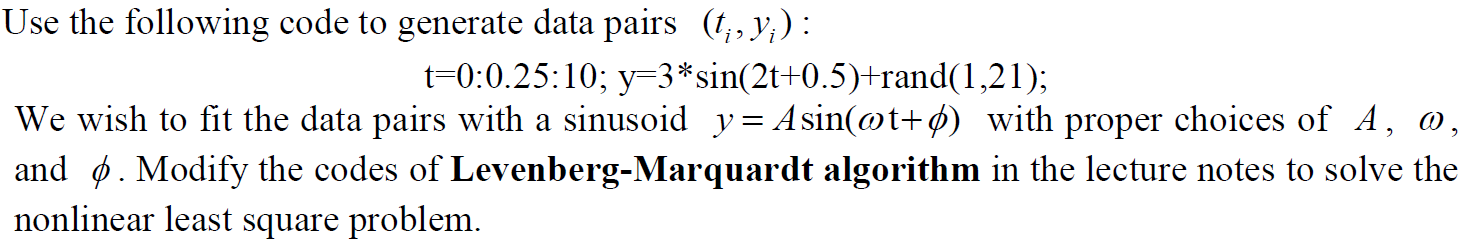


圖 7 問題3求解過程可視化(3D)

1. Levenberg-Marquardt Algorithm



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。

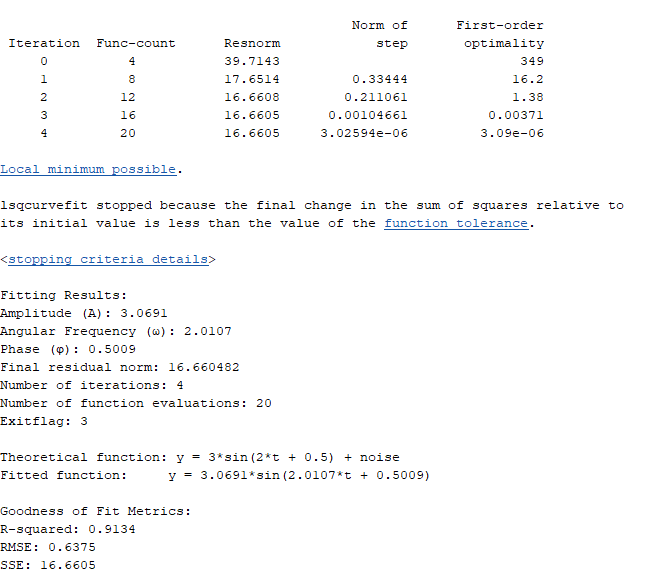


圖 8 問題4求解過程

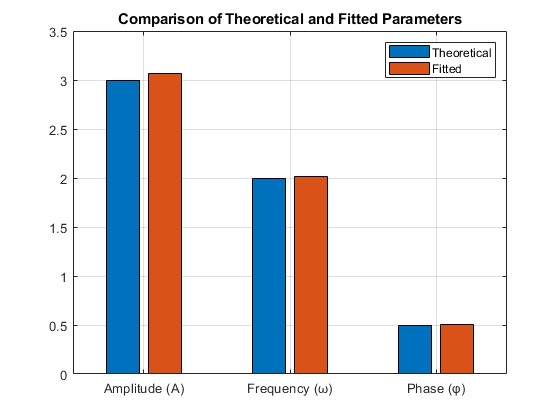


圖 9 理論值與擬合值

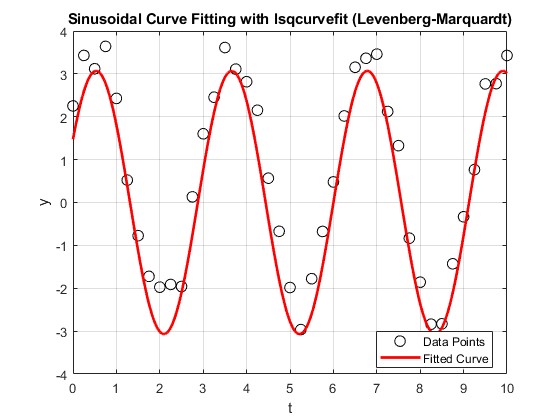


圖 10 加入雜訊的資料與計算出的擬合方程式

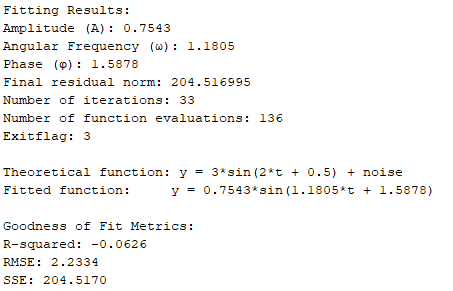


圖 11 問題4求解過程(2)

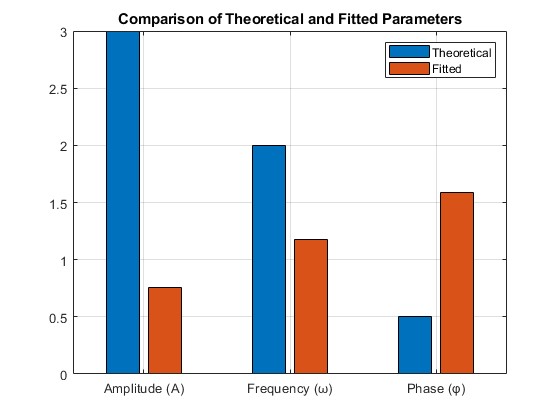


圖 12 理論值與擬合值(2)

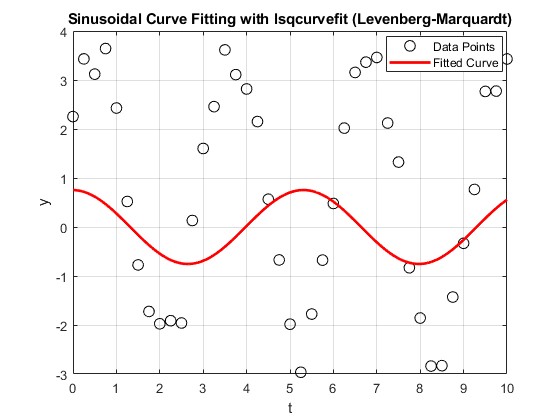
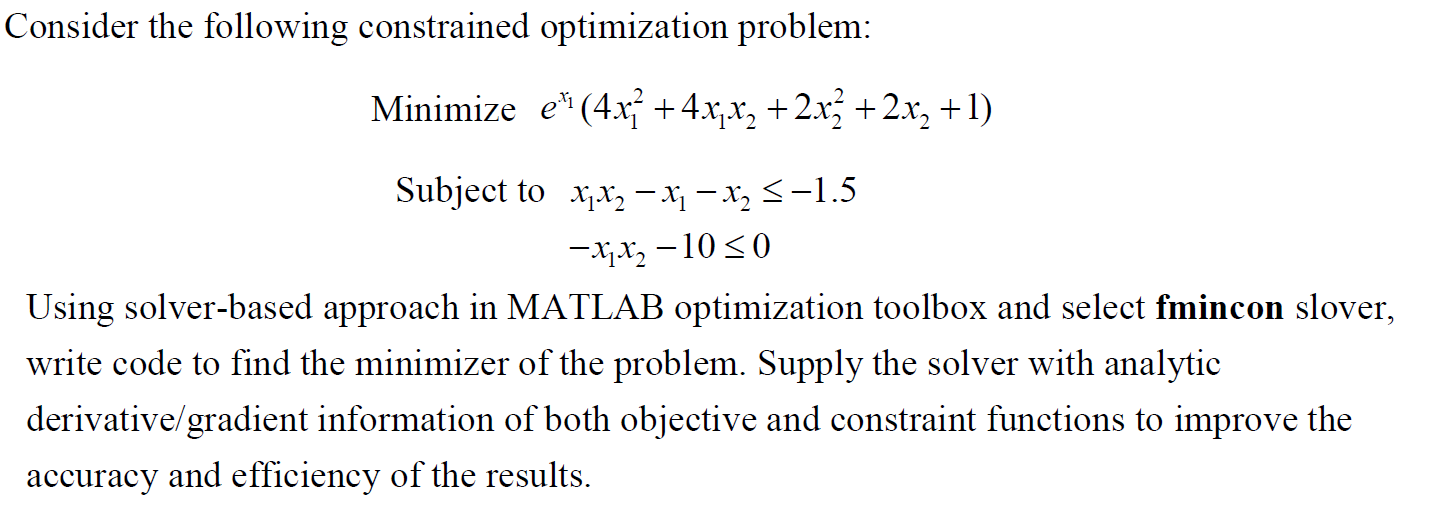


圖 13 加入雜訊的資料與計算出的擬合方程式(2)

1. Solver-Based Approach



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。

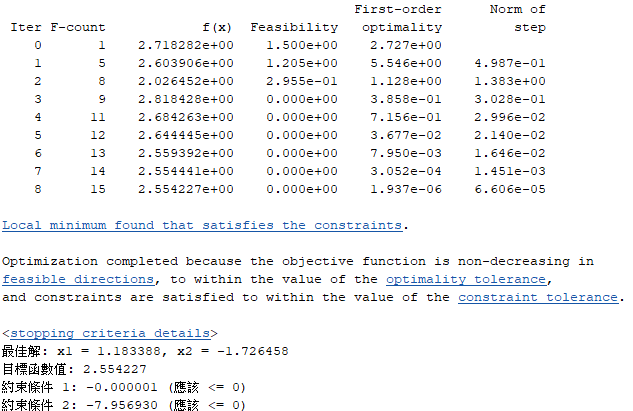


圖 14 問題5執行過程

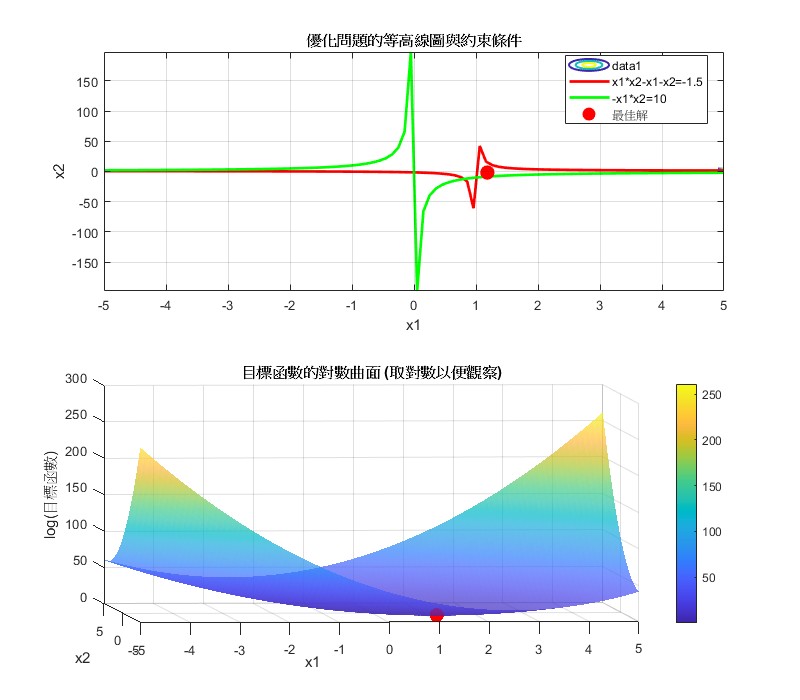
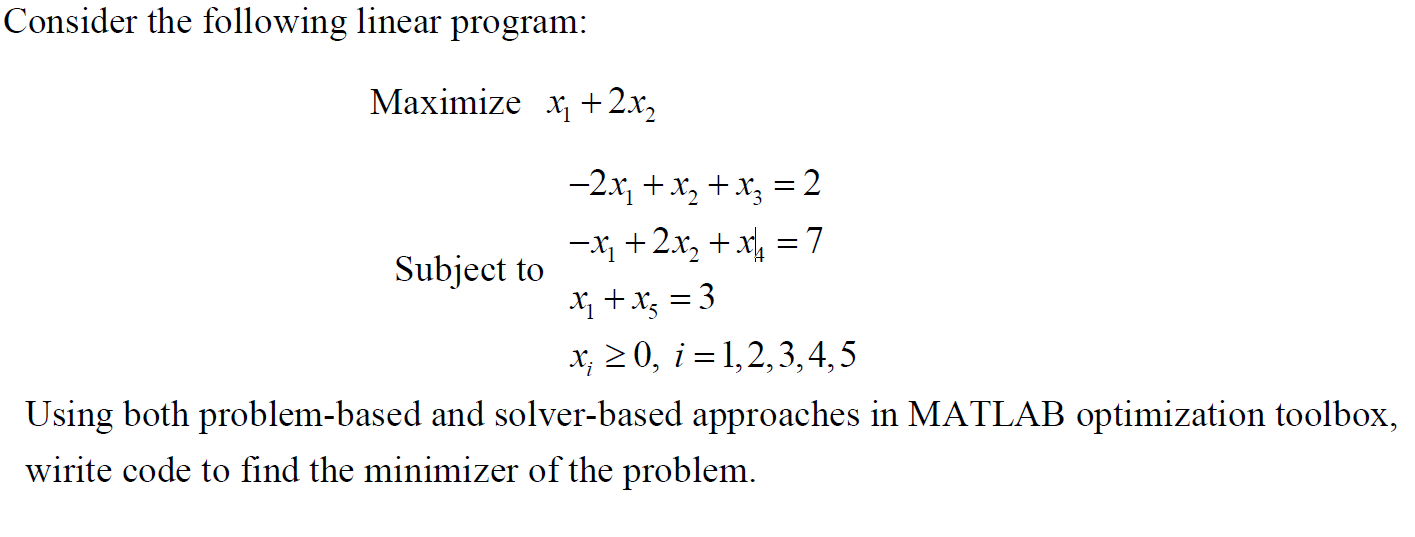


圖 15 問題5執行結果

1. Problem-Based & Solver-Based Approach



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。

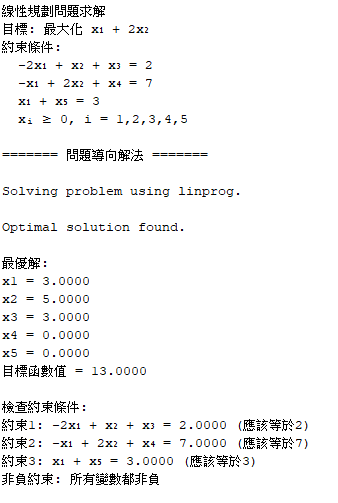


圖 16 問題導向法求解過程與結果

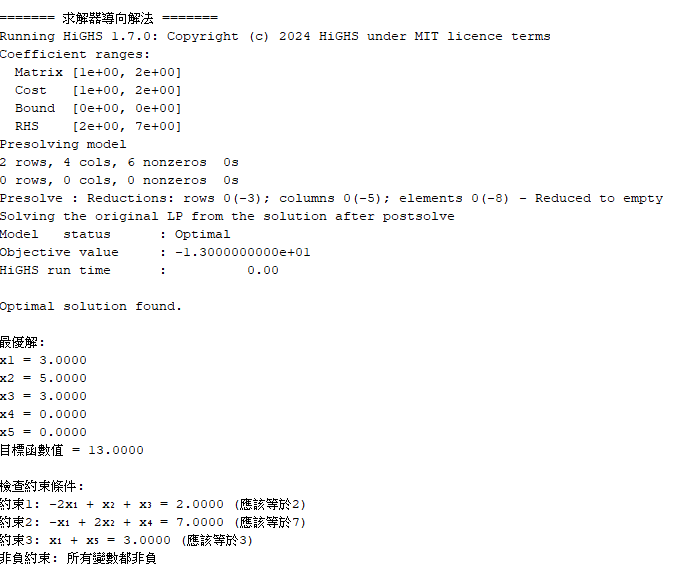


圖 17 求解器法求解過程與結果

**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**;**

1. 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**

1. 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**);**

1. 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**));**

***% 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**);**

1. 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**

1. 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'**);**

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**);**

**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**