ASE 381.P3 Optimal Control Theory
Homework 2

Junette Hsin
Masters Student, Aerospace Engineering and Engineering Mechanics, University of Texas, Austin, TX 78712

Problem 1

Problem 1: Write your own program in MATLAB, python or other similar programming language that will determine numerically the solution of the following unconstrained optimization problem:

UOP#1: Minimize the performance index

$$f(x) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}Q\mathbf{x} - \mathbf{x}^{\mathrm{T}}b,$$

where $\mathbf{x} \in \mathbb{R}^m$, $Q \in \mathbb{R}^{m\times m}$, $Q = Q^{\mathrm{T}} > 0$, $b \in \mathbb{R}^m$. (Note: You should all know how to find the analytic expression for the solution of this problem; we have solved a very similar problem in class soon. The objective here is to address the same problem numerically).

The matrix Q and the vector b should be two of the input variables to your program whereas an approximation $\tilde{\mathbf{x}}_\star$ of the unique global minimizer $\mathbf{x}_\star$ of f should be the output variable of your program. In particular, the main steps of the iterative process implemented by your program are the following:
(1) Make an initial guess $\mathbf{x}^0$ for the unknown minimizer.
(2) Start the iterative process by taking $\mathbf{x}^{k+1} = \mathbf{x}^k - a_k g(\mathbf{x}^k)$, where $g(\mathbf{x}^k) = Q\mathbf{x}^k - b$ and a_k is a real number (scalar) that is the unique global minimizer of the function

$$E(a_k) = \frac{1}{2}(\mathbf{x}^k - a_k g(\mathbf{x}^k))^{\mathrm{T}}Q(\mathbf{x}^k - a_k g(\mathbf{x}^k)) - (\mathbf{x}^k - a_k g(\mathbf{x}^k))^{\mathrm{T}}b,$$

for a given $\mathbf{x}^k$ (the subscript k refers to the iteration number, not the component of a the vector; $\mathbf{x}^k$ is a vector corresponding to the k-th iteration of the algorithm). Note that you should be able to find the exact expression for a_k by solving the above minimization problem analytically (a_k is just a scalar!).
(3) Stop the process when $\|\mathbf{x}^N - \mathbf{x}^{N-1}\| < \epsilon$ (here $\|\cdot\|$ denotes the Euclidean norm) for some positive integer N and a sufficiently small $\epsilon > 0$ (ϵ: error tolerance). Then, set $\tilde{\mathbf{x}}_\star = \mathbf{x}_N$.
Note that both the error tolerance ϵ and the initial guess $\mathbf{x}^0$ should be two additional input variables of your program.

Test your program for the case when $Q = \begin{bmatrix} 12 & 1 \\ 1 & 2 \end{bmatrix}$, $b = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ and $\epsilon = 0.001$. First, find the exact value $\mathbf{x}_\star$ of the strict global minimizer analytically as we did in class (you do not need the iterative process for that). Compute an approximation $\tilde{\mathbf{x}}_\star$ of the global minimizer using your program for three different initial guesses $\mathbf{x}_0$, namely (1) $\mathbf{x}^0 = [13,\ -2]^{\mathrm{T}}$, (2) $\mathbf{x}^0 = [-10,\ 7]^{\mathrm{T}}$ and (3) $\mathbf{x}^0 = [-2,\ 14]^{\mathrm{T}}$. After how many iterations did your algorithm converge in each case? Plot a diagram of the approximation error $\|\mathbf{x}_\star - \mathbf{x}^k\|$ versus the iteration step k, for each of the three cases. Finally, plot the sequence of points generated by your algorithm in the $x_1 - x_2$ plane.

Note: In your response, include a printout of the source code of your program.

A. Solution

The analytical solution for the strict global minimizer for all 3 cases is $x_* = [0.173913043478261, 0.91304347826087]$.

The unique global minimizer a_k was found analytically by differentiating $E(a_k)$ with respect to a_k, which resulted in the following expression:

$$a_{k*} = [x_k^T Q g_k - g_k^T b][g_k^T Q g_k]^{-1} \tag{1}$$

Code specific to each of the 3 cases are located in their individual sections, but additional functions and code required for each algorithm can be found in the Appendix.

B. Case 1: $\mathbf{x}^0 = [13, -2]^T$

1. Plots

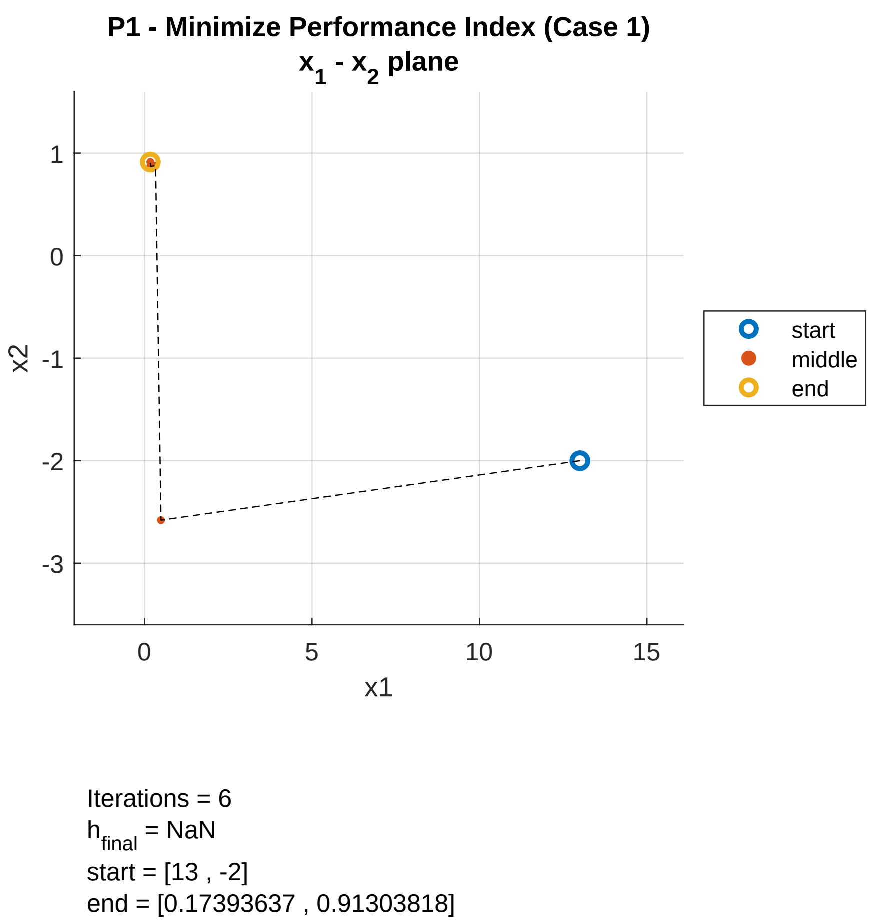

Fig. 1 Problem 1: Case 1: x1-x2 plane

The algorithm took 6 iterations to converge below the threshold $\epsilon = 0.001$.

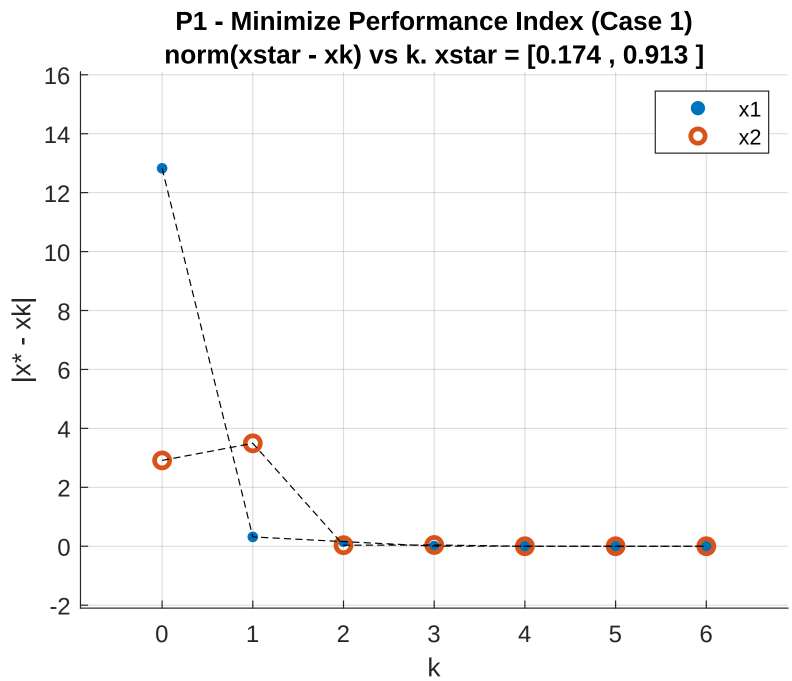

Fig. 2 Problem 1: Case 1: Iteration vs. Error

C. Case 2: $x^0 = [-10, 7]^T$

1. Plots

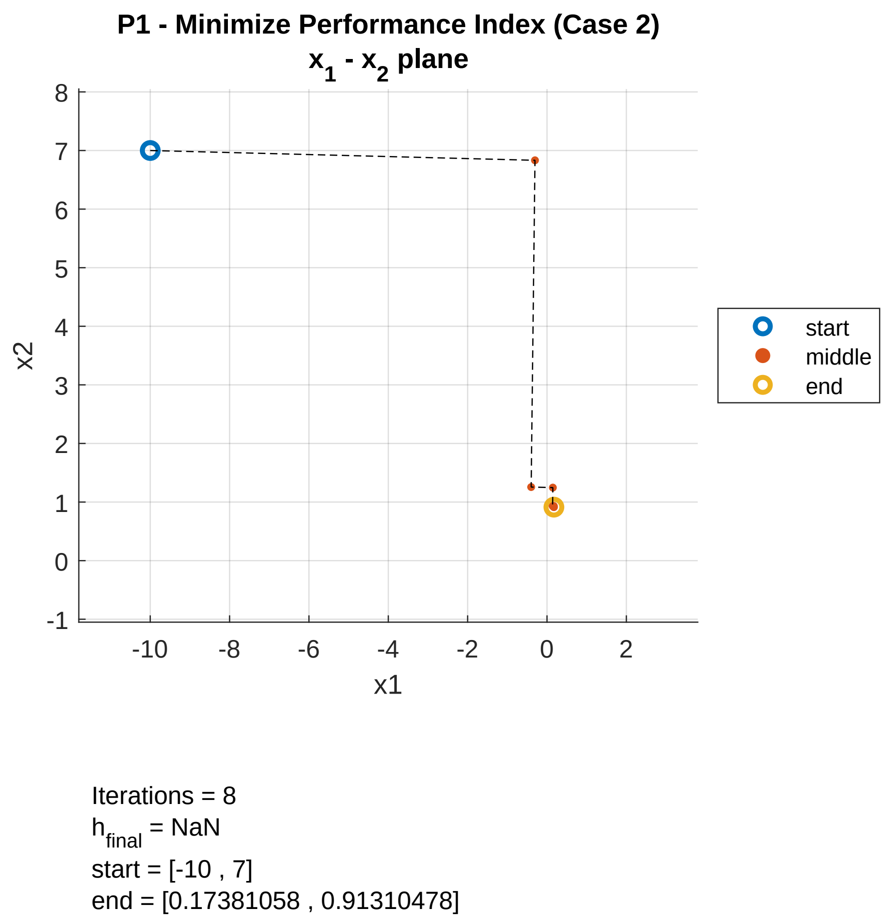

Fig. 3 Problem 1: Case 2: x1-x2 plane

The algorithm took 8 iterations to converge below the threshold $\epsilon = 0.001$.

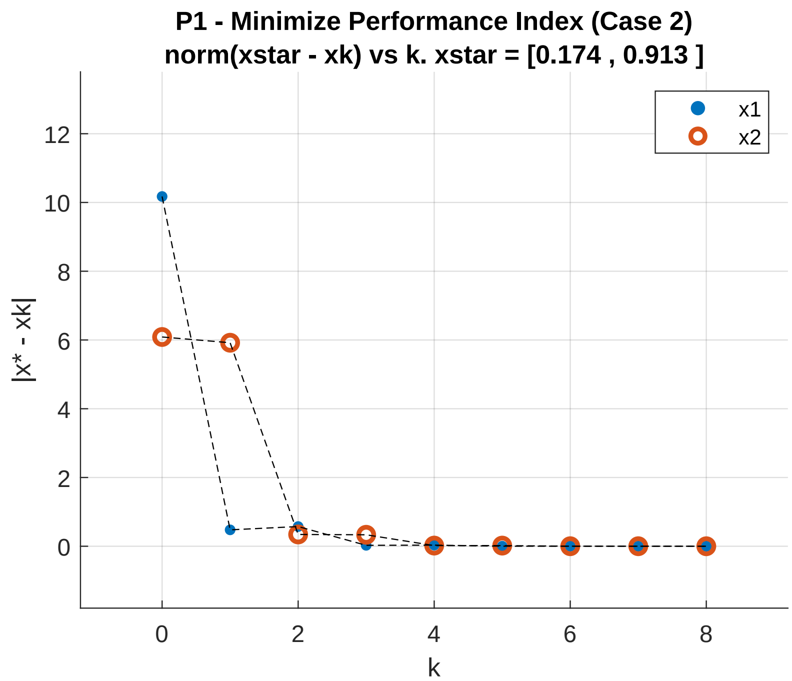

Fig. 4 Problem 1: Case 2: Iteration vs. Error

D. Case 3: x^0 = [-2, 14]^T

1. Plots

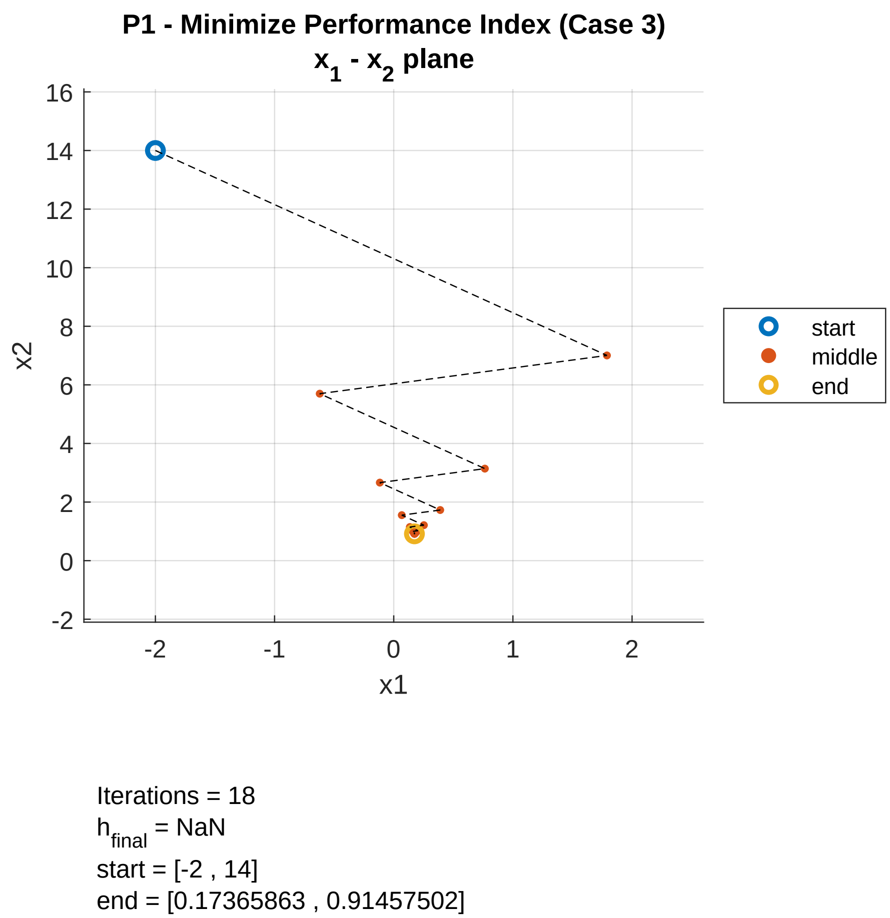

Fig. 5 Problem 1: Case 3: x1-x2 plane

The algorithm took 18 iterations to converge below the threshold $\epsilon = 0.001$.

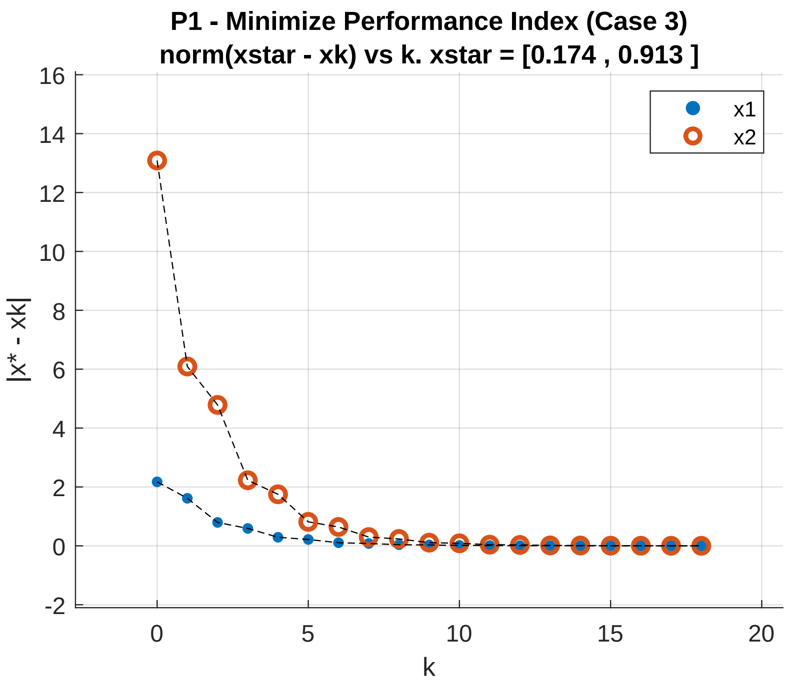

Fig. 6 Problem 1: Case 3: Iteration vs. Error

E. Code

```
%% Problem 1

clear; close all

% inputs Q and b
Q = [12 1; 1 2];
b = [3; 2];

% analytical solution
xstar = b' * Q^-1;

% g function
g = @(x) Q*x - b;

% error threshold
err   = 1e-3;
delta = 1;

% FIRST GUESS
disp('FIRST GUESS')
x0 = [13; -2];
[x_arr, i] = min_perf(delta, err, x0, Q, b, g);

        % plot 1
        fname1 = 'P1 - Minimize Performance Index (Case 1)';
        plot_x1x2(fname1, x_arr, i)

        % plot 2
        plot_xstar_err(fname1, x_arr, xstar, i)

% SECOND GUESS
```

```
disp('SECOND GUESS')
x0 = [-10; 7];
[x_arr, i] = min_perf(delta, err, x0, Q, b, g);

        % plot
        fname1 = 'P1 - Minimize Performance Index (Case 2)';
        plot_x1x2(fname1, x_arr, i)

        % plot 2
        plot_xstar_err(fname1, x_arr, xstar, i)

% THIRD GUESS
disp('THIRD GUESS')
x0 = [-2; 14];
[x_arr, i] = min_perf(delta, err, x0, Q, b, g);

        % plot
        fname1 = 'P1 - Minimize Performance Index (Case 3)';
        plot_x1x2(fname1, x_arr, i)

        % plot 2
        plot_xstar_err(fname1, x_arr, xstar, i)
```

Problem 2

Problem 2: Write a program in MATLAB, python or any other similar programming language you like that solves the following equality constrained optimization problem:

COP#2: Minimize the performance index

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

under the equality constraint

$$h(x_1, x_2) = 0, \text{ where } h(x_1, x_2) = (x_1 + 0.5)^2 + (x_2 + 0.5)^2 - 0.25.$$

Your program will implement the so-called *penalty* method. The main idea of this method is to obtain the solution to the constrained optimization problem by constructing a sequence of points $\{\mathbf{x}^k\}$, where $\mathbf{x}^k := [x_1^k,\ x_2^k]^{\mathrm{T}}$ is, for each $k \in \{1, 2, \ldots\}$, the global minimizer of the following (k-th) unconstrained optimization problem:

UOP#2: Minimize the performance index

$$\phi_k(x_1, x_2) = f(x_1, x_2) + \frac{1}{2} a_k h^2(x_1, x_2),$$

where $a_k = \beta a_{k-1}$.

The algorithm terminates when $|h(x_1^k, x_2^k)| \leq 10^{-4}$. For your simulations, take $a_0 = 0.1$, $\beta = 6$, $[x_1^0,\ x_2^0]^{\mathrm{T}} = [1,\ 1]^{\mathrm{T}}$. What is the computed minimizer? How many iterations were needed before successful termination? What is the value of $h(x_1^k, x_2^k)$ at the last step? Plot the sequence of points generated by your algorithm in the $x_1 - x_2$ plane.

For the computation of the global minimizer $\mathbf{x}^k := [x_1^k,\ x_2^k]^{\mathrm{T}}$ of the k-th unconstrained optimization problem, you can use MATLAB's function `fminsearch`.

Note: In your response, include a printout of the source code of your program.

F. Solution

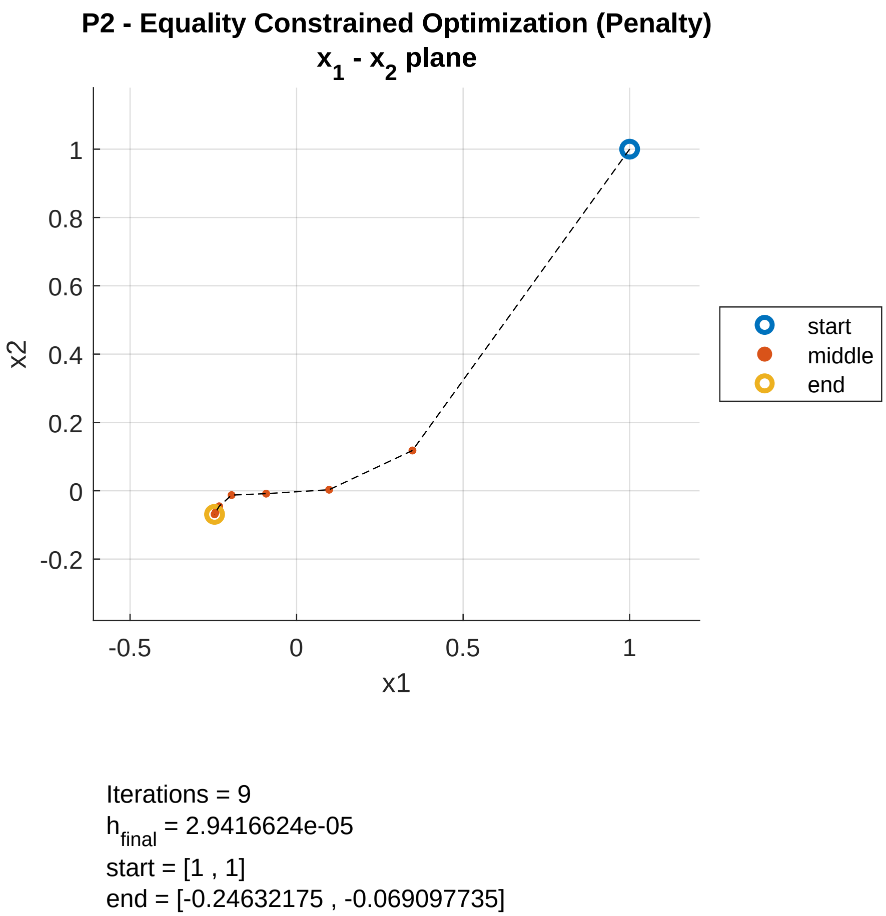

Fig. 7 Problem 2: x1-x2 plane

The computed minimizer is [-0.246321750090767, -0.0690977348084581]. The algorithm required 9 iterations before convergence. The value of h at the last step is 2.94166242132965e-05.

G. Code

```
%% Problem 2

clear;

x = sym('x', [2 1]);

% create performance index functions
f = 100 * ( x(2) - x(1)^2 )^2 + ( 1 - x(1) )^2;
f = matlabFunction(f);
h = ( x(1) + 0.5 )^2 + ( x(2) + 0.5 )^2 - 0.25;
```

```
h = matlabFunction(h);

% initialize
b     = 6;
a0    = 0.1;
x0    = [1; 1];
err   = 10^-4;

% 0 iteration
j     = 0;
akm1 = a0;
xkm1 = x0;
h_err = h(xkm1(1), xkm1(2));
x_arr = x0';

% iterate
while h_err > err

        % current index
        j  = j + 1;
        ak = b * akm1;

        phi = @(x) f(x(1), x(2)) + 1/2 * ak * h(x(1), x(2))^2;
        xk  = fminsearch(phi, xkm1);

        % new penalty
        h_err = norm(h(xk(1), xk(2)));

        % save output
        x_arr = [x_arr; xk'];

        % set up next index
        akm1 = ak;
        xkm1 = xk;

end

% plot
fname1 = 'P2 - Equality Constrained Optimization (Penalty)';
plot_x1x2(fname1, x_arr, j, h_err)
```

Problem 3

Problem 3: Write a program in MATLAB, python or any other similar programming language that solves the equality constrained optimization problem given in Problem 2 (COP#2) but this time use the *Lagrange multiplier* method. Again, the main idea of the new algorithm is to obtain the solution to the constrained optimization problem by constructing a sequence of points $\{\mathbf{x}^k\}$, where $\mathbf{x}^k := [x_1^k,\ x_2^k]^{\mathrm{T}}$ is, for each $k \in \{1, 2, \dots\}$, the global minimizer of the following (k-th) unconstrained optimization problem:

UOP#3: Minimize the performance index:

$$\psi_k(x_1, x_2) = f(x_1, x_2) + \lambda_k h(x_1, x_2) + \frac{1}{2} a_k h^2(x_1, x_2),$$

where $\lambda_k = \lambda_{k-1} + a_{k-1} h(x_1^{k-1}, x_2^{k-1})$ *and* $a_k = \beta a_{k-1}$. The process terminates when $|h(x_1^k, x_2^k)| \leq 10^{-4}$. For your simulations, take $a_0 = 0.1$, $\beta = 6$, $\lambda_0 = 10$ and $[x_1^0,\ x_2^0]^{\mathrm{T}} = [1,\ 1]^{\mathrm{T}}$.
What is the computed minimizer? How many iterations were needed before successful termination? What is the value of $h(x_1^k, x_2^k)$ at the last step? Plot the sequence of points generated by your algorithm in the $x_1 - x_2$ plane. Compare your answers with those in Problem 2.

Again, for the computation of the global minimizer $\mathbf{x}^k := [x_1^k,\ x_2^k]^{\mathrm{T}}$ of the k-th unconstrained optimization problem, you can use MATLAB's function `fminsearch`.

Note: In your response, include a printout of the source code of your program.

H. Solution

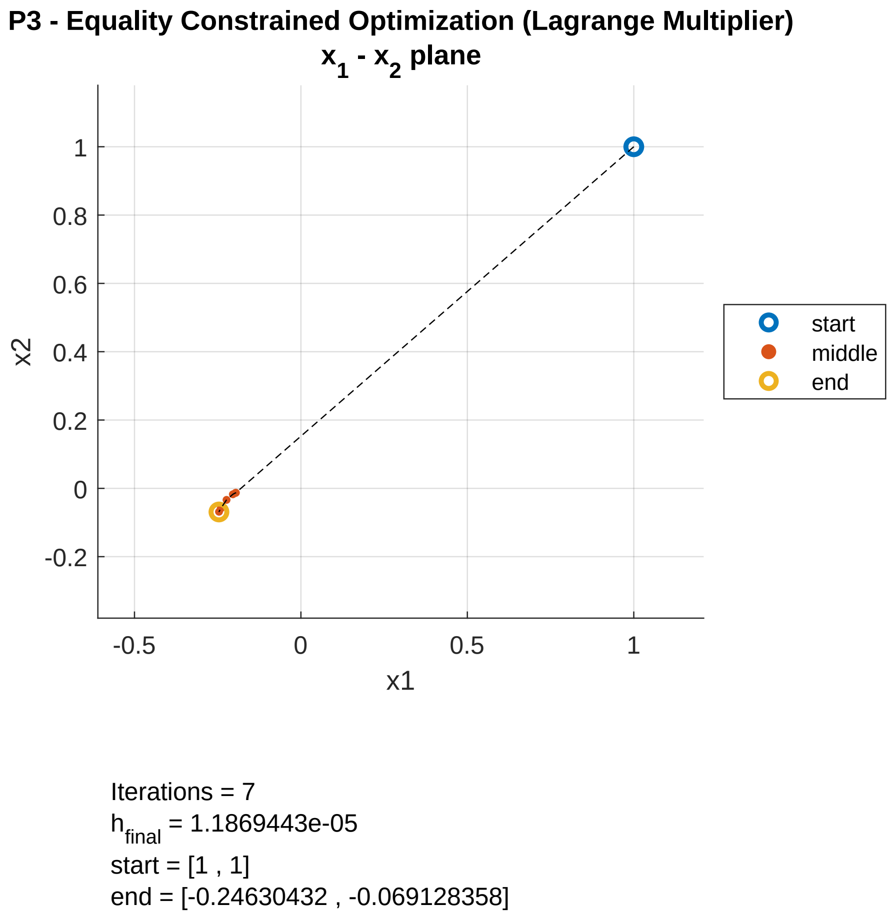

Fig. 8 Problem 3: x1-x2 plane

The computed minimizer is [-0.246304320398368, -0.069128358330955]. The algorithm required 7 iterations before convergence. The value of h at the last step is 1.18694431120447e-05.

The penalty method was used for Problems 2 and 3, but in Problem 3 the Lagrange multiplier method was also implemented. The result is that the Problem 3 algorithm converged more quickly (7 iterations vs. 9) and the penalty value on the final iteration was smaller (1.187e-5 vs. 2.942e-5) than in Problem 2. Additionally, the trajectory of the computed solution at each iteration is straighter and more direct in Problem 3 which can be seen by comparing their plots. The lagrange multiplier method improved the algorithm's performance.

I. Code

```
%% Problem 3

clear;

```

```matlab
x = sym('x', [2 1]);

% create performance index functions
f = 100 * (x(2) - x(1)^2)^2 + (1 - x(1))^2;
f = matlabFunction(f);
h = (x(1) + 0.5)^2 + (x(2) + 0.5)^2 - 0.25;
h = matlabFunction(h);

% initialize
b     = 6;
a0    = 0.1;
x0    = [1; 1];
err   = 10^-4;
k     = 0;
lmda0 = 10;

% first iteration
akm1 = a0;
xkm1 = x0;
lmdakm1 = lmda0;
h_err = h(xkm1(1), xkm1(2));
x_arr = x0';

% iterate
while h_err > err

        % current index
        k     = k + 1;
        ak    = b * akm1;
        lmdak = lmdakm1 + akm1 * h(xkm1(1), xkm1(2));

        phi = @(x) f(x(1), x(2)) + lmdak * h(x(1), x(2)) + 1/2 * ak * h(x(1), x(2))^2;
        xk  = fminsearch(phi, xkm1);

        % penalty
        h_err = norm(h(xk(1), xk(2)));

        % save output
        x_arr = [x_arr; xk'];

        % set up next index
        akm1    = ak;
        lmdakm1 = lmdak;
        xkm1    = xk;

end

% plot
fname1 = 'P3 - Equality Constrained Optimization (Lagrange Multiplier)';
plot_x1x2(fname1, x_arr, k, h_err)
```

Appendix

Supplementary MATLAB code

```
%% subfunctions

function [x_arr, i] = min_perf(delta, err, x0, Q, b, g)
% Minimize performance index

% initialize
i     = 0;
x_arr = x0';
xkm1  = x0;

% start iterative process
while delta > err

        % current index
        i = i + 1;

        % calc ak
%       ak = ( 1/2 * g(xkm1)' * Q * xkm1 + 1/2 * xkm1' * Q * g(xkm1) - g(xkm1)' * b) * ...
%           ( g(xkm1)' * Q * g(xkm1) )^-1;
        ak = inv(g(xkm1)' * Q* g(xkm1)) * (xkm1' * Q * g(xkm1) - g(xkm1)' * b );

        % calc xk
        xk = xkm1 - ak * g(xkm1);
        delta = norm(xkm1 - xk);

        if isnan(delta)
                disp('Contains NaNs')
                break
        end

        % save output
        x_arr = [x_arr; xk'];

        % set up next index
        xkm1 = xk;

end

end

% ------------------------------------------------------------------------

function plot_x1x2(fname1, x_arr, k, h_err)

if ~exist('h_err', 'var')
        h_err = NaN;
end
% plot x1-x2 plane

fname2 = 'x_1 - x_2 plane';

figure('name', [fname1 ' - ' fname2], 'position', [100 100 600 600])
        subplot(3,1,1:2)
                hold on; grid on;
                scatter(x_arr(1,1), x_arr(1,2), 40, 'linewidth', 2);
                scatter(x_arr(2:end-1,1), x_arr(2:end-1,2), 10, 'filled')
                scatter(x_arr(end,1), x_arr(end,2), 40, 'linewidth', 2);
                plot(x_arr(:,1), x_arr(:,2), '--k');
                bigger_lim;

                xlabel('x1'); ylabel('x2');
                legend('start', 'middle', 'end', 'location', 'eastoutside')
                title( {fname1; fname2} );
        subplot(3,1,3)
```

```matlab
		pos = get(gca, 'position');
%		text = {'test1'; 'test2'; 'test3'};

		text = { ''; '';
			sprintf('Iterations = %d', k);
			sprintf('h_{final} = %.8g', h_err);
			sprintf( 'start = [%.8g , %.8g]', x_arr(1,1), x_arr(1,2) );
			sprintf( 'end = [%.8g , %.8g]', x_arr(end,1), x_arr(end,2) ) };

		annotation('textbox', pos, ...
			'String', text, ...
			'edgecolor', 'none');
		axis off

end

function plot_xstar_err(fname1, x_arr, xstar, i)

	% plot | xstar - xk |
	dx = abs(x_arr - xstar);
	fname2 = sprintf('norm(xstar - xk) vs k. xstar = [%.3g , %.3g ]', xstar(1), xstar(2));
	figure('name', [fname1 ' - ' fname2] );
		hold on; grid on;
		scatter( [0:1:i]', dx(:,1), 20, 'filled');
		scatter( [0:1:i]', dx(:,2), 40, 'linewidth', 2);
		plot( [0:1:i]', dx(:,1), '--k');
		plot( [0:1:i]', dx(:,2), '--k');

		legend('x1', 'x2')
		bigger_lim

		xlabel('k')
		ylabel('|x* - xk|')

		title( {fname1; fname2} );

end

function bigger_lim
% Increase y-axis limits on plot by 30% on current axes

	ylims		= get(gca, 'ylim');
	yrange		= ylims(2) - ylims(1);
	new_ylim	= [ ylims(1) - 0.15*yrange, ylims(2) + 0.15*yrange ];
	set(gca, 'ylim', new_ylim);

	xlims		= get(gca, 'xlim');
	xrange		= xlims(2) - xlims(1);
	new_xlim	= [ xlims(1) - 0.15*xrange, xlims(2) + 0.15*xrange ];
	set(gca, 'xlim', new_xlim);

end
```