1 Question 1

(4 points)

Recall Equation (9.6) in the notes.

Let $B = Q\Lambda Q^T$ be the eigenvalue decomposition of B where Q is orthonormal and

$$\Lambda = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_n)$$

with $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$ being the eigenvalues of B.

Let d^* be given by

$$d^* = -\sum_{j:\lambda_j \neq \lambda_1} \frac{q_j^T b}{\lambda_j - \lambda_1} q_j + \tau q_1 \tag{1.1}$$

where τ is computed from

$$\Delta^2 = \sum_{j:\lambda_j \neq \lambda_1} \frac{(q_j^T b)^2}{(\lambda_j - \lambda_1)^2} + \tau^2$$

Verify

$$(B - \lambda_1 I)d^* = -b$$

Proof:

Let $Q = [q_1 \quad \cdots \quad q_n]$, where $q_1, ..., q_n \in R^n$, then $Q^{-1} = Q^T = \begin{bmatrix} q_1^T \\ \vdots \\ q_n^T \end{bmatrix}$ since Q is orthonormal. First, we show B is symmetric:

$$B^T = (Q\Lambda Q^T)^T = Q\Lambda^T Q^T$$

Since Λ is a diagonal matrix, $\Lambda^T = \Lambda$, so $B^T = Q\Lambda^T Q^T = Q\Lambda Q^T = B$. Then,

$$\begin{aligned}
(B - \lambda_1 I)d^* &= (Q\Lambda Q^T - \lambda_1 I)d^* = (Q\Lambda Q^T - \lambda_1 QQ^T)d^* = (Q\Lambda Q^T - Q\lambda_1 I Q^T)d^* \\
&= Q(\Lambda - \lambda_1 I)Q^T d^* \\
&= \left([q_1 \quad \cdots \quad q_n] \left(\begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} - \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_1 \end{bmatrix} \right) \begin{bmatrix} q_1^T \\ \vdots \\ q_n^T \end{bmatrix} \right) d^* \\
&= \left([q_1 \quad \cdots \quad q_n] \begin{bmatrix} \lambda_1 - \lambda_1 & & \\ & \ddots & \\ & & \lambda_n - \lambda_1 \end{bmatrix} \begin{bmatrix} q_1^T \\ \vdots \\ q_n^T \end{bmatrix} \right) d^* \\
&= \left(\sum_{i=1}^{n} (\lambda_i - \lambda_1) q_i q_i^T \right) d^* \\
&= -\left(\sum_{i=1}^{n} (\lambda_i - \lambda_1) q_i q_i^T \right) \left(\sum_{j:\lambda_j \neq \lambda_1}^{n} \frac{q_j^T b}{\lambda_j - \lambda_1} q_j + \tau q_1 \right)
\end{aligned}$$

Since for any i such that $\lambda_i = \lambda_1$, $(\lambda_i - \lambda_1)q_i q_i^T = (\lambda_1 - \lambda_1)q_i q_i^T = 0$, we have

$$\sum_{i=1}^{n} (\lambda_i - \lambda_1) q_i q_i^T = \sum_{i:\lambda_i \neq \lambda_1}^{n} (\lambda_i - \lambda_1) q_i q_i^T$$

Hence,

$$(B - \lambda_1 I)d^* = -\left(\sum_{i:\lambda_i \neq \lambda_1}^{n} (\lambda_i - \lambda_1) q_i q_i^T \right) \left(\sum_{j:\lambda_j \neq \lambda_1}^{n} \frac{q_j^T b}{\lambda_j - \lambda_1} q_j + \tau q_1 \right)$$

Note that for any i, j such that i $\neq$ j and $\lambda_i \neq \lambda_1, \lambda_j \neq \lambda_1$, we have $[(\lambda_i - \lambda_1)q_iq_i^T]\left(\frac{q_j^Tb}{\lambda_j-\lambda_1}q_j + \tau q_1\right) = (\lambda_i - \lambda_1)q_iq_i^T\frac{q_j^Tb}{\lambda_j-\lambda_1}q_j + (\lambda_i - \lambda_1)q_iq_i^T\tau q_1 = \frac{q_j^Tb}{\lambda_j-\lambda_1}(\lambda_i - \lambda_1)q_i(q_i^Tq_j) + \tau(\lambda_i - \lambda_1)q_i(q_i^Tq_1)$

Since Q is orthonormal, all columns of Q are orthogonal to each other, which means $q_i^Tq_j = 0$ and $q_i^Tq_1 = 0$. Therefore,

$$[(\lambda_i - \lambda_1)q_iq_i^T]\left(\frac{q_j^Tb}{\lambda_j - \lambda_1}q_j + \tau q_1\right)$$

$$= \frac{q_j^Tb}{\lambda_j - \lambda_1}(\lambda_i - \lambda_1)q_i(q_i^Tq_j) + \tau(\lambda_i - \lambda_1)q_i(q_i^Tq_1) = 0$$

Hence,

$$\begin{aligned}(B - \lambda_1 I)d^* &= -\left(\sum_{i:\lambda_i\neq\lambda_1}^{n}(\lambda_i - \lambda_1)q_iq_i^T\right)\left(\sum_{j:\lambda_j\neq\lambda_1}^{n}\frac{q_j^Tb}{\lambda_j - \lambda_1}q_j + \tau q_1\right)\\ &= -\sum_{i:\lambda_i\neq\lambda_1}^{n}[(\lambda_i - \lambda_1)q_iq_i^T]\left(\frac{q_i^Tb}{\lambda_i - \lambda_1}q_i + \tau q_1\right)\\ &= -\sum_{i:\lambda_i\neq\lambda_1}^{n}(\lambda_i - \lambda_1)\frac{q_i^Tb}{\lambda_i - \lambda_1}q_iq_i^Tq_i + [(\lambda_i - \lambda_1)q_iq_i^T]\tau q_1\\ &= -\sum_{i:\lambda_i\neq\lambda_1}^{n}q_i^Tbq_iq_i^Tq_i + \tau(\lambda_i - \lambda_1)q_iq_i^Tq_1\\ &= -\sum_{i:\lambda_i\neq\lambda_1}^{n}q_i^Tbq_i\|q_i\|_2^2 + \tau(\lambda_i - \lambda_1)q_i(q_i^Tq_1)\end{aligned}$$

Since Q is orthonormal, $\|q_i\| = 1$ for any $i = 1, \ldots, n$. Also, since $i \neq 1$, $q_i^Tq_1 = 0$. Thus,

$$(B - \lambda_1 I)d^* = -\sum_{i:\lambda_i\neq\lambda_1}^{n}q_i^Tbq_i\|q_i\|_2^2 + \tau(\lambda_i - \lambda_1)q_i(q_i^Tq_1) = -\sum_{i:\lambda_i\neq\lambda_1}^{n}q_i^Tbq_i$$

Note that by assumption, $q_k^Tb = 0, \forall k\ s.t. \lambda_k = \lambda_i$, so $\sum_{i:\lambda_i=\lambda_1}^{n}q_i^Tbq_i = 0$, which means

$$(B - \lambda_1 I)d^* = -\sum_{i:\lambda_i\neq\lambda_1}^{n}q_i^Tbq_i - 0 = -\sum_{i:\lambda_i\neq\lambda_1}^{n}q_i^Tbq_i - \sum_{i:\lambda_i=\lambda_1}^{n}q_i^Tbq_i = -\sum_{i=1}^{n}q_i^Tbq_i$$

Since $q_1, \ldots, q_n$ are orthogonal eigenvectors of B, which is a symmetric matrix, b can be represented as a linear combination of $\{q_1, \ldots, q_n\}$. That is,

$$\exists \alpha_1, \ldots, \alpha_n \in R, s.t. b = \sum_{i=1}^{n} \alpha_i q_i$$

Hence, for any $j = 1, \ldots, n$, we all have

$$q_j^T b q_j = q_j^T \left(\sum_{i=1}^{n} \alpha_i q_i \right) q_j = \sum_{i=1}^{n} q_j^T \alpha_i q_i q_j = \sum_{i=1}^{n} \alpha_i q_j^T q_i q_j$$

Since $q_j^T q_i = \begin{cases} 0, i \neq j \\ 1, i = j \end{cases}$ as we showed, we have

$$q_j^T b q_j = \sum_{i=1}^{n} \alpha_i q_j^T q_i q_j = \alpha_j q_j$$

Therefore,

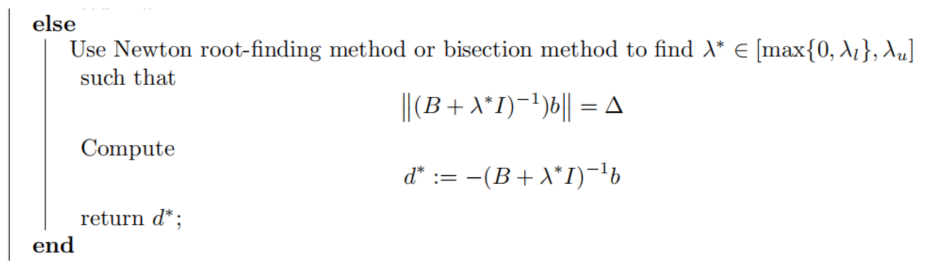
$$(B - \lambda_1 I) d^* = -\sum_{i=1}^{n} q_i^T b q_i = -\sum_{i=1}^{n} \alpha_i q_i = -b$$

2 Question 2

[Coding](8 points)

1. Let $b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$ and $B = \begin{pmatrix} -2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ in the trust region subproblem. Compute the optimal λ^* when $\Delta = 2$ (Your solutions only need to be accurate to 2 decimal places)
2. Let $b = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$ and $B = \begin{pmatrix} -2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ in the trust region subproblem. Compute the optimal λ^* and d^* when $\Delta = 1$ (Your solutions only need to be accurate to 2 decimal places)

Solution:
We can implement Algorithm3 in 9.2 in the lecture note on MATLAB. Note that in the below part, where we have to implement Newton root finding method:



```
else
    Use Newton root-finding method or bisection method to find $\lambda^* \in [\max\{0, \lambda_l\}, \lambda_u]$
    such that
        $\|(B + \lambda^* I)^{-1})b\| = \Delta$
    Compute
        $d^* := -(B + \lambda^* I)^{-1} b$
    return $d^*$;
end
```

We use the iterative method $\lambda^{n+1} = \lambda^n - \frac{f(\lambda^n)}{f'(\lambda^n)}$, where $f(\lambda) = \|(B + \lambda I)^{-1} b\| - \Delta$

The derivative $f'(\lambda) = (\|(B + \lambda I)^{-1} b\| - \Delta)' = -\frac{b^T[(B+\lambda I)^{-1}]^T (B+\lambda I)^{-2} b}{\|(B+\lambda I)^{-1} b\|}$ is calculated by a matrix calculus tool, which is hardcoded into my code.

Then, the result is $r^* = 2.5465$ for question 1, and $r^* = 2.1991$, $d^* = \begin{bmatrix} 0 \\ -0.8340 \\ -0.4547 \\ -0.3126 \end{bmatrix}$

for question 2.

```
function d = subproblem(B, b, del)
% first, do eigendecomposition of B
lambd = eig(B);  % get all eigenvalues
lambdl = min(lambd);  % get the minimum eigenvalue
% see where B is pd by checking its eigenvalues
if lambdl > 0
    % B is pd and d has norm less than delta, then B has inverse
    d = -inv(B) * b;  % calculate d
    if norm(b) < del
        % d has norm less than delta, then d is optimal, and we are done
        return;
    end
end
% get the eigenvector v1 of lambda1
[V,D] = eig(B);
n = size(B);
n = n(1);  % n is the size of B
% initialize a matrix to store ALL eigenvectors that has eigenvalue
%   lambda1
Lbdleigv = zeros(n, n);
for i = 1: n
     if D(i, i) == lambdl
         Lbdleigv(:, i) = V(:, i);
     end
end
% see if ALL such eigenvectors have dot product 0 with b
check_vec = (b' * Lbdleigv)';
if norm(check_vec) == 0
    % ALL such eigenvectors have dot product 0 with b
    % calculate -d(-lambda1)
    d_lbdl = 0.0;
    for j = 1: n
        if D(j, j) ~= lambdl
             d_lbdl = d_lbdl + ((V(:, j)' * b) / (D(j, j) - lambdl)) * V(:, i);
        end
    end

    if norm(d_lbdl) <= del
        tau = sqrt(del.^2 - norm(d_lbdl).^2);
        d = -d_lbdl + tau * V(:, 1);  % d is the optimal solution
        return;
    end
end
% calculate lower and upper bound
lower_sig_term = 0.0;
for j = 1: n
    if D(j, j) == lambdl
         lower_sig_term = lower_sig_term + (V(:, j)' * b).^2;
    end
end
lower_sig_term = lower_sig_term.^(1/2);
lbd_lower = -lambdl + (1 / del) * lower_sig_term;

lbd_upper = norm(b) / del - lambdl;

lbd_lower = max(lbd_lower, 0);

% implement newton's method to find root
err = 0.0000001;  % the max error of the root
lbd_st = (lbd_upper + lbd_lower) / 2;  % initial guess is between upper and lower bound
lbd_st = lbd_upper;
Idtt = eye(n);
dist_to_x = norm(inv(B + lbd_st * Idtt) * b) - del; % distance to the x-axis
while abs(dist_to_x) > err
    % get the derivative of the function
    deri = - (b' * inv(B + lbd_st * Idtt)' * inv(B + lbd_st * Idtt).^2 * b) / norm(inv(B + lbd_st ↙
Idtt) * b);
    % update lambda^*
    lbd_st = lbd_st - dist_to_x / deri;

    % update distance to the x-axis
    dist_to_x = norm(inv(B + lbd_st * Idtt) * b) - del;
end
% get the optimal d
d = -inv(B + lbd_st * Idtt) * b;
return
end
```

Script.1 My implementation of the subproblem algorithm

3. Implement the trust region method or use any built in packages for trust region methods (matlab or Python or other languages) to find the local minimizer of the following function in the interval $x, y \in [-5, 5]$:

$$f(x,y) = -5x^2 + 4y^2 + 6\sin(xy) - 2x + x^4;$$

(You get 2 bonus points if you implement your own trust region method, your solutions only need to be accurate to 1 decimal place.

If you use matlab, you may use *fminunc* function link. If you use python, you can use the Python optimization package link.)

Solution:
First, we use the built-in function hessian(f) and gradient(f) to get the hessian matrix and the gradient vector of $f(x,y)$, and then implement Algorithm 2 in the lecture note with the help from the subproblem solver I implemented in Q2.1.
Here is the code:

```
function x = TrustRegion(f, vars,max_Del, ini_Del, x_ini, rou_1, rou_2, gm_1, gm_2, yt, max_iter)
x = x_ini;
del = min(ini_Del, max_Del);
% solve for the hessian matrix and gradient vector of f
B = hessian(f);
b = gradient(f);

% set the stopping criterion to be that #iterations reaches max_iter
iter = 0;
while iter < max_iter
    b_x = double(subs(b, vars, x')); % current gradient b
    B_x = double(subs(B, vars, x')); % current hessian B

    d = subproblem(B_x, b_x, del); % solve the subproblem
    disp(iter);
    f_x_k = double(subs(f, vars, x')); % f(x^k)
    f_x_k_1 = double(subs(f, vars, (x + d)')); % f(x^(k+1))

    m_k_0 = f_x_k; % compute m_k(0)
    m_k_d = f_x_k + b_x' * d + (1/2) * d' * B_x * d; % compute m_k(d)

    % compute the measurement
    rou_k = (f_x_k - f_x_k_1) / (m_k_0 - m_k_d);

    % update the radius del
    if rou_k < rou_1
        del = gm_1 * del;
    elseif rou_k > rou_2 && norm(d) == del
        del = min(gm_2 * del, max_Del);
    end

    % check if we need to update x
    if rou_k > yt
        x = x + d;
    end
    iter = iter + 1;
    disp(x);
end
```

Script.2: My implementation of the Trust Region algorithm

According to the result of my script, the local minimizer is $x_{mine} = \begin{bmatrix} 1.7432 \\ -0.6184 \end{bmatrix}$ and the parameters in the function call is:
$\text{x\_mine} =$
$\text{TrustRegion}(f, [x, y], 0.001, 0.0001, [1, 1]', 0.25, 0.75, 0.25, 2, 0.05, 100000);$
According to the built-in MATLAB package, the local minimizer is $x_{MATLAB} = \begin{bmatrix} 1.7432 \\ -0.6184 \end{bmatrix}$ after executing script.3.

```
fun = @(x) (-5)*x(1)^2 + 4*x(2)^2 + 6*sin(x(1)*x(2)) - 2*x(1) + x(1)^4;
[x,fval] = fminunc(fun,[1,1]);
```

Script.3 Built-in MATLAB package call

We don't see obvious difference between the result from my implementation and the result from the MATLAB built-in package. And the corresponding values of the objective function based on the two solutions are both around -13.2025.

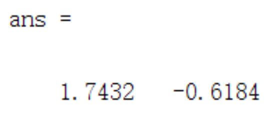
4. Implement a Newton's method to find the local minimizer of the above function, find an initial point such that the trust region method converges but Newton method fails to converge to one of the local minimizers. (Your initial point doesn't have to be in the interval $[-5, 5]$, your solutions only need to be accurate to 1 decimal place.)

Solution:
First, we implement Newton's method in MATLAB trivially:

```
function x = Newton(f, vars, max_iter, x_ini)
x = x_ini;

H = hessian(f);  % get the Hessian matrix of f
g = gradient(f);  % get the gradient vector of f

% start iteration until reaches max_iter
iter = 0;
while iter < max_iter
    disp(iter);
    % get the current Hessian matrix and gradient vector of f
    H_x = double(subs(H, vars, x'));
    g_x = double(subs(g, vars, x'));

    % get the inverse of H to get d
    d = -inv(H_x) * g_x;

    % update x
    x = x + d;

    iter = iter + 1;
end

end
```

Script.4: My implementation of Newton's method

Then, we pick up $x_0 = \begin{bmatrix} 2.5 \\ -1 \end{bmatrix}$ as the initial point, which is close enough to the local minimizer. Then, we successfully get the local minimizer $x^* = \begin{bmatrix} 1.7432 \\ -0.6184 \end{bmatrix}$. This is the correct local minimizer according to part 3. Then, we pick up $x_0 = \begin{bmatrix} 4 \\ -4 \end{bmatrix}$ as the initial point. The algorithm does not converge, but the trust region method converges based on this initial point.



```
ans =

    1.7432   -0.6184
```

Pic.1 Convergence of the trust region method

3 Question 3

(6 points) Consider the following constraint optimization problem

$$\begin{aligned} \min_{x,y\in\mathbb{R}} \quad & x \\ \text{s. t.} \quad & 16-(x-4)^2-y^2 \geq 0 \\ & x^2+(y-2)^2-4=0 \end{aligned}$$

Compute all its KKT points, and determine if they are local minimizer, saddle point or global minimizer.

Solution:

First, we rearrange the NLP to

$$\min_{x,y\in R} x$$

$$s.t. x^2+(y-2)^2-4=0$$
$$(x-4)^2+y^2-16\leq 0$$

The Lagrangian function is

$$L(x,y,\lambda_1,\lambda_2)=x+\lambda_1(x^2+(y-2)^2-4)+\lambda_2((x-4)^2+y^2-16)$$

By the stationary condition,

$$\begin{bmatrix}1\\0\end{bmatrix}+2\lambda_1\begin{bmatrix}x\\y-2\end{bmatrix}+2\lambda_2\begin{bmatrix}x-4\\y\end{bmatrix}=0$$

By the complementarity condition for λ_1:

$$\lambda_1(x^2+(y-2)^2-4)=0$$

When $\lambda_1=0$, the stationary condition becomes

$$\begin{bmatrix}1\\0\end{bmatrix}+2\lambda_2\begin{bmatrix}x-4\\y\end{bmatrix}=0$$

$$\begin{cases}1+2\lambda_2(x-4)=0 \ (1)\\ 2\lambda_2 y=0 \ (2)\end{cases}$$

If $y=0$, then by the second constraint,

$$(x-4)^2-16\leq 0$$
$$(x-4)^2\leq 16$$
$$-4\leq x-4\leq 4$$
$$0\leq x\leq 8$$

By the first constraint,

$$x^2+4-4=0$$
$$x=0$$

Then, we substitute it back to (1),

$$1+2\lambda_2(0-4)=0$$
$$\lambda_2=\frac{1}{8}\geq 0$$

Hence, $(0,0)$ is a KKT point with $(\lambda_1,\lambda_2)=\left(0,\frac{1}{8}\right)$.

By the complementarity condition for λ_2:

$$\lambda_2((x-4)^2+y^2-16)=0$$

If $\lambda_2=0$, then (1) does not hold.

When $\lambda_2 = 0$, the stationary condition becomes

$$\begin{bmatrix}1\\0\end{bmatrix} + 2\lambda_1 \begin{bmatrix}x\\y-2\end{bmatrix} = 0$$

$$\begin{cases} 1 + 2\lambda_1 x = 0 \ (1) \\ 2\lambda_1(y-2) = 0 \ (2) \end{cases}$$

From (2), we can see $\lambda_1 = 0$ or $y - 2 = 0$. But when $\lambda_1 = 0$, (1) does not hold, so we only say $y = 2$. Then, substitute it back to the first constraint,

$$x^2 - 4 = 0$$

$$x = \pm 2$$

However, when $x = -2$, the second constraint becomes $36 + 4 - 16 > 0$, which is violated. So, we can only keep $x = 2$. Then, we substitute it back to (1) to get

$$1 + 2\lambda_1 x = 0$$

$$\lambda_1 = -\frac{1}{4}$$

Thus, $(2, 2)$ is a KKT point with $(\lambda_1, \lambda_2) = \left(-\frac{1}{4}, 0\right)$.

When $\lambda_1, \lambda_2 \neq 0$, the complementarity conditions gives

$$x^2 + (y-2)^2 - 4 = (x-4)^2 + y^2 - 16 = 0$$

$$x^2 + y^2 - 4y + 4 - 4 = x^2 - 8x + 16 + y^2 - 16 = 0$$

$$-4y = -8x$$

$$y = 2x$$

Substitute it back to the first equation gives

$$x^2 + (2x-2)^2 - 4 = 0$$

$$x^2 + 4x^2 - 8x + 4 - 4 = 0$$

$$x(5x - 8) = 0$$

$$x = 0 \ or \ x = \frac{8}{5}$$

When $x = 0$, y = 2x = 0, we already discussed it, so we can get rid of it.

When $x = \frac{8}{5}, y = 2x = \frac{16}{5}$, the stationary condition becomes

$$\begin{bmatrix}1\\0\end{bmatrix} + 2\lambda_1 \begin{bmatrix}\frac{8}{5}\\\frac{6}{5}\end{bmatrix} + 2\lambda_2 \begin{bmatrix}-\frac{12}{5}\\\frac{16}{5}\end{bmatrix} = 0$$

$$\begin{cases} \frac{16}{5}\lambda_1 - \frac{24}{5}\lambda_2 = -1 \\ \frac{12}{5}\lambda_1 + \frac{32}{5}\lambda_2 = 0 \end{cases}$$

$$\begin{bmatrix}16 & -24 & -5\\12 & 32 & 0\end{bmatrix}$$

$$\lambda_1 = -\frac{1}{5}, \lambda_2 = \frac{3}{40} \geq 0$$

Therefore, $\left(\frac{8}{5}, \frac{16}{5}\right)$ is a KKT point with $(\lambda_1, \lambda_2) = \left(-\frac{1}{5}, \frac{3}{40}\right)$.

Overall, we have three KKT points: $(0,0), (2,2)$, and $\left(\frac{8}{5}, \frac{16}{5}\right)$.

The gradients of the constraints are:

$$\nabla c_1(x) = \begin{bmatrix} 2x \\ 2(y-2) \end{bmatrix} = 2\begin{bmatrix} x \\ y-2 \end{bmatrix}$$

$$\nabla c_2(x) = \begin{bmatrix} 2(x-4) \\ 2y \end{bmatrix} = 2\begin{bmatrix} x-4 \\ y \end{bmatrix}$$

Then, for $(0,0)$, the active constraints are both c_1 and c_2, whose gradients are

$$\nabla c_1(0,0) = \begin{bmatrix} 0 \\ -4 \end{bmatrix}$$

$$\nabla c_2(0,0) = \begin{bmatrix} -8 \\ 0 \end{bmatrix}$$

Since they are orthogonal, they are not linearly dependent. Hence, LICQ holds, we say $T_\Omega(0,0) = F(0,0)$. Then, we just have to get the linearized feasible direction set. Let $d = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \in F(0,0)$, then

$$d^T \nabla c_1(0,0) = -4d_2 = 0$$
$$d_2 = 0$$
$$d^T \nabla c_2(0,0) = -8d_1 \le 0$$
$$d_1 \ge 0$$

Hence, $T_\Omega(0,0) = F(0,0) = \{(d_1, 0) | d_1 \ge 0\}$.

Then, we get the critical cone $C\left((0,0), \left(0, \frac{1}{8}\right)\right)$:

$$d^T \nabla c_2(0,0) = -8d_1 = 0$$
$$d_1 = 0$$

Hence, $C\left((0,0), \left(0, \frac{1}{8}\right)\right) = \{(0,0)\}$, which means $(0,0)$ is a local minimizer by Theorem 10.15.

For $(2,2)$, the only active constraint is c_1, whose gradient is

$$\nabla c_1(2,2) = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

Since we only have this one active constraint, LICQ holds, so $T_\Omega(2,2) = F(2,2)$. Let $d = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \in F(2,2)$, then

$$d^T \nabla c_1(2,2) = 4d_1 = 0$$
$$d_1 = 0$$

Hence, $T_\Omega(2,2) = F(2,2) = \{(0, d_2) | d_2 \in R\}$. Then, since it does not have any active inequality constraint, $C\left((2,2), \left(-\frac{1}{4}, 0\right)\right) = F(2,2) = \{(0, d_2) | d_2 \in R\}$. Then, we compute $\nabla_{xx}^2 L\left((2,2), \left(-\frac{1}{4}, 0\right)\right)$ by:

$$L(x, y, \lambda_1, \lambda_2) = x + \lambda_1(x^2 + (y-2)^2 - 4) + \lambda_2((x-4)^2 + y^2 - 16)$$

$$\nabla_x L(x, y, \lambda_1, \lambda_2) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \lambda_1 \begin{bmatrix} 2x \\ 2(y-2) \end{bmatrix} + \lambda_2 \begin{bmatrix} 2(x-4) \\ 2y \end{bmatrix}$$

$$\nabla^2_{xx}L(x,y,\lambda_1,\lambda_2) = \begin{bmatrix} 2\lambda_1 + 2\lambda_2 & 0 \\ 0 & 2\lambda_1 + 2\lambda_2 \end{bmatrix} = 2(\lambda_1 + \lambda_2)\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\nabla^2_{xx}L\left((2,2),\left(-\frac{1}{4},0\right)\right) = -\frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Let $d = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \in C\left((2,2),\left(-\frac{1}{4},0\right)\right)$, then $d^T\nabla^2_{xx}L\left((2,2),\left(-\frac{1}{4},0\right)\right)d = -\frac{1}{2}\begin{bmatrix} 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = -\frac{1}{2} < 0$, so it's a saddle point according to Theorem 10.14.

For $\left(\frac{8}{5},\frac{16}{5}\right)$, the gradient of the active constraints are

$$\nabla c_1\left(\frac{8}{5},\frac{16}{5}\right) = \begin{bmatrix} 2x \\ 2(y-2) \end{bmatrix} = \begin{bmatrix} \frac{16}{5} \\ \frac{12}{5} \end{bmatrix}$$

$$\nabla c_2\left(\frac{8}{5},\frac{16}{5}\right) = \begin{bmatrix} 2(x-4) \\ 2y \end{bmatrix} = \begin{bmatrix} -\frac{24}{5} \\ \frac{32}{5} \end{bmatrix}$$

Then, since they are not parallel, they are linearly independent. Hence, LICQ holds, which means $T_\Omega\left(\frac{8}{5},\frac{16}{5}\right) = F\left(\frac{8}{5},\frac{16}{5}\right)$. Then, for any $d \in F\left(\frac{8}{5},\frac{16}{5}\right)$,

$$d^T\nabla c_1\left(\frac{8}{5},\frac{16}{5}\right) = \frac{16}{5}d_1 + \frac{12}{5}d_2 = 0$$
$$4d_1 + 3d_2 = 0$$
$$d_1 = -\frac{3}{4}d_2(*)$$
$$dd^T\nabla c_2\left(\frac{8}{5},\frac{16}{5}\right) = -\frac{24}{5}d_1 + \frac{32}{5}d_2 = 0$$
$$d_1 = \frac{4}{3}d_2(**)$$

Combine (*) and (**), we have

$$d_1 = -\frac{3}{4}d_2 = \frac{4}{3}d_2$$
$$d_1 = d_2 = 0$$

Hence, $F\left(\frac{8}{5},\frac{16}{5}\right) = \{(0,0)\}$. Then, by the definition of critical cones, $F\left(\frac{8}{5},\frac{16}{5}\right) = \{(0,0)\} = C\left(\left(\frac{8}{5},\frac{16}{5}\right),\left(-\frac{1}{5},\frac{3}{40}\right)\right)$ trivially, we know $\left(\frac{8}{5},\frac{16}{5}\right)$ is a local minimizer by Theorem 10.15.

Then, since $0 < \frac{8}{5}$, we know $(0,0)$ is a global minimizer, $\left(\frac{8}{5},\frac{16}{5}\right)$ is a local minimizer, and $(2,2)$ is a saddle point.

4 Question 4

(4 points) We say *linear constraint qualification* holds if all the constraint functions $c_i(x), i \in \mathcal{I} \cup \mathcal{E}$ are linear functions plus a constant (affine functions).

Prove that if Linear constrant qualification holds at a feasible point $x \in \Omega$, then $T_\Omega(x) = \mathcal{F}(x)$

Proof:

Let $x \in \Omega$ be any feasible point. Since linear constraint qualification holds for all constraints, we have

$$\nabla c_i(x) = a_i \in R^n, \forall i \in E$$
$$\nabla c_t(x) = b_t \in R^n, \forall t \in I$$

, where n is the dimension of x, and a_i, b_j are all constant vectors that are made up of the coefficients of the non-constant terms of the affine constraint functions. Then, for any $d \in F(x)$, we have

$$d^T a_i = 0, \forall i \in E$$

$$d^T b_j \le 0, \forall j \in A(x) \cap I$$

For all inactive constraints, let $\varepsilon_t > 0$ be such that $c_t(x) = -\varepsilon_t < 0$. Then, let $\delta_{min} = \min\left\{\frac{\varepsilon_t}{d^T b_t} | t \text{ is inactive and } d^T b_t > 0\right\}$ if $d^T b_t > 0$ for some inactive t.

Otherwise, $\delta_{min} = 1$. Then, define $\{t_k\}$ to be $t_k = \frac{1}{k}\delta_{min} > 0$. It can be easy to see that $\{t_k\}_{k=1}^{+\infty}$ is a positive scaler sequence decreasingly converging to 0 as k approaches positive infinity from the upper bound $t_1 = \delta_{min}$.

Let $\{z_k\}_{k=1}^{+\infty}$ be a vector sequence such that $z_k = x + t_k d \in R^n$. Then, we see if z_k is feasible. **For all $i \in E$, since c_i are affine**, they have perfect linear approximation, so

$$c_i(z_k) = c_i(x + t_k d) = c_i(x) + \nabla c_i(x)^T (t_k d) = c_i(x) + t_k \nabla c_i(x)^T d$$

Note that since x is feasible, $c_i(x) = 0$, so

$$c_i(z_k) = c_i(x) + t_k \nabla c_i(x)^T d = \nabla c_i(x)^T t_k d = t_k d^T a_i = 0$$

For all $j \in A(x) \cap I$,

$$c_j(z_k) = c_j(x + t_k d) = c_j(x) + \nabla c_j(x)^T (t_k d) = c_j(x) + t_k d^T b_j$$

Since t_k is positive,

$$c_j(z_k) = c_j(x) + t_k d^T b_j \le 0 + t_k 0 = 0$$

For other inactive constraints (i.e. $c_t(x) < 0$),

$$c_t(z_k) = c_t(x + t_k d) = c_t(x) + t_k d^T \nabla c_t(x) = c_t(x) + t_k d^T \mathrm{b_t}$$

If $d^T \mathrm{b_t} \le 0$, then since $c_t(x) < 0$ and $t_k > 0$, we have

$$c_t(z_k) = c_t(x) + t_k d^T \mathrm{b_t} < 0$$

If $d^T \mathrm{b_t} > 0$, then we have

$$c_t(z_k) = c_t(x) + t_k d^T \mathrm{b_t} = -\varepsilon_t + \frac{1}{k}\delta_{min} d^T \mathrm{b_t} \le -\varepsilon_t + \frac{1}{k}\frac{\varepsilon_t}{d^T b_t} d^T \mathrm{b_t} = -\varepsilon_t + \frac{1}{k}\varepsilon_t$$
$$= \left(\frac{1}{k} - 1\right)\varepsilon_t$$

Note that since $k \ge 1, \frac{1}{k} - 1 \le 0, \varepsilon_t > 0$, we know

$$c_t(z_k) = \left(\frac{1}{k} - 1\right)\varepsilon_t \le 0$$

Therefore, z_k is feasible. Then, we see

$$\lim_{k\to+\infty} z_k = \lim_{k\to+\infty}(x + t_k d) = x + d \lim_{k\to+\infty}\frac{1}{k}\delta_{min} = x$$

Thus, z_k is approaching x. Finally, we have

$$\lim_{k\to+\infty}\frac{z_k - x}{t_k} = \lim_{k\to+\infty}\frac{x + t_k d - x}{t_k} = d$$

Therefore, $d \in T_\Omega(x)$ by definition, which means $F(x) \subseteq T_\Omega(x)$. Also, by Lemma 10.8, $T_\Omega(x) \subseteq F(x)$, we know $T_\Omega(x) = F(x)$.