Homework 1: Dimitri Lezcano

Thursday, September 17, 2020 2:48 PM

1) a) $L(x) = (1-x_1)^2 + 200(x_2 - x_1^2)^2$

$$\nabla L = \begin{pmatrix} -2(1-x_1) + 400(-2x_1)(x_2 - x_1^2) \\ 400(x_2 - x_1^2) \end{pmatrix}$$

$$= \begin{pmatrix} -2 + 2x_1 - 800(x_1 x_2 - x_1^3) \\ 400(x_2 - x_1^2) \end{pmatrix}$$

$\nabla L = 0$ when

$400(x_2 - x_1^2) = 0$

$\Rightarrow x_2 = x_1^2$

and

$-2 + 2x_1 - 800(x_1 \cdot x_2^2 - x_1^3) = 0$

$\Rightarrow -2 + 2x_1 = 0$

$\Rightarrow x_1 = 1 \Rightarrow x_2 = 1^2 = 1$

so $x^* = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ is the critical point

So look at the Hessian:

$$H_L(x) = \begin{pmatrix} \partial^2 L/\partial x_1^2 & \partial^2 L/\partial x_1 \partial x_2 \\ \partial^2 L/\partial x_2 \partial x_1 & \partial^2 L/\partial x_2^2 \end{pmatrix}$$

$$\frac{\partial^2 L}{\partial x^2} = 2 - 800(x_2 - 3x_1^2)$$

$$\frac{\partial^2 L}{\partial x_1 \partial x_2} = \frac{\partial^2 L}{\partial x_2 \partial x_1} = -800 x_1$$

∂^2 [illegible]

$$\frac{\partial^2 L}{\partial x_2^2} = 400$$

$$H_L(x^*) = \begin{pmatrix} 2 - 800(1-3) & -800 \\ -800 & 400 \end{pmatrix}$$

$$H_L(x^*) = \begin{pmatrix} 3202 & -800 \\ -800 & 400 \end{pmatrix}$$

where $\lambda(H_L(x^*)) \approx 3415, 188$

$\therefore H_L > 0 \therefore$

x^* is a minima.

b) $L(u) = (u-1)(u+2)(u-3)$

$$\frac{dL}{du} = (u+2)(u-3) + (u-1)(u-3) + (u-1)(u+2)$$

$$= (u^2 - u - 6) + (u^2 - 4u + 3) + (u^2 + u - 2)$$

$dL/du = 3u^2 - 4u - 5 = 0$

when $u = \dfrac{4 \pm \sqrt{16 + 4 \cdot 3 \cdot 5}}{2 \cdot 3}$

$$u = \frac{4 \pm \sqrt{76}}{6}$$

$$= \frac{2}{3} \pm \frac{\sqrt{19}}{3} = u_1^*, u_2^*$$

$$\frac{d^2 L}{du^2} = [(u+2) + (u-3)] + [(u-1) + (u-3)] + [(u-1) + (u+2)]$$

$$\frac{d^2 L}{du^2} = 6u - 4$$

$$\frac{d^2L}{du^2} = 6u - 4$$

$$\left.\frac{d^2L}{du^2}\right|_{u=u^*} = 6u^* - 4 = 4 \pm 2\sqrt{14} - 4 = \pm 2\sqrt{14}$$

So $u_1^* = \frac{2}{3} + \frac{\sqrt{14}}{3}$ is a minimum

$u_2^* = \frac{2}{3} - \frac{\sqrt{14}}{3}$ is a Maximum

c) $L(u) = (u_1^2 + 3u_1 - 4)(u_2^2 - u_2 + 3)$

$$\nabla L = \begin{pmatrix} (2u_1 + 3)(u_2^2 - u_2 + 3) \\ (u_1^2 + 3u_1 - 4)(2u_2 - 1) \end{pmatrix} = 0$$

when

$(2u_1 + 3)(u_2^2 - u_2 + 3) = 0$ &

$(u_1^2 + 3u_1 - 4)(2u_2 - 1) = 0$

So if $u_1 = -3/2 \Rightarrow 2u_2 - 1 = 0 \Rightarrow u_2 = 1/2$

$v_1^* = \begin{pmatrix} -3/2 \\ 1/2 \end{pmatrix}$

If $(u_1^2 + 3u_1 - 4) = 0$

↳ $(u_1 + 4)(u_1 - 1) = 0$

↳ $u_1 = 1, -4$.

Then $(u_2^2 - u_2 + 3) = 0$

Then $(u_2^2 - u_2 + 3) = 0$

$\Rightarrow u_2 = \frac{1 \pm \sqrt{1 - 4 \cdot 1 \cdot 3}}{2}$

$= \frac{1 \pm \sqrt{-11}}{2}$

$u_2 = \frac{1}{2} \pm i\frac{\sqrt{11}}{2} \notin \mathbb{R}$

So only solution (in the reals) is

$v_1^* = \begin{pmatrix} -3/2 \\ 1/2 \end{pmatrix}$ $\quad 2(9/4 - \frac{9}{2} - 4) = 1(-\frac{9}{2} - 8)$

New Hessian

$\frac{\partial^2 L}{\partial u_1^2} = 2(u_2^2 - u_2 + 3)\Big|_{u=v^*} = 11/2$

$\frac{\partial^2 L}{\partial u_2^2} = 2(u_1^2 + 3u_1 - 4)\Big|_{u=v^*} = -25/2$

$\frac{\partial^2 L}{\partial u_1 \partial u_2} = (2u_1 + 3)(2u_2 - 1)\Big|_{u=v^*} = 0$

So $H_L(v^*) = \begin{pmatrix} 11/2 & 0 \\ 0 & -25/2 \end{pmatrix}$

So $v^* = \begin{pmatrix} -3/2 \\ 1/2 \end{pmatrix}$ is a saddle point

2) $L(x) = \frac{1}{2} x^T x$

subject to $x_1 + x_2 + x_3 = 0$

or $e = (1 \ 1 \ 1)^T$

$e^T x = 0$

easy way: $c^T x = 0$ means

$$x \in \text{span}\left\{ \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} \right\}$$

v_1 v_2

so $x = a v_1 + b v_2$.

Then problem becomes

$L(a,b) = \frac{1}{2}(a v_1 + b v_2)^T (a v_1 + b v_2)$

w/ constraint set is fixed.

$\therefore$

$$L(a,b) = \frac{1}{2}\left(a\|v_1\|^2 + b\|v_2\|^2 + 2ab\, v_1^T v_2\right)$$
$$= \frac{1}{2}\left(a\sqrt{2} + b\sqrt{2} + 2ab\right)$$
$$= \frac{\sqrt{2}}{2}\left(a + b + \sqrt{2}\,ab\right)$$

So $\nabla_{(a,b)} L = \frac{\sqrt{2}}{2}\begin{pmatrix} 1 + \sqrt{2}b \\ 1 + \sqrt{2}a \end{pmatrix} = 0$

iff

$1 + \sqrt{2}b = 0 \Rightarrow b = -\sqrt{2}/2$

$1 + \sqrt{2}a = 0 \Rightarrow a = -\sqrt{2}/2$

$$H_L(a,b) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$\lambda(H_L) = \pm 1$ so

we have

$x^* = -\frac{\sqrt{2}}{2}\begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} - \frac{\sqrt{2}}{2}\begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$ is a saddle point

c | -1 | c | 1 | 1 | point

$x^* = \frac{-\sqrt{2}}{2}\begin{pmatrix} 1 \\ 1 \\ -2 \end{pmatrix}$

b) $L(u) = (u_1^2 + 3u_1 - 4)(u_2^2 - u_2 + 3)$

s.t. $u_1 - 2u_2 = 0$

$\hookrightarrow u_1 = 2u_2$

$L(u) = (4u_2^2 + 6u_2 - 4)(u_2^2 - u_2 + 3)$

$= 2(2u_2^2 + 3u_2 - 2)(u_2^2 - u_2 + 3)$

$= 2[2u_2^4 + (-2+3)u_2^3 + (6-2-3)u_2^2 + (9+2)u_2 - 6]$

$= 2(2u_2^4 + u_2^3 + u_2^2 + 11u_2 - 6)$

$\frac{1}{2}\frac{dL}{du} = 8u_2^3 + 3u_2^2 + 2u_2 + 11 = 0$

when $u_2 \approx -1.168, 0.391 \pm 1.01i$

not real

so $u_2 = -1.168$ and

$u_1 = \frac{1}{2}u_2 = -0.584$

$u^* = \begin{pmatrix} -0.584 \\ -1.168 \end{pmatrix}$

$\frac{d^2L}{du_2^2}\Big|_{u_2 = u_2^*} = 24u_2^2 + 6u_2 + 2\Big|_{u_2 = u_2^*} = 27$

so u^* is a minimum!

So u^* is a minimum!

3) a)

$$\min L(x,u) = \tfrac{1}{2} x^T Q x + u^T R u + s^T x$$
$$\text{s.t. } f(x,u) = Ax + Bu + c = 0$$

$x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $Q \geq 0$, $R > 0$

$A \in M_n(\mathbb{R})$, $B \in M_{n,m}(\mathbb{R})$

$s, c \in \mathbb{R}^n$

i) Necessary and Sufficient conditions

$$H(x,u,\lambda) = L(x,u) + \lambda^T f(x,u)$$

$$= \tfrac{1}{2} x^T Q x + u^T R u + s^T x + \lambda^T (Ax + Bu + c)$$

[1st order]

$$\nabla_\lambda H = f(x,u) = 0 \quad \text{(given by constraint)}$$

$$\nabla_x H = \tfrac{1}{2}(Q + Q^T)x + s + A\lambda = 0$$

$$\nabla_u H = \tfrac{1}{2}(R + R^T)u + B^T\lambda = 0$$

$\because R, Q \geq 0$, $R = R^T$ & $Q = Q^T \therefore$

- $\nabla_x(c^T x) = c$
- $\nabla_x(x^T C x) = \tfrac{1}{2}(C + C^T)x$

$$\nabla_x H = Qx + s + A^T\lambda = 0$$

$$\nabla_u H = Ru + B^T\lambda = 0$$

$$\rightarrow ① \quad u = -R^{-1}B^T\lambda$$

(R^{-1} exists b/c $|R| \neq 0$ b/c $R > 0$)

and we have

$$② \quad \dot{\lambda} = -Qx - S$$

$$L = H - \lambda^T f$$

$$dL = dH - \lambda^T df = 0 \quad @ \text{ optimum}$$

Hessian (x, u)

$$dL \approx (\partial_x H, \partial_u H)\begin{pmatrix} dx \\ du \end{pmatrix} + \frac{1}{2}\begin{pmatrix} dx \\ du \end{pmatrix}^T H_H(x,u)\begin{pmatrix} dx \\ du \end{pmatrix} - \lambda^T df$$

(to 2nd order)

by constraint $f = 0 \Rightarrow df = 0$

$$\Rightarrow df = \partial_x f\, dx + \partial_u f\, du = A\,dx + B\,du$$

By 1st order condition

$$(\partial_x H, \partial_u H)\begin{pmatrix} dx \\ du \end{pmatrix} = 0$$

so

$$dL \approx \frac{1}{2}\begin{pmatrix} dx \\ du \end{pmatrix}^T \begin{pmatrix} H_{xx} & H_{xu} \\ H_{ux} & H_{uu} \end{pmatrix}\begin{pmatrix} dx \\ du \end{pmatrix}$$

$$H_{xx} = Q$$

$$H_{xu} = H_{ux} = 0$$

$$H_{uu} = R$$

$$dL = \frac{1}{2}\begin{pmatrix} dx \\ du \end{pmatrix}^T \begin{pmatrix} Q & 0 \\ 0 & R \end{pmatrix}\begin{pmatrix} dx \\ du \end{pmatrix}$$

$$③ = \frac{1}{2} dx^T Q\, dx + \frac{1}{2} du^T R\, du \geq 0$$

(3) $= \frac{1}{2} dx^T Q dx + \frac{1}{2} du^T R du \geq 0$ which is true! $\because Q \geq 0$

s.t. $A dx = -B du$ (from $df = 0$) $R \geq 0$

(1), (2), (3) are the necessary and sufficient conditions. (really only need (1) & (2))

ii) Assume A full rank

(1) $u = -R^{-1} B^T \lambda$

(2) $A^T \lambda = -Qx - s$

(3) $dx^T Q dx + du^T R du$ s.t. $A dx = -B du$

$A^{-T} = (A^T)^{-1}$

(2) $\Rightarrow \lambda = -A^{-T} Q x - s$

(1) + (2) $\Rightarrow u = -R^{-1} B^T \lambda$

$= -R^{-1} B^T (-A^{-T} Q x - A^{-T} s)$

(4) $u = R^{-1} B^T A^{-T} (Qx + s)$

(4) + constraint $\Rightarrow$

$Ax + Bu + c = 0$

$Ax + B(R^{-1} B^T A^{-T})(Qx + s) + c = 0$

$(A + B R^{-1} B^T A^{-T} Q) x = -(B R^{-1} B^T A^{-T} s + c)$

$A(I + A^{-1} B R^{-1} B^T A^{-T} Q) x = -(B R^{-1} B^T A^{-T} s + c)$

$$C = A^{-1} B R^{-1} B^T A^{-T} Q$$

Note that $(A^{-1} B R^{-1} B^T A^{-T})^T$

$= A^{-1} B R^{-T} B^T A^{-T}$

$= A^{-1} B R^{-1} B^T A^{-T}$ $\because R^{-T} = R^{-1}$

$\Rightarrow = U$ where $U = U^T$ $\because R^{-1} \succeq 0$

$C = U \cdot Q$ $\therefore U \succeq 0$ $\because R \succeq 0$

and since $U, Q \succeq 0$, $C \succeq 0$.

Thus $\exists\, V \in O(n)$, $D \succeq 0$ diagonal s.t.

$C = V D V^T$ $\therefore$

$(I + C) = (I + V D V^T) = V(I + D) V^T$

and $(I + C)^{-1} = V (I + D)^{-1} V^T$

where $(I + D)^{-1} = \begin{pmatrix} \frac{1}{1+\lambda_1} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{1+\lambda_n} \end{pmatrix}$

$\lambda_i \in \sigma(C) \geq 0$.

Therefore:

$A(I + C) x = -(B R^{-1} B^T A^{-T} s + C)$

$(I + C) x = -(U s + A^{-1} C)$

$x = -(I + C)^{-1} (U s + A^{-1} C)$

and finally

$u = -R^{-1} B^T A^{-T} (Q x + s)$

$u = -R^{-1} B^T A^{-T} s + R^{-1} B^T A^{-T} (I + C)^{-1} (U s + A^{-1} C)$

b) $\min \left(L(y) = \frac{1}{2} y^T M y + k^T y \right)$

s.t.

$f(\quad) = 1 \quad \quad = 0$

s.t.

$$f(y) = Ay + c = 0$$

$y \in \mathbb{R}^n$, $M \succeq 0$, $A \in \mathbb{R}^{m \times n}$ for $m \le n$ full rank,
$k \in \mathbb{R}^n$, $c \in \mathbb{R}^m$ $\rightarrow A^T \in \mathbb{R}^{n \times m}$ full rank $m \le n$

$f \in \mathbb{R}^m$ $\quad rk(A) = rk(A^T) = m$

$c \in \mathbb{R}^n = \mathbb{R}^{(n-m)+m}$

$$\nabla_y L(y^*) = -\sum_{i=1}^{m} \lambda_i \nabla_y f_i(y^*) \in \mathbb{R}^n$$

$$\nabla_y L(y^*) = My + k \in \mathbb{R}^n$$

$$\left(\sum_i \lambda_i \nabla f_i\right)_j = \sum_{i=1}^{m} \lambda_i \frac{\partial f_i}{\partial x_j}$$

$$= \sum_{i=1}^{m} \frac{\partial f_i}{\partial x_j} \lambda_i$$

$$\frac{\partial f_i}{\partial x_j} = \frac{\partial}{\partial x_j} (Ax + c)_i$$

$$= \frac{\partial}{\partial x_j} (Ax)_i$$

$$= \frac{\partial}{\partial x_j} \left(\sum_k A_{ik} x_k\right)$$

$$= A_{ij}$$

$$\Rightarrow$$

$$\left(\sum_i \lambda_i \nabla f_i\right)_j = \sum_i \lambda_i \frac{\partial f_i}{\partial x_j}$$

$$= \sum_i A_{ij} \lambda_i$$

$$= \sum_i A_{ij} \lambda_i$$
$$= A^T \lambda$$

so $M y^* + K = -A^T \lambda$

$$y^* = -M^{-1} A^T \lambda - M^{-1} K$$

by constraint $A y^* + C = 0$

$$-A M^{-1} A^T \lambda - M^{-1} K + C = 0$$
$$A M^{-1} A^T \lambda = C - M^{-1} K$$

$\therefore A, M$ full rank, $A M^{-1} A^T$ is full rank so

$$\lambda = (A M^{-1} A^T)^{-1} (C - M^{-1} K)$$

so $y^* = -M^{-1} A^T (A M^{-1} A^T)^{-1} (C - M^{-1} K) - M^{-1} K$

$H_L = M > 0$ $\therefore$ for all critical points y_C

y_C is a global minimum.

$\therefore$ since y^* is a critical point

y^* is a global minimum

4) a) Changing the step-size causes for gradient descent to be unstable and overshoot and go to ∞. the smaller the step size, the more stable.

b) Newton's method is MUCH more efficient than gradient Descent

c) Again Newton's method converges much faster than gradient descent.

d) $\min \; L(x,u) = x^2 + 4x + 8u^2$

s.t. $f(x,u) = x - 7u + 5 \leq 0$

$y = \begin{pmatrix} x \\ u \end{pmatrix}$

unconstrained:

$\nabla L = \begin{pmatrix} 2x+4 \\ 16u \end{pmatrix} = 0$ if $\begin{pmatrix} x \\ u \end{pmatrix} = \begin{pmatrix} -2 \\ 0 \end{pmatrix} = y^*$

$f(y^*) = -2 - 7 \cdot 0 + 5 = 3 \not\leq 0$.

So

$\nabla L = -\lambda \nabla f$

$\begin{pmatrix} 2x+4 \\ 16u \end{pmatrix} = -\lambda \begin{pmatrix} 1 \\ -7 \end{pmatrix}$

$16u = 7\lambda$

$\lambda = -2x - 4$

$16u = 7(-2x-4)$

① $14x + 16u = -28$ Good constraint

$x - 7u + 5 = 0$

② $x - 7u = -5$

① + ② ⇒

$\begin{pmatrix} 14 & 16 \\ 1 & -7 \end{pmatrix} \begin{pmatrix} x \\ u \end{pmatrix} = \begin{pmatrix} -28 \\ -5 \end{pmatrix}$

$$y^* = \begin{pmatrix} x \\ u \end{pmatrix} \approx \begin{pmatrix} -2.4211 \\ 0.3684 \end{pmatrix}$$

5) $f^k(d) = f(x^k) + \nabla f(x^k)^T d + \frac{1}{2} d^T \nabla^2 f(x^k) d$

$\min f^k(d)$ (solution is d^k)
s.t. $\|d\| \le \gamma^k$

for $\gamma^k > 0$

same above equivalent to solving

$$(\nabla^2 f(x^k) + \delta^k I) d^k = -\nabla f(x^k)$$

first $\|d\| \le \gamma_k \equiv \|d\|^2 \le \gamma_k^2$

$\equiv d^T d \le \gamma_k^2$

constraint $f(d) = d^T d - \gamma_k^2 \le 0$

If $f(d^*) \le 0$, then

$\nabla_d f^k(d) = \nabla f(x^k) + \nabla^2 f(x^k) d = 0$

if $(\nabla^2 f + \theta I) d = -\nabla f$

If $f(d^*) > 0$,

$H(d, \lambda) = f^k(d) + \lambda f(d)$

$= f(x^k) + \nabla f(x^k)^T d + \frac{1}{2} d^T \nabla^2 f(x^k) d + \lambda (d^T d - \gamma_k^2)$

$\nabla_d H = \nabla f + \nabla^2 f d + \lambda d = 0$

$\Rightarrow (\nabla^2 f + \lambda I) d = -\nabla f$

$\Rightarrow (\nabla^2 f + \lambda I) d = -\nabla f$

Solution $d = d^k$ of problem,

in other words d^k is

$$(\nabla^2 f + \delta^k I) d^k = -\nabla f$$

where we can think of δ^k as either

0 for a direct minimization of the problem

or a small perturbation to the direct minimization

to ensure that we satisfy the constraints,

since we know $\nabla^2 f$ is symmetric, it perturbs

the eigenvalues of $\nabla^2 f$ to conserve the

constraint.

$\because \nabla^2 f$ symmetric $\exists U \in O(n)$, D diagonal

s.t. $\nabla^2 f = U D U^T$, Then

$$(\nabla^2 f + \delta^k I) = (U D U^T + \delta^k I)$$

$$= U (D + \delta^k I) U^T$$

Small perturbations of eigenvalues of Hessian in order to maintain constraint.

```
%% prob_4ab.m
%
% this is a MATLAB script to perform HW1.4 a-b
%
% - written by: Dimitri Lezcano

%% Set-up
global N_max epsilon

N_max = 15000; % max # of iterations
epsilon = 0.005; % magnitude of dk to stop iterating
step_size = 0.001;
x_0 = [0; 0];

%% Perform gradient descent optimization
[x_grad, L_grad] = gradient_descent(@ cost_fn, @ dcost_fn, x_0, step_size);

%% perform the Newton method optimization

[x_new, L_new] = newton_method(@ cost_fn, @ dcost_fn, @hessian_cost_fn, x_0);

%% plot the results
% gradient descent
fgrad = figure(1);
subplot(2,1,1);
quiver(x_grad(1,:), x_grad(2,:), [diff(x_grad(1,:)), 0], [diff(x_grad(2,:)), 0],0, ...
    'Linewidth', 1.5); hold on
plot(x_grad(1,:), x_grad(2,:), 'o'); hold off;
xlabel('x1'); ylabel('x2');
title('trajectory');

subplot(2,1,2);
plot(L_grad,'Linewidth', 1.5)
xlabel('iteration'); ylabel('cost');
title('cost');

sgtitle(sprintf('4.a) gradient descent | %s = %.4f', '\alpha', step_size));

% Newton Method
fnew = figure(2);
subplot(2,1,1);
quiver(x_new(1,:), x_new(2,:), [diff(x_new(1,:)), 0], [diff(x_new(2,:)), 0],0, ...
    'Linewidth', 1.5); hold on;
plot(x_new(1,:), x_new(2,:), '.', 'markersize', 20); hold off;
% plot(x(1,:), x(2,:), '.'); hold off;
xlabel('x1'); ylabel('x2');
title('trajectory');

subplot(2,1,2);
plot(L_new, 'Linewidth', 1.5)
xlabel('iteration'); ylabel('cost');
title('cost');
```

```
sgtitle('4.a) Newton method');

%% saving
file_base = "hw1_prob_4ab";

saveas(fgrad, file_base + "_grad.png");
disp('Saved figure ' + file_base + "_grad.png");

saveas(fnew, file_base + "_newton.png");
disp('Saved figure ' + file_base + "_newton.png");

%% Functions
% % plotting function for trajectory
% function plot_traj(x1, x2)
%     for i = 1:length(x1)
%         quiver(
%
% end

% cost function for 4(a)
function L = cost_fn(x1, x2)
   L = (1 - x1).^2 + 200 * (x2 - x1.^2).^2;

end

% gradient for cost function of 4(a)
function dL = dcost_fn(x1, x2)
   dL = [ - 2 + 2*x1 - 800*(x1.*x2 - x1.^3);
         400* (x2 - x1.^2)];

end

% hessian for cost function of 4(a)
function d2L = hessian_cost_fn(x1, x2)
   d2L_x1_2 = 2 - 800 * (x2 - 3 * x1.^2);
   d2L_x2_2 = 400;
   d2L_x1_x2 = -800*x1;

   d2L = [d2L_x1_2, d2L_x1_x2;
         d2L_x1_x2, d2L_x2_2];

end

% the gradient descent returning a 2xN array of [x1; x2]
function [x_mat, L_mat] = gradient_descent(L, grad_L, x_0, step_size)
   global N_max epsilon

   % initialize the arrays to return
   x_mat = zeros(length(x_0), N_max);
   L_mat = zeros(N_max, 1); % array of losses

   % initial point
   x_mat(:,1) = x_0;
   L_mat(1) = L(x_0(1), x_0(2));
```

```
    for k = 2:N_max
        % determine the gradient
        grad_L_k = grad_L(x_mat(1,k-1), x_mat(2, k-1));

        % determine Dk matrix
        Dk = eye(length(grad_L_k));

        % determine dk, vector direction
        dk = Dk * grad_L_k;

        % take a step
        x_mat(:, k) = x_mat(:,k-1) - step_size * dk;

        % calculate the cost
        L_mat(k) = L(x_mat(1,k), x_mat(2,k));

        % check if we have converged
        if norm(dk) < epsilon
            break;
        end

    end

    % return where we stopped
    x_mat = x_mat(:,1:k);
    L_mat = L_mat(1:k);

end

% newton-method implementation
function [x_mat, L_mat] = newton_method(L, grad_L, hess_L, x_0)
    global N_max epsilon

    % initialize the arrays to return
    x_mat = zeros(length(x_0), N_max);
    L_mat = zeros(N_max, 1); % array of losses

    % initial point
    x_mat(:,1) = x_0;
    L_mat(1) = L(x_0(1), x_0(2));

    for k = 2:N_max
        % determine the gradient
        grad_L_k = grad_L(x_mat(1,k-1), x_mat(2, k-1));

        % determine Dk matrix
        inv_Dk = hess_L(x_mat(1,k-1), x_mat(2,k-1));

        % determine dk, vector direction
        dk = inv_Dk \ grad_L_k;

        % take a step
        x_mat(:, k) = x_mat(:,k-1) - dk;
```

```
        % calculate the cost
        L_mat(k) = L(x_mat(1,k), x_mat(2,k));

        % check if we have converged
        if norm(dk) < epsilon
            break;
        end

    end

    % return where we stopped
    x_mat = x_mat(:,1:k);
    L_mat = L_mat(1:k);

end
```

```
%% prob_4c.m
%
% this is a MATLAB script to perform HW1.4 c
%
% - written by: Dimitri Lezcano

%% Set-up
global N_max epsilon

N_max = 15000; % max # of iterations
epsilon = 0.005; % magnitude of dk to stop iterating
step_size = 0.001;
x_0 = [1.5; 3.0];

%% Perform gradient descent optimization
[x_grad, L_grad] = gradient_descent(@ cost_fn, @ dcost_fn, x_0, step_size);

%% perform the Newton method optimization

[x_new, L_new] = newton_method(@ cost_fn, @ dcost_fn, @hessian_cost_fn, x_0);

%% plot the results
% gradient descent
fgrad = figure(1);
subplot(2,1,1);
quiver(x_grad(1,:), x_grad(2,:), [diff(x_grad(1,:)), 0], [diff(x_grad(2,:)), 0],0, ...
    'Linewidth', 1.5); hold on
plot(x_grad(1,:), x_grad(2,:), 'o'); hold off;
xlabel('x1'); ylabel('x2');
title('trajectory');

subplot(2,1,2);
plot(L_grad,'Linewidth', 1.5)
xlabel('iteration'); ylabel('cost');
title('cost');

sgtitle(sprintf('4.a) gradient descent | %s = %.4f', '\alpha', step_size));

% Newton Method
fnew = figure(2);
subplot(2,1,1);
quiver(x_new(1,:), x_new(2,:), [diff(x_new(1,:)), 0], [diff(x_new(2,:)), 0],0, ...
    'Linewidth', 1.5); hold on;
plot(x_new(1,:), x_new(2,:), '.', 'markersize', 20); hold off;
% plot(x(1,:), x(2,:), '.'); hold off;
xlabel('x1'); ylabel('x2');
title('trajectory');

subplot(2,1,2);
plot(L_new, 'Linewidth', 1.5)
xlabel('iteration'); ylabel('cost');
title('cost');
```

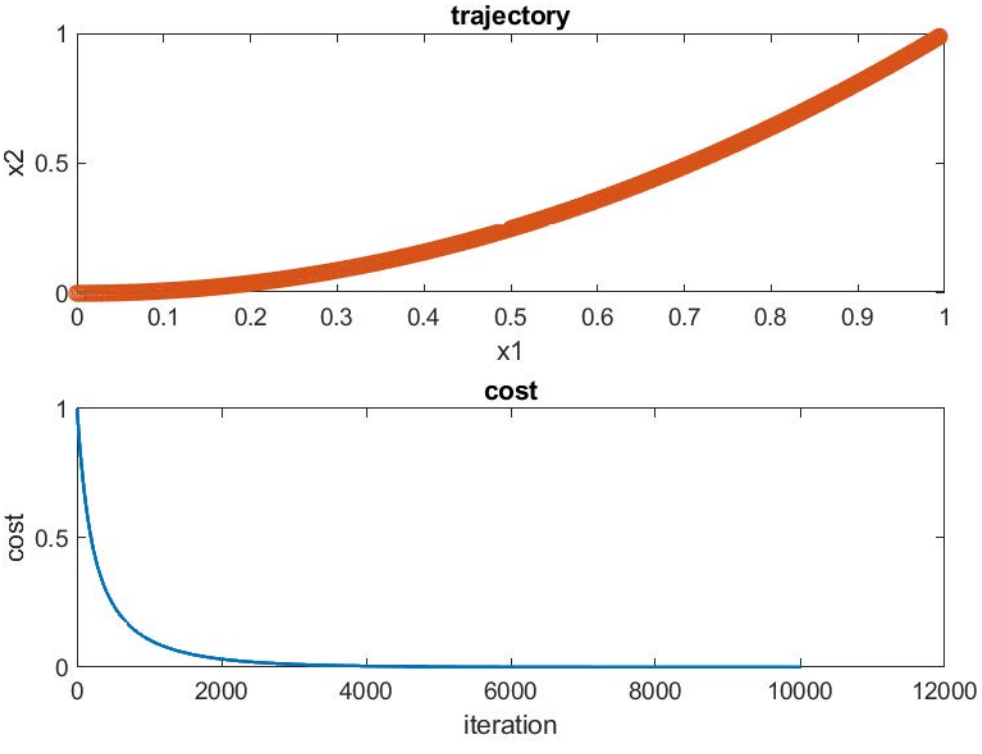
```
sgtitle('4.a) Newton method');

%% saving
file_base = "hw1_prob_4c";

saveas(fgrad, file_base + "_grad.png");
disp('Saved figure ' + file_base + "_grad.png");

saveas(fnew, file_base + "_newton.png");
disp('Saved figure ' + file_base + "_newton.png");

%% Functions
% % plotting function for trajectory
% function plot_traj(x1, x2)
%     for i = 1:length(x1)
%         quiver(
%
% end

% cost function for 4(a)
function L = cost_fn(x1, x2)
%     L = (1 - x1)..^2 + 200 .* (x2 - x1..^2)..^2;
    L = x1 .* exp(-x1.^2 - x2.^2/2) + x1.^2/10 + x2.^2/10;

end

% gradient for cost function of 4(a)
function dL = dcost_fn(x1, x2)
    dL = [ x1/5 + exp(- x1.^2 - x2.^2/2) - 2.*x1.^2.*exp(- x1.^2 - x2.^2/2);
          x2/5 - x1.*x2.*exp(- x1.^2 - x2.^2/2)];

end

% hessian for cost function of 4(a)
function d2L = hessian_cost_fn(x1, x2)
    d2L_x1_2 = 4.*x1.^3.*exp(- x1.^2 - x2.^2/2) - 6.*x1.*exp(- x1.^2 - x2.^2/2) + 1/5;
    d2L_x2_2 = x1.*x2.^2.*exp(- x1.^2 - x2.^2/2) - x1.*exp(- x1.^2 - x2.^2/2) + 1/5;
    d2L_x1_x2 = x2.*exp(- x1.^2 - x2.^2/2).*(2.*x1.^2 - 1);

    d2L = [d2L_x1_2, d2L_x1_x2;
          d2L_x1_x2, d2L_x2_2];

end

% the gradient descent returning a 2xN array of [x1; x2]
function [x_mat, L_mat] = gradient_descent(L, grad_L, x_0, step_size)
    global N_max epsilon

    % initialize the arrays to return
    x_mat = zeros(length(x_0), N_max);
    L_mat = zeros(N_max, 1); % array of losses

    % initial point
    x_mat(:,1) = x_0;
```

```
    L_mat(1) = L(x_0(1), x_0(2));

    for k = 2:N_max
        % determine the gradient
        grad_L_k = grad_L(x_mat(1,k-1), x_mat(2, k-1));

        % determine Dk matrix
        Dk = eye(length(grad_L_k));

        % determine dk, vector direction
        dk = Dk * grad_L_k;

        % take a step
        x_mat(:, k) = x_mat(:,k-1) - step_size .* dk;

        % calculate the cost
        L_mat(k) = L(x_mat(1,k), x_mat(2,k));

        % check if we have converged
        if norm(dk) < epsilon
            break;
        end

    end

    % return where we stopped
    x_mat = x_mat(:,1:k);
    L_mat = L_mat(1:k);

end

% newton-method implementation
function [x_mat, L_mat] = newton_method(L, grad_L, hess_L, x_0)
    global N_max epsilon

    % initialize the arrays to return
    x_mat = zeros(length(x_0), N_max);
    L_mat = zeros(N_max, 1); % array of losses

    % initial point
    x_mat(:,1) = x_0;
    L_mat(1) = L(x_0(1), x_0(2));

    for k = 2:N_max
        % determine the gradient
        grad_L_k = grad_L(x_mat(1,k-1), x_mat(2, k-1));

        % determine Dk matrix
        inv_Dk = hess_L(x_mat(1,k-1), x_mat(2,k-1));

        % determine dk, vector direction
        dk = inv_Dk \ grad_L_k;

        % take a step
```

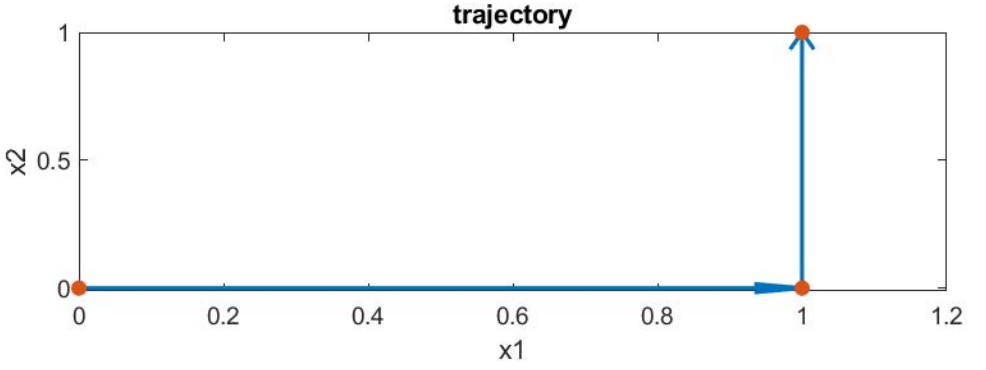
```
        x_mat(:, k) = x_mat(:,k-1) - dk;

        % calculate the cost
        L_mat(k) = L(x_mat(1,k), x_mat(2,k));

        % check if we have converged
        if norm(dk) < epsilon
            break;
        end

    end

    % return where we stopped
    x_mat = x_mat(:,1:k);
    L_mat = L_mat(1:k);

end
```

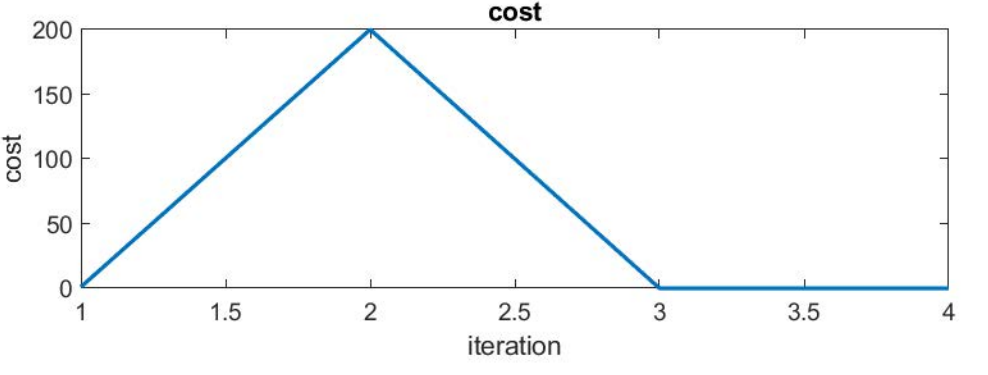
```
%% prob_4d.m
%
% This is a script to verify prob 4d's optimization
%
% - written by: Dimitri Lezcano

%% Set-up
% the cost function and the constraints
cost_fn = @(y) y(1).^2 + 4 * y(1) + 8 * y(2).^2;

% constraint A * y <= b | [1 -7]*[x; u] <= -5
A = [1, -7];
b = -5;

x_0 = [0; 1];

%% My optimal solution
y_optim_mine = [-2.4211; 0.3684];

%% using fmincon to get optimized value

y_optim = fmincon(cost_fn, x_0, A, b);

fprintf('   mine: x* = %.5f, u* = %.5f\n', y_optim_mine);
fprintf('fmincon: x* = %.5f, u* = %.5f\n', y_optim);

%% write the output file
file_out = "hw1_prob_4d.txt";
fileID = fopen(file_out, 'w');
fprintf(fileID, '   mine: x* = %.5f, u* = %.5f\n', y_optim_mine);
fprintf(fileID, 'fmincon: x* = %.5f, u* = %.5f\n', y_optim);
fclose(fileID);
disp("Wrote file:" + file_out)
```

4.a) gradient descent | α = 0.0010

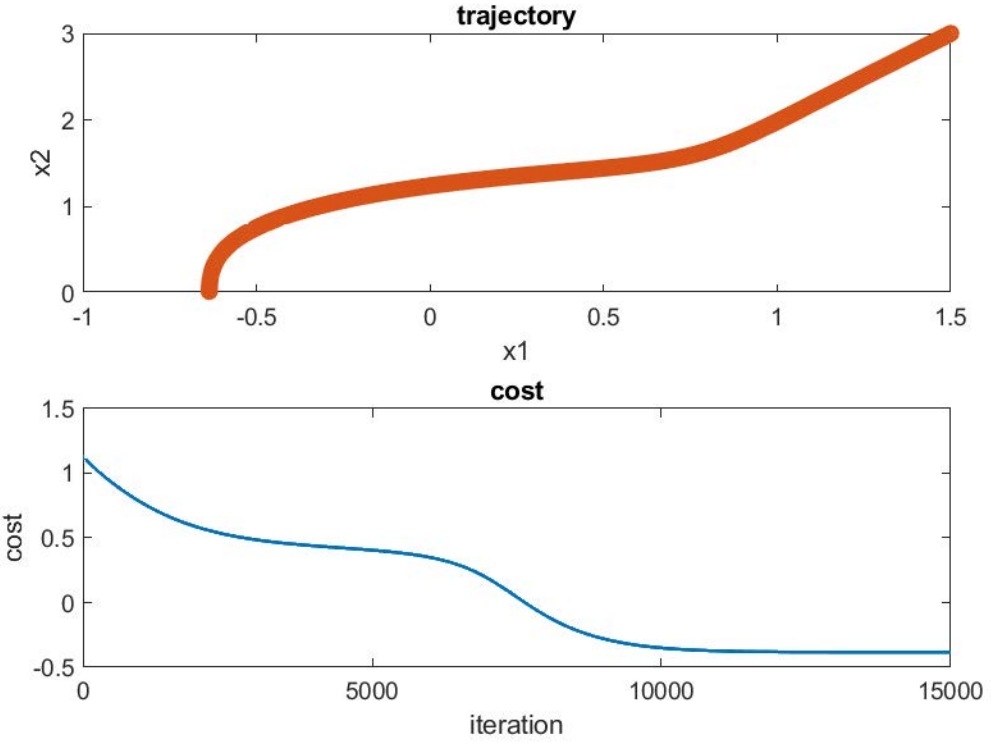


4.a) Newton method

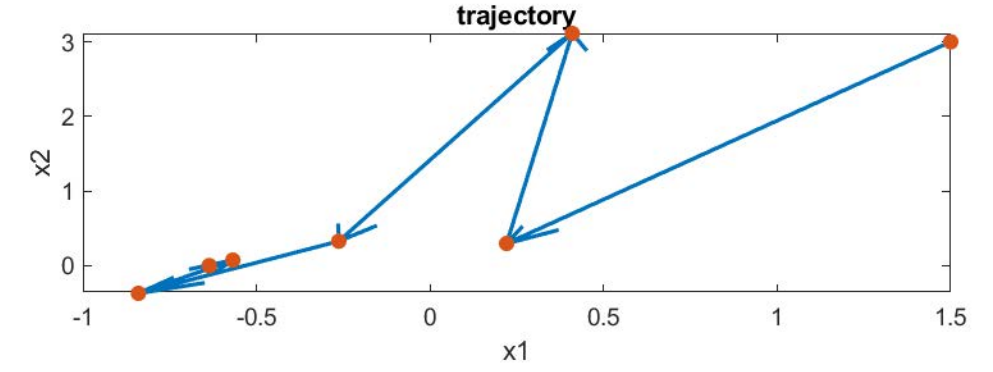


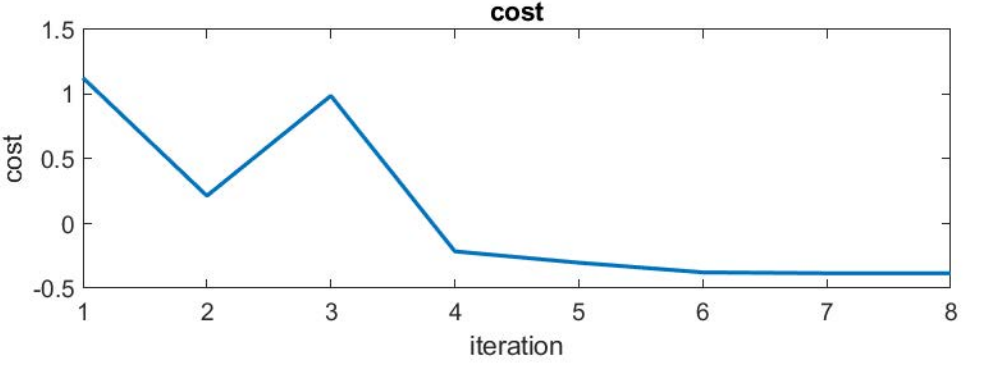


4.a) gradient descent | α = 0.0010



4.a) Newton method





Problem 4d output:
   mine: x* = -2.42110, u* = 0.36840
fmincon: x* = -2.42105, u* = 0.36842