PROBLEMS IN 02612 CONSTRAINED OPTIMIZATION

ASSIGNMENT 2

Jens	Østergaard	s173130
Mirza	Hasanbasic	s172987
Yevgen	Zainchkovskyy	s062870
Jonas	Bentsen	s163145

Wednesday May 2, 2018

DTU Compute

Department of Applied Mathematics and Computer Science

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Problem 1

Interior-Point Algorithm for Linear Programming

Question 1 Problem 14.15 in Nocedal and Wright, p. 419-420. You must describe and list the algorithm you use. You must also provide a test script that tests your interior-point LP implementation.

Consider the following linear problem

$$\begin{aligned} \min \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{1.1}$$

x and c are vectors in $\mathbb{R}^n$, b is a vector in $\mathbb{R}^m$ and A is a $m \times n$ matrix (corresponds to having n variables and m constraints).

1.1. Starting point Choosing a starting point is essential for obtaining convergence and to choose the right starting point we solve the following problem.

$$\begin{array}{lll} \min\limits_{x} \ \frac{1}{2}x^T x & s.t. & Ax = b \\ \min\limits_{(\lambda,s)} \ \frac{1}{2}s^T s & s.t. & A^T\lambda + s = c \end{array} \tag{1.2}$$

This can be rewritten explicit as

$$\tilde{x} = A^T(AA^T)^{-1}b \qquad \tilde{\lambda} = (AA^T)^{-1}Ac, \qquad \tilde{s} = c - A^T\tilde{\lambda} \tag{1.3}$$

but as $\tilde{x}$ and $\tilde{s}$ will have non-positive components we adjust them to:

$$\hat{x} = \tilde{x} + \delta_x e \qquad \hat{s} = \tilde{s} + \delta_s e \qquad e = (1, \ldots, 1)^T \tag{1.4}$$

Where $\delta_x = \max(-(3/2)\min_i \tilde{x}_i, 0)$ and $\delta_s = \max(-(3/2)\min_i \tilde{s}_i, 0)$

This will make $\tilde{x} \geq 0$ and $\tilde{s} \geq 0$ which will make them suitable as a starting point.

To ensure that the components x^0 and s^0 is not to close to zero or to dissimilar two more scarlars are added. The scalars are defined as:

$$\hat{\delta}_x = \frac{1}{2}\frac{\hat{x}^T\hat{s}}{e^T\hat{s}} \qquad \hat{\delta}_s = \frac{1}{2}\frac{\hat{x}^T\hat{s}}{e^T\hat{x}}, \tag{1.5}$$

and the starting point is defined as:

$$x^0 = \hat{x} + \hat{\delta}_x e, \qquad \lambda^0 = \tilde{\lambda} \qquad s^0 = \hat{s} + \hat{\delta}_s e \tag{1.6}$$

1.2. Predictor-Corrector Algorithm In this section the mathematical calculations are explained given in the algorithm. This is done to solve the linear problem. It is notice that a inequality constrain is present. This is handled by introducing a slack variable s the lagrangian for the equation (1.2) is given by

$$\mathcal{L}(x, \lambda, s) = c^T x - \lambda^T (Ax - b) - s^T x \qquad \text{where } \lambda \in \mathbb{R}^m \tag{1.7}$$

Then the KKT conditions for the system can be writen as

$$\begin{aligned} \nabla_x \mathcal{L}(x, \lambda, s) = A^T\lambda - s &= c \\ Ax &= b \\ x_i s_i &= 0, \quad i = 1, \ldots n \\ (x, s) &\geq 0 \end{aligned} \tag{1.8}$$

$x_i s_i$ can be reformulated into matrix from as XSe where $X = \text{diag}(x_1, \ldots, x_n)$, $S = \text{diag}(s_1, \ldots, s_n)$ and $e = (1, 1, \ldots, 1)^T$

Now the primal-dual interior point can be derived by restating the optimally conditions as a mapping $F \in \mathbb{R}^{2n+m} \to \mathbb{R}^{2n+m}$ the

$$\begin{aligned} F(x, \lambda, s) = \begin{bmatrix} A^T\lambda + s - c \\ Ax - b \\ XSe \end{bmatrix} &= 0 \\ (x, s) &\geq 0 \end{aligned} \tag{1.9}$$

It is necessary to determine step and a measure of desirability of each search point. The desirability is the pairwise products of $x_i s_i$ where $i = 1, \ldots, n$ and $x > 0$, $s > 0$. This is know as the duality measure and is defined as

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i s_i = \frac{x^T s}{n} \tag{1.10}$$

The Newton's method is used to determine the search direction and the Jacobian is needed to define the Newton's method:

$$J(x, \lambda, s) = \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta s \end{bmatrix} = -F(x, \lambda.s) \qquad \text{where } J(x, \lambda, s) = \begin{bmatrix} \partial_x F & \partial_\lambda F & \partial_s F \end{bmatrix} \tag{1.11}$$

Newton's system of equations can now be written as,

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{aff}} \\ \Delta \lambda^{\text{aff}} \\ \Delta s^{\text{aff}} \end{bmatrix} = - \begin{bmatrix} A^T \lambda + s - c \\ Ax - b \\ XSe \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ -XSe \end{bmatrix} \tag{1.12}$$

This results in the affine search direction also known as the predictor step.

The challenge is that the affine search direction of the Newton's method generates possible linearization errors. To compensate for this a corrector step used.

By solving the system for $(\Delta x^{\text{aff}} \Delta \lambda^{\text{aff}} \Delta s^{\text{aff}})^T$ will give

$$(x_i + \Delta x_i^{\text{aff}})(s_i + \Delta s_i^{\text{aff}}) = x_i s_i + x_i \Delta s_i^{\text{aff}} + s_i \Delta x_i^{\text{aff}} + \Delta x_i^{\text{aff}} \Delta s_i^{\text{aff}} = \Delta x_i^{\text{aff}} \Delta s_i^{\text{aff}} \tag{1.13}$$

This is the updated value of $x_i s_i$. This is clearly different from the ideal value 0. In an attempt to correct can be carried out by solving the problem

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{cor}} \\ \Delta \lambda^{\text{cor}} \\ \Delta s^{\text{cor}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\Delta X^{\text{aff}} \Delta S^{\text{aff}} e \end{bmatrix}, \tag{1.14}$$

to obtain the step $(\Delta x^{\text{cor}}, \Delta \lambda^{\text{cor}}, \Delta s^{\text{cor}})$

By combining the predictor and corrector step $(\Delta x^{\text{aff}}, \Delta \lambda^{\text{aff}}, \Delta s^{\text{aff}})$ + $(\Delta x^{\text{cor}}, \Delta \lambda^{\text{cor}}, \Delta s^{\text{cor}})$. Combining the the affine and the corrector step will often reduce the duality measure better then the affine step would do alone.

With these two components it is only necessary to center the path to ensure that the Newton's method wont take too small steps in the affine direction. This is a result of the complementary condition from (1.8).

By introducing the center path $\mathcal{C}$, each point $(x_\tau, \lambda_\tau, s_\tau) \in \mathcal{C}$ is ensured to be strictly feasible points. Note that $\mathcal{C}$ is parameterized by a scalar $\tau > 0$. This satisfies the following conditions

$$\begin{aligned} \nabla_x \mathcal{L}(x, \lambda, s) = A^T\lambda - s &= c \\ Ax &= b \\ x_i s_i &= \tau, \quad i = 1, \dots n \\ (x, s) &\geq 0 \end{aligned} \tag{1.15}$$

These conditions differ from the KKT conditions at the complementarity condition. It is visible that (1.15) approximate (1.8) in an increasing degree as $\tau \to 0$ This ensure that $\mathcal{C}$ converges to primal-dual solution of the linear program.

The central path can now be defined as $\mathcal{C} = \{(x_\tau, \lambda_\tau, s_\tau) | \tau > 0\}$.

The modified step can be computed by solving

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{cor}} \\ \Delta \lambda^{\text{cor}} \\ \Delta s^{\text{cor}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -XSe + \mu\sigma e \end{bmatrix} \tag{1.16}$$

Where μ is the duailty measure defined in (1.10) and σ is the centering parameter $\sigma = \left(\frac{\mu_{\text{aff}}}{\mu}\right)^3$ where $\mu_{\text{aff}} = (x + \alpha_{\text{aff}}^{\text{pri}} \delta x^{\text{aff}})^T (s + \alpha_{\text{aff}}^{\text{dual}} \delta s^{\text{aff}})/n$.

$$\alpha_{\text{aff}}^{\text{pri}} = \min\left(1, \min_{i:\Delta x_i^{\text{aff}} < 0} -\frac{x_i}{\Delta x_i^{\text{aff}}}\right), \qquad \alpha_{\text{aff}}^{\text{dual}} = \min\left(1, \min_{i:\Delta s_i^{\text{aff}} < 0} -\frac{s_i}{\Delta s_i^{\text{aff}}}\right), \tag{1.17}$$

This ensure that the maximum allowable step along the affine-scaling direction can be taken.

With (1.12),(1.14) and (1.16) can be combined to calculate the search direction by solving.

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{cor}} \\ \Delta \lambda^{\text{cor}} \\ \Delta s^{\text{cor}} \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ -XSe + -\Delta X^{\text{aff}} \Delta S^{\text{aff}} e + \mu \sigma e \end{bmatrix} \tag{1.18}$$

With the given direction, the final step is finding the step length.

The quantities are defined as:

$$\alpha_{k,\max}^{\text{pri}} \stackrel{\text{def}}{=} \min_{i:\Delta x_i^k < 0} -\frac{x_i^k}{\Delta x_i^k}, \qquad \alpha_{k,\max}^{\text{dual}} \stackrel{\text{def}}{=} \min_{i:\Delta s_i^k < 0} -\frac{s_i^k}{\Delta s_i^k}. \tag{1.19}$$

and the step length is then calculated by

$$\alpha_k^{\text{pri}} = \min(1, \eta \alpha_{k,\max}^{\text{pri}}), \qquad \alpha_k^{\text{dual}} = \min(1, \eta \alpha_{k,\max}^{\text{dual}}) \tag{1.20}$$

where $\eta_k \in [0.9, 1.0]$ is chosen so that $\eta_k \to 1$ as the iterate converges to the primal-dual solution.

The new iterate is obtained by:

$$\begin{aligned} x^{k+1} &= x^k + \alpha_k^{\text{pri}} \Delta x^k \\ (x^{k+1}, s^{k+1}) &= (\lambda^k, s^k) + \alpha_k^{\text{dual}} (\Delta \lambda^k, \Delta s^k) \end{aligned} \tag{1.21}$$

The algorithm is seen here

Algorithm 1 Predictor-Corrector Algorithm (Mehrotra [207])

1: Calculate (x_0, λ_0, s_0)
2: **for** k=0,1,2,... **do**
3: Set $(x, \lambda, s) = (x^k \lambda^k, s^k)$ and solve (1.12) for $(\Delta x^{\text{aff}}, \Delta \lambda^{\text{aff}}, \Delta s^{\text{aff}})$;
4: Calculate $\alpha_{\text{aff}}^{\text{pri}}, \alpha_{\text{aff}}^{\text{dual}}$ and μ_{aff};
5: Set centering parameter to $\sigma = (\mu_{\text{aff}}/\mu)^3$;
6: Solve 1.18 for $(\Delta x, \Delta \lambda, \Delta s)$;
7: Calculate $\alpha_k^{\text{pri}}, \alpha_k^{\text{dual}}$;
8: Set

$$\begin{aligned} x^{k+1} &= k^k + \alpha_k^{\text{pri}} \Delta x \\ (\lambda^{k+1}, s^{k+1}) &= (\lambda^k, s^k) + \alpha_k^{\text{dual}} (\Delta \lambda, \Delta s); \end{aligned}$$

9: **end for**

With the implementation given below:

```
% Interior point (IP) algorithm for solving Linear Programming (LP)
% problems of the type:
%
%   min c'x              c in R^n,     x in R^n
%   s.t Ax = b, x >= 0   A in R^{m*n} b in R^m
%
%   n = number of unknows
%   m = number of constraints

function [x_list, l_list, s_list, ra_norm, rl_norm, dual_gap, Converged] = iplp(c, A, b)
% Step 1, calculate starting point (page 410, book)
Converged = false;
eta = 0.99;
epsilon = 1e-8;
n = size(c,1);
m = size(b,1);

x_hat = A'*inv(A*A')*b;
l_hat = inv(A*A')*A*c;
s_hat = c - A'*l_hat;

delta_x = max(-3/2*min(x_hat), 0);
delta_s = max(-3/2*min(s_hat), 0);

x_hat = x_hat + delta_x * ones(n, 1);
s_hat = s_hat + delta_s * ones(n, 1);

delta_hat_x = (1/2) * (x_hat' * s_hat) / (ones(n,1)' * s_hat);
delta_hat_s = (1/2) * (x_hat' * s_hat) / (ones(n,1)' * x_hat);

x_list = x_hat + delta_hat_x * ones(n,1);
l_list = l_hat;
s_list = s_hat + delta_hat_s * ones(n,1);

rl_norm = [];
ra_norm = [];
dual_gap = [];

% Step 2, Predictor-Corrector algorithm
max_k = 10;

rc  = A'*l_list(:, end) + s_list(:, end) - c;
rb  = A*x_list(:, end) - b;

for k = 1:max_k
```

```
x = x_list(:, end);
l = l_list(:, end);
s = s_list(:, end);

% Solving 14.30 for \delta x^{aff}, \delta l^{aff} and \delta s^{aff},
KKT_like = [zeros(n)        A'      eye(n)
            A         zeros(m)  zeros(m,n)
            diag(s) zeros(n,m)     diag(x)];

XSe = diag(x)*diag(s)*ones(n,1);

rhs = [ -rc;
         -rb;
        -XSe];

[L, D, p] = ldl(KKT_like, 'lower', 'vector');
Delta = zeros(size(rhs,1),1);
Delta(p) = L'\(D\(L\rhs(p)));
Delta_x_aff = Delta(1:n);
Delta_l_aff = Delta(n+1:n+m);
Delta_s_aff = Delta(n+m+1:end);

% Step 2.3.
% Calculate \alhpa_{aff}^{pri} and \alhpa_{aff}^{dual}
all_xis = -(x./Delta_x_aff); % 14.32a
a_aff_pri  = min([1;all_xis(Delta_x_aff<0)]);

all_sis = -(s./Delta_s_aff); % 14.32b
a_aff_dual = min([1;all_sis(Delta_s_aff<0)]);

% Calculate \mu_{aff} (14.33)
mu_aff = ((x + a_aff_pri * Delta_x_aff)' * ...
    (s + a_aff_dual * Delta_s_aff)) / n;

mu = (x'*s)/n; % (14.6)

% Calculate centering parameter, sigma (14.34)
sigma = (mu_aff/mu)^3;

% Solve 14.35 for \Delta x, \Delta l and \Delta s
XSe_aff = -(diag(x)*diag(s)*ones(n,1)) ...
    - (diag(Delta_x_aff)*diag(Delta_s_aff) * ones(n,1)) ...
    + (sigma*mu*ones(n,1));

rhs = [     -rc;
```

```
            -rb;
        XSe_aff];

    stepDelta = zeros(size(rhs,1),1);
    stepDelta(p) = L'\(D\(L\rhs(p)));
    stepDelta_x = stepDelta(1:n);
    stepDelta_l = stepDelta(n+1:n+m);
    stepDelta_s = stepDelta(n+m+1:end);

    % Calculate \alpha_k^{pri} and \alpha_k^{dual}
    all_xis = -(x./stepDelta_x);
    alpha_k_max_pri = min(all_xis(stepDelta_x<0));
    alpha_k_pri = min([1; eta*alpha_k_max_pri]);

    all_sis = -(s./stepDelta_s);
    alpha_k_max_dual = min(all_sis(stepDelta_s<0));
    alpha_k_dual = min([1; eta*alpha_k_max_dual]);

    % Set the x, l and s for the next step
    x_list = [x_list x+alpha_k_pri*stepDelta_x];
    l_list = [l_list l+alpha_k_dual*stepDelta_l];
    s_list = [s_list s+alpha_k_dual*stepDelta_s];

    % Convergence
    rl_norm  = [rl_norm, norm(rc, 2)]; % Gradient of the Lagrangian
    ra_norm  = [ra_norm, norm(rb, 2)]; % Constraint equality
    dual_gap = [dual_gap, abs(x_list(:, end)'*s_list(:,end)/n)];

    Converged = (ra_norm(end) < epsilon) && (rl_norm(end) < epsilon) && (dual_gap(end) < epsilon);
    if Converged
        break
    end

    % Set rc and rb for the next step
    rb  = (1 - alpha_k_pri)*rb;  % r_L in slides
    rc  = (1 - alpha_k_dual)*rc; %
end
end
```

1.3. Testing the algorithm We test our implementation as described in the problem: Matrix A is generated randomly, and x, s, b and c set as follows:

$$
\begin{aligned}
x_i &= \begin{cases} \text{random positive number} & \text{i} = 1,\ 2,\ \ldots,\ \text{m}, \\ 0 & \text{i} = \text{m+1,m+2,}\ldots\text{,n}, \end{cases} \\
s_i &= \begin{cases} \text{random positive number} & \text{i} = \text{m+1,m+2,}\ldots\text{,n} \\ 0 & \text{i} = 1,\ 2,\ \ldots,\ \text{m} \end{cases} \\
\lambda &= \text{random vector} \\
c &= A^\top \lambda + s \\
b &= Ax
\end{aligned}
$$

Essentially generating the problem *backwards*. First we generate some random unknowns x, slack s and Lagrange multipliers λ. We then calculate c and b which allows us to run the solver while knowing the optimal values of x in advance. The code is as follows:

```
n = 300; % number unknows
m = 200; % number of constraints

A = rand(m,n);
x = zeros(n, 1);
x(1:m, 1) = abs(rand(m,1));
s = zeros(n, 1);
s(m+1:n, 1) = abs(rand(n-m,1));
l = rand(m,1);
c = A'*l+s;
b = A*x;
```

Running the code with tolerance of $\epsilon = 10^{-8}$, 300 unknowns and 200 constraints, our code converged in just 10 steps!

Finally, we found it interesting to follow the convergence of our solver (according to slide 17, Lecture 07) which we present in the figure below:

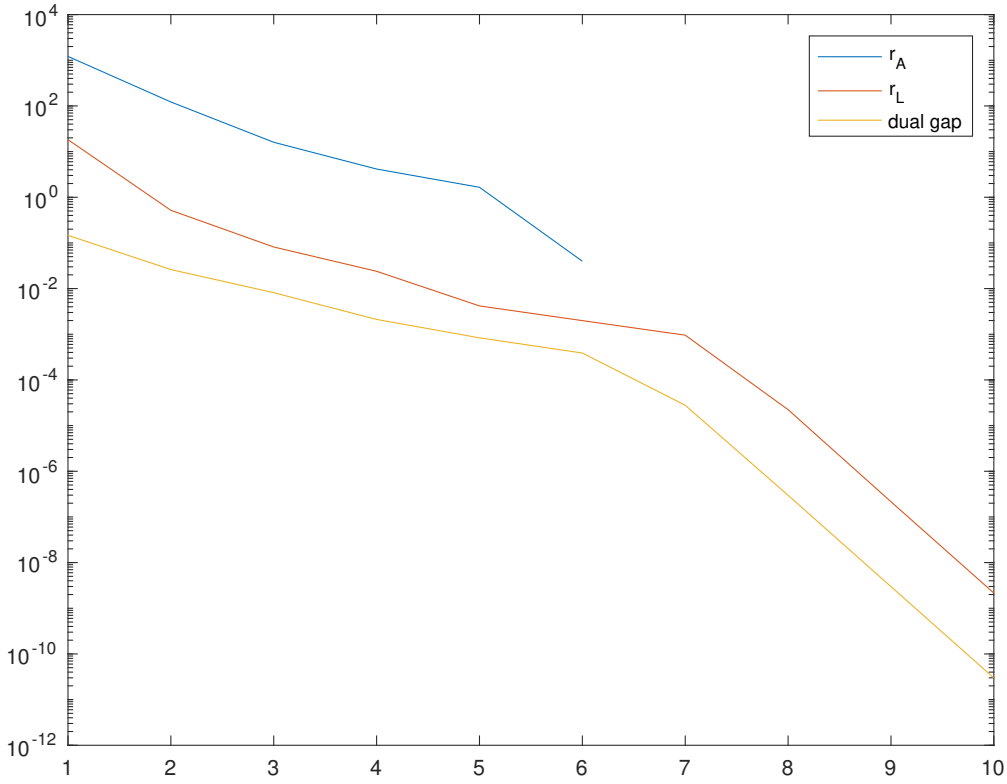

Figure 1.1: Convergence of the IP algorithm.

Problem 2

Equality Constrained SQP

Question 1 Solve problem 18.3 in Nocedal and Wright. Make a table with the iteration sequence. Describe your program and make a flow chart of its structure. You should have separate functions for computation of $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)$, $c_i(x)$, $\nabla c_i(x)$ $\nabla^2 c_i(x)$.

Problem 18.3 presents us with the following:

$$\begin{aligned} \min \quad & \exp(x1x2x3x4x5) - \frac{1}{2}(x_1^3 + x_2^3 + 1)^2 \\ \text{subject to} \quad & x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ & x_2 x_3 - 5x_4 x_5 = 0 \\ & x_1^3 + x_2^3 + 1 = 0 \end{aligned} \tag{2.1}$$

The starting point is defined $x_0 = (-1.8, 1.7, 1.9, -0.8, -0.8)$ and the optimal solution is $x* = (-1.71, 1.59, 1.82, -0.763, -0.763)$

Consider the equality-constrained problem

$$\begin{aligned} \min \quad & f(x) \\ \text{subject to} \quad & c(x) = 0 \end{aligned} \tag{2.2}$$

The $A(x)$ notation to denote the Jacobian matrix,

$$A(x)' = \left[\nabla c_1(x), \nabla c_2(x), ..., \nabla c_m(x)\right]. \tag{2.3}$$

In this problem, the SQP metod will be used. Initally the Lagrangian function is given by

$\mathcal{L}(x, \lambda) = f(x) - \lambda' c(x)$. The first order KKT conditions of the equality constrained problem 2.2 can be written as

$$F(x, \lambda) = \begin{bmatrix} \nabla f(x) - A(x)'\lambda \\ c(x) \end{bmatrix} = 0 \tag{2.4}$$

Then the Jacobian of 2.4 will become

$$F'(x, \lambda) = \begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x, \lambda) & -A(x)' \\ A(x) & 0 \end{bmatrix} \tag{2.5}$$

The Newton step is given by

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix} \tag{2.6}$$

where p_k and p_λ solve the Newton-KKT system

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x, \lambda) & -A(x)' \\ A(x) & 0 \end{bmatrix} \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix} = \begin{bmatrix} \nabla - \nabla f_k + A'_k \lambda_k \lambda \\ -c_k \end{bmatrix} \tag{2.7}$$

Assumption 1. *Note, that the constraint Jacobian $A(x)$ has full row rank and the matrix $\nabla^2_{xx}\mathcal{L}(x, \lambda)$ is positive definite on the tangent space of the constraints, that is, $d'\nabla^2_{xx}\mathcal{L}(x, \lambda)d > 0$ for all $d \neq 0$ such that $A(x)d = 0$*

With equation 2.1, suppose that at the iterate (x_k, λ_k), then the equation is modelled using quadratic programming

$$\begin{aligned} \min_p \quad & f_k + \nabla f'_k p + \frac{1}{2} p' \nabla^2_{xx}\mathcal{L}_k p \\ \text{subject to} \quad & A_k p + c_k = 0 \end{aligned} \tag{2.8}$$

If assumption 1 is true, then this problem will have a unique solution (p_k, l_k) that satisfies

$$\begin{aligned} \nabla^2_{xx}\mathcal{L}_k p_k + \nabla f_k - A'_k l_k = 0 \\ A_k p_k + c_k = 0 \end{aligned} \tag{2.9}$$

It is seen that the vector p_k and l_k can be identified with the solution of the Newtion equations 2.7. Furthermore we we subtract the term $A'_k \lambda_k$ from both sides of the first equation in 2.7, then we obtain

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}_k & -A'_k \\ A_k & 0 \end{bmatrix} \begin{bmatrix} p_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f_k \\ -c_k \end{bmatrix} \tag{2.10}$$

The SQP method can be stated as:

Algorithm 2 Local SQP Algorithm for solving 2.2

1: Choose an initial pair (x_0, λ_0); set $k \leftarrow 0$
2: **while** $||\nabla\mathcal{L}_k||_\infty$>tol **and** $||c_k||_\infty$>tol **do**
3: Evaluate $f_k, \nabla f_k, \nabla^2_{xx}\mathcal{L}_k, c_k$ and A_k
4: Solve 2.8 to obtain p_k and l_k
5: Set $x_{k+1} \leftarrow x_k + p_k$ and $\lambda_{k+1} \leftarrow l_k$
6: **end while**

Presented with a flow chart of our implemenation below:

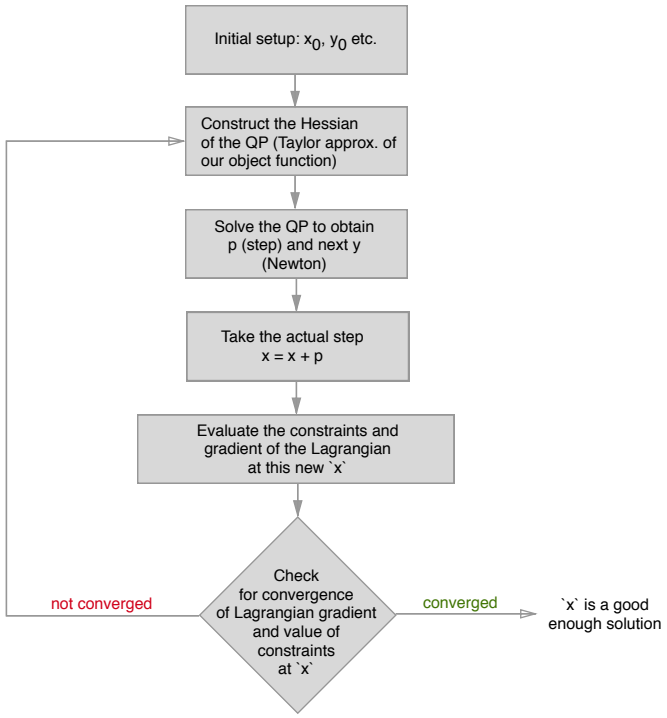

Figure 2.1: Flowchart of our equality constrained SQP solver.

Now 2.1 can be solved with algorithm 2. Implementation in Matlab is given below (note the use of tensors to pack the Hessian of the constraints into a single entity):

```
x0 = [-1.8; 1.7; 1.9; -0.8; -0.8];
y0 = [1;1;1];
[f, df, d2f] = prob2_obj(x0);
[c, dc, d2c] = prob2_constr(x0);
max_iter = 100;
epsilon = 10^-8;
Converged = false;
iter = 0;
m = size(c,1);
x = x0;
y = y0;

while ~Converged && (iter < max_iter)
    iter = iter + 1;
    % Compute Hessian Matrix
    H = d2f;
    for i = 1:m
        H = H - y(i) * d2c(:,:,i);
    end
    % Solve equality constrained QP
    [p, y] = EQQP(H, df, dc, -c);
    % Take step
    x = x + p;
    % Re-evaluate
    [f, df, d2f] = prob2_obj(x);
    [c, dc, d2c] = prob2_constr(x);
    % Lagrangian gradient
    dL = df - dc*y;
    % Convergence / KKT 1st order conditions
    Converged = (norm(dL, inf) < epsilon) && (norm(c, inf) < epsilon);
end
```

Where `[f, df, d2f] = prob2_obj(x0);` is defined in Matlab as a function, where the output for the MatLab function is the functionvalue f ∇f and $\nabla^2_{xx} f$ and equivalently for the constraints `[c, dc, d2c] = prob2_constr(x0);` . The code can be seen in the appendix section 4.1.1 and 4.1.2 page 37

Table 2.1: The sequence of algorithm 2 with tolerance

Iter.	1	2	3	4	5	6
x_1	-1.8	-1.75375	-1.75963	-1.7192	-1.71711	-1.71714
x_2	1.7	1.6388	1.64462	1.59834	1.59568	1.59571
x_3	1.9	1.76364	1.74651	1.82518	1.8273	1.82725
x_4	-0.8	-0.760239	-0.757955	-0.764343	-0.763648	-0.763643
x_5	-0.8	-0.760239	-0.757955	-0.764343	-0.763648	-0.763643

In table 2.1 convergence acan be seen.

Question 2 Implement the procedure with a damped BFGS approximation to the Hessian matrix. Make a table with the iteration sequence.

Now approximate the Hessian matrix for each step by a matrix B_k. Possible reasoning behind estimation of the Hessian is not always easy to compute.

The idea is to update B_k, where vectors s_k and y_k are utilized and defined as

$$s_k = x_{k+1} - x_k, \tag{2.11}$$

and

$$y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1}). \tag{2.12}$$

Procedure 1 - Damped BFGS Updated. *Given that matrix B_k is symmetric and positive definite.*

Define s_k as in 2.11 and y_k as in 2.12 and set

$$r_k = \theta_k y_k + (1 - \theta_k) B_k s_k$$

where the scalar θ_k is defined as

$$\theta_k = \begin{cases} 1 & \text{if } s_k' y_k \geq 0.2 s_k' B_k s_k \\ (0.8 s_k' B_k s_k)/(s_k' B_k s_k - s_k' y_k) & \text{if } s_k' y_k < 0.2 s_k' B_k s_k \end{cases} \tag{2.13}$$

Update B_k as follows:

$$B_{k+1} = B_k - \frac{B_k s_k s_k' B_k}{s_k' B_k s_k} + \frac{r_k r_k'}{s_k' r_k} \tag{2.14}$$

From equation 2.14, it can be seen that it is the standard BFGS update, with y_k replaced by r_k. This will make sure that B_{k+1} is positive definite

Then EQSQP is given with the damped BFGS approximation, and the implementation is below.

```
% Problem 2.2

x0 = [-1.8; 1.7; 1.9; -0.8; -0.8];
%x0 = [-1.5; 1.2; 2.5; 0; 0];

y0 = [10; 10; 10];
```

```
[f, df, d2f] = prob2_obj(x0);
[c, dc, d2c] = prob2_constr(x0);

n = size(x0, 1);
B = eye(n);

max_iter = 100;
epsilon = 1e-10;

Converged = false;
iter = 1;

x = x0;
y = y0;

xarray = [];
xarray = [xarray x0];
normdlarray = [];
normclarray = [];

while ~Converged && (iter <= max_iter)
    % Solve equality constrained QP
    [p, y] = EQQP(B, df, dc, -c);

    % Take step and modify x
    x = x + p;

    % Calculate dLxklk1 (yk)
    % with old `x` and new `y`
    yk = df - dc*y;

    % Re-evaluate
    [f, df, d2f] = prob2_obj(x);
    [c, dc, d2c] = prob2_constr(x);

    % Lagrangian gradient
    dL = df - dc*y;

    % BFGS procedure
    q = dL - yk;
    Bp = B*p;

    cond = (p'*q >= 0.2*p'*Bp);

    % https://stackoverflow.com/questions/6409233/matlab-conditional-assignment
```

```
    theta = cond.*1 + (~cond).*((0.8*p'*Bp)/((p'*Bp)-(p'*q)));

    r = theta*q + (1-theta)*Bp;
    B = B + ((r*r')/(p'*r)) - ((Bp*Bp')/(p'*Bp));

    % Convergence / KKT 1st order conditions
    Converged = (norm(dL, inf) < epsilon) && (norm(c, inf) < epsilon);
    iter = iter + 1;
    xarray = [xarray x];
    normdlarray = [normdlarray norm(dL, inf)];
    normclarray = [normclarray norm(c, inf)];
end
```

The output in each itereation is displayed in table 2.2

Table 2.2: Iteration of damped BFGS

Iter.	0	1	2	3	4	5	6	7	8	9
x_1	-1.8	-1.7269	-1.7244	-1.7239	-1.7218	-1.7171	-1.7171	-1.7171	-1.7171	-1.7171
x_2	1.7	1.6087	1.6041	1.6035	1.6011	1.5957	1.5957	1.5957	1.5957	1.5957
x_3	1.9	1.8132	1.8138	1.8146	1.8186	1.8273	1.8272	1.8272	1.8272	1.8272
x_4	-0.8	-0.7636	-0.7628	-0.7629	-0.7631	-0.7637	-0.7636	-0.7636	-0.7636	-0.7636
x_5	-0.8	-0.7636	-0.7628	-0.7629	-0.7631	-0.7637	-0.7636	-0.7636	-0.7636	-0.7636

When comparing the iterations from table 2.1 and 2.2, then it can seen, that using BFGS requires almost twice the iterations, but it does not require to use the second derivative.

Question 3 Implement the procedure with a damped BFGS approximation to the Hessian matrix and line search. Make a table with the iteration sequence.

Implementations using the BFGS results in many iterations. One way to reduce iterations is to introduce line search. In SQP methods, there is a merit functions that decides if a trial step should be accepted, where in line search methods, this merit function controls the size of the step. For computation and evaluation of a merit function, inequality constraints, are often converted to the form $\bar{c}(x, s) = c(x) - s = 0$, where $s \geq 0$ is a vector of slack variables.

So the ℓ_1 merit function for 2.1 takes form $\phi_1(x; \mu) = f(x) + \mu||c(x)||_1$. In a line search method, a step $\alpha_k p_k$ will be accepted if the following sufficient decreases condition holds

$$\phi_1(x_k + \alpha_k p_k; \mu_k) > \phi_1(x_k; \mu_k) + \nu\alpha_k D_1(\phi(x_k; \mu_k)p_k)$$

where $D_1(\phi(x_k; \mu_k)p_k)$ denotes the direction derivative of ϕ_1 in the direction of p_k.

An effective method of choosing μ, considers the effect of the step on a model of the merit function. So if ϕ_1 is defined, as piecewise quadratic model, by

$$q_\mu(p) = f_k + \nabla f_k' p + \frac{\sigma}{2} p' \nabla^2_{xx} \mathcal{L}_k p + \mu \cdot m(p), \tag{2.15}$$

where

$$m(p) = ||c_k + A_k p||_1.$$

After computing a step p_k, the penalty parameter μ is chosen large enough that

$$q_\mu(0) - q_\mu(p_k) \geq \rho\mu[m(0) - m(p_k)], \quad \rho \in (0, 1). \tag{2.16}$$

Then it follows from 2.15 and 2.8 that inequality 2.16 is satisfied for

$$\mu \geq \frac{\nabla f_k' p_k + (\sigma/2) p_k' \nabla^2_{xx} \mathcal{L}_k p_k}{(1-\rho)||c_k||_1}. \tag{2.17}$$

The step for x may not be good for large steps. Line search method proposes reducing the size for every step, by a chosen factor α, which is found using the merit function.

Algorithm 3 Line Search SQP Algorithm

Choose parameters $nu \in (0, 0.5)$, $\tau \in (0, 1)$ and an initial pair (x_0, λ_0)
Evaluate $f_0, \nabla f_0, c_0$ and A_0
If a quasi-Newton Approximation is used, choose an initial $n \times n$ symmetric positive definite Hessian approximation B_0, otherwise compute $\nabla^2_{xx} \mathcal{L}_0$
while until a convergence test is satisfied **do**
    Compute p_k by solving 2.8; let $\hat{\lambda}$ be the corresponding multiplier
    Set $p_\lambda \leftarrow \hat{\lambda} - \lambda_k$
    Choose μ_k to satisfy 2.17 with $\sigma = 1$
    Set $\alpha_k \leftarrow 1$
    while $\phi_1(x_k + \alpha_k p_k; \mu_k) > \phi_1(x_k; \mu_k) + \nu \alpha_k D_1(\phi(x_k; \mu_k) p_k)$ **do**
        Reset $\alpha_k \leftarrow \tau_\alpha \alpha_k$ for some $\tau_\alpha \in (0; \tau]$
    end while
    $x_{k+1} \leftarrow x_k + \alpha_k p_k$ and $\lambda_{k+1} \leftarrow \lambda_k + \alpha_k + p_\lambda$
    Evaluate $f_{k+1}, \nabla f_{k+1}, c_{k+1}$ and A_{k+1} (and possibly $\nabla^2_{xx} \mathcal{L}_{k+1}$
    If a quasi-Newton approximation is used, set

$$s_k \leftarrow \alpha_k p_k \quad \text{and} \quad y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1})$$

    and obtain B_{k+1} by updating B_k using a quasi-Newton formular
end while

The corresponding MabLab code can be seen below

```
% Problem 2.3 - EQSQP with a damped BFGS
% approximation to the Hessian matrix and line search.

clear all;
clc

x0 = [-1.8; 1.7; 1.9; -0.8; -0.8];
%x0 = [-1.5; 1.2; 2.5; 0; 0];

y0 = [10; 10; 10];

[f, df, d2f] = prob2_obj(x0);
[c, dc, d2c] = prob2_constr(x0);

n = size(x0, 1);
B = eye(n);

max_iter = 100;
epsilon = 1e-10;

rho = 0.99;
eta = 0.55;
tau = 0.99;
max_ls_iter = 20;

Converged = false;
iter = 1;

x = x0;
y = y0;
lambda = abs(y0);

xarray = [];
xarray = [xarray x0];
normdlarray3 = [];
normclarray3 = [];

while ~Converged && (iter <= max_iter)
    % Solve equality constrained QP
    [p, y_hat] = EQQP(B, df, dc, -c);
    p_lambda = y_hat - y; % 18.3 book

    % -- BEGIN LINE SEARCH PART (18.3 Book) -- %
    alpha = 1;
    mu = (df'*p + 1/2*p'*B*p)/((1-rho)*norm(c,1)); % Penalty parameter
```

```
[f_new, ~, ~] = prob2_obj(x+alpha*p);
[c_new, ~, ~] = prob2_constr(x+alpha*p);

ls_iter = 1;
while ls_iter < max_ls_iter
    part1 = (f_new + mu*norm(c_new,1));
    part2 = (df'*p + mu*norm(c_new,1));
    D1    = (df'*p + mu*norm(c,    1));

    part3 = eta*alpha*D1;

    if part1 > (part2 + part3); break; end
    alpha = tau * alpha;
    [f_new, ~, ~] = prob2_obj(x+alpha*p);
    [c_new, ~, ~] = prob2_constr(x+alpha*p);
    ls_iter = ls_iter+1;
end
% -- END LINE SEARCH PART -- %

% Take step and modify `x` and `lambda`
x = x + alpha*p;
y = y + alpha*p_lambda;

% Calculate dLxklk1 (yk)
% with old `x` and new `y`
yk = df - dc*y;

% Re-evaluate
[f, df, d2f] = prob2_obj(x);
[c, dc, d2c] = prob2_constr(x);

% Lagrangian gradient
dL = df - dc*y;

% BFGS procedure
q = dL - yk;
Bp = B*p;

cond = (p'*q >= 0.2*p'*Bp);

% https://stackoverflow.com/questions/6409233/matlab-conditional-assignment
theta = cond.*1 + (~cond).*((0.8*p'*Bp)/((p'*Bp)-(p'*q)));

r = theta*q + (1-theta)*Bp;
```

```
    B = B + ((r*r')/(p'*r)) - ((Bp*Bp')/(p'*Bp));

    % Convergence / KKT 1st order conditions
    Converged = (norm(dL, inf) < epsilon) && (norm(c, inf) < epsilon);
    iter = iter + 1;
    xarray = [xarray x];
    normdlarray3 = [normdlarray3 norm(dL, inf)];
    normclarray3 = [normclarray3 norm(c, inf)];
end
```

The x values from each iteration, are displayed in table 2.3, with each iteration sequence.

Table 2.3: Iteration of algorithm 3

Iter.	0	1	2	3	4	5	6	7	8	9
x_1	-1.8	-1.7269	-1.7244	-1.7239	-1.7218	-1.7171	-1.7171	-1.7171	-1.7171	-1.7171
x_2	1.7	1.6087	1.6041	1.6035	1.6011	1.5957	1.5957	1.5957	1.5957	1.5957
x_3	1.9	1.8132	1.8138	1.8146	1.8186	1.8273	1.8272	1.8272	1.8272	1.8272
x_4	-0.8	-0.7636	-0.7628	-0.7629	-0.7631	-0.7637	-0.7636	-0.7636	-0.7636	-0.7636
x_5	-0.8	-0.7636	-0.7628	-0.7629	-0.7631	-0.7637	-0.7636	-0.7636	-0.7636	-0.7636

Comparing table 2.2 and 2.3, the iterations are seen to be exactly the same. This is due to the initialize values being close to x where the line search is not needed in any iteration step.

Question 4 Plot the rate of convergence for the 3 algorithms implemented. Comment on the rate for convergence for the 3 algorithms

From the figure 2.2, we can see that the basic overshoots and the convergence is faster, yet is oscillating compared to the two other algorithms. It is though seen that it convergence quicker in this case. Note that BFGS and BFGS with line search have exactly the same iteration sequence.

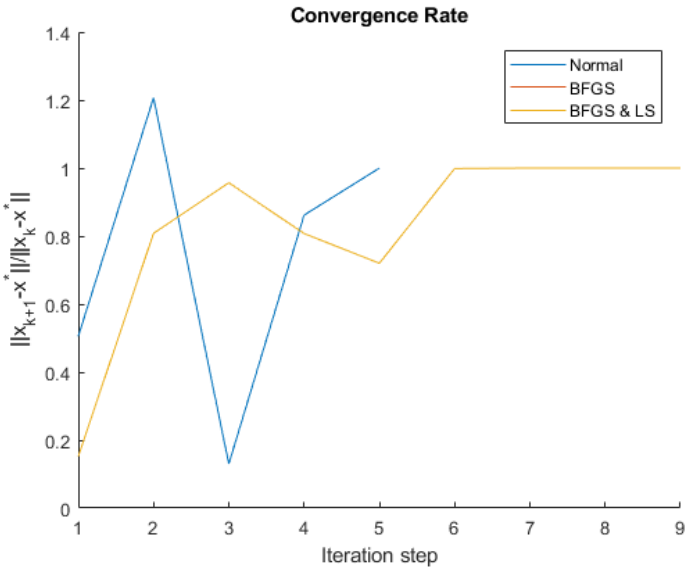

Figure 2.2: Convergence rate of the x values for each method.

Problem 3

Inequality Constrained SQP

In the three following questions different versions of the SQP procedure is asked to be implemented on the Himmelblau, EQP stated underneath.

$$\min_x \quad f(x) = (x_1^2 + x^2 - 11)^2 + (x1 + x2 - 7)^2 \tag{3.1a}$$
$$\text{s.t.} \quad c_1(x) = (x_1 + 2)^2 - x_2 \geq 0 \tag{3.1b}$$
$$c_2(x) = -4x_1 + 10x_2 \geq 0 \tag{3.1c}$$

To solve this problem the BFGS approximation of the Hessian matrix will be used and updated in every iteration. Furthermore the Hessian will be initiated as the identify matrix i.e.

$$B_0 = I,$$

and this framework will then be used to implement and test a basic SQP with damped BFGS, the SQP with damped BFGS and line search, and finally the Trust region based SQP will be implemented.

The SQP extended to the inequality constrained space is a simple extension the the equality based introduced in part 2. Consider the general problem:

$$\min_x \quad f(x) \tag{3.2a}$$
$$\text{s.t.} \quad c_i(x) = 0 \quad i \in \mathcal{E} \tag{3.2b}$$
$$c_j(x) \geq 0 \quad j \in \mathcal{I} \tag{3.2c}$$

The equality and inequality constraints are then linearized using the same method as in part 2

to give to following:

$$\min_p \quad f_k + \nabla f_k' p + p' B_k p \tag{3.3a}$$
$$\text{s.t.} \quad \nabla c_i(x_k)' p + c_i(x_k) = 0 \quad i \in \mathcal{E} \tag{3.3b}$$
$$\nabla c_j(x_k)' p + c_j(k) \geq 0 \quad j \in \mathcal{I} \tag{3.3c}$$

This problem can now be solved using the active set algorithm seen in the previous assignment applied to the implementations of the equality constrained SQP in part 2. Better results where though seen in assignment one by implementing the ℓ_∞ norm to a EQP. This gives the expectancy of equivilant behavior from the implementations of the SQP as the exact same general problem is set up now as in the EQP.

Therefore the problem (3.8) will have the ℓ_∞ norm introduced together with the slack variables v, w and t,

$$\min_p \quad f_k + \nabla f_k' p + p' B_k p + \mu \sum_{i \in \mathcal{E}} t_i + \mu \sum_{j \in \mathcal{I}} v_j + w_j \tag{3.4a}$$
$$\text{s.t.} \quad \nabla c_i(x_k)' p + c_i(x_k) = v_j - w_j \quad i \in \mathcal{E} \tag{3.4b}$$
$$\nabla c_j(x_k)' p + c_j(k) \geq -t_i \quad j \in \mathcal{I} \tag{3.4c}$$
$$v, w, t \geq 0, \tag{3.4d}$$
$$\tag{3.4e}$$

where μ is the penalty term used in the objective function to ensure the equality constraints will be upheld at a given point. Initially the term can be set sufficiently high to produce solutions but in the trust region implementation the addition of algorithm Algorithm 18.5 from Nocedal & Wright will be implemented to ensure no imbalance in the merit functions. The introduction of the slack variables also handles any inconsistent linerezations that could occur. Furthermore it also allows the initial guess to be outside the feasible space.

In the Himmelblau problem there are no equality constraints so the equality constraints can be dropped and the resulting general framework would be:

$$\min_p \quad f_k + \nabla f_k' p + p' B_k p + \mu \sum_{i \in \mathcal{E}} t_i \tag{3.5a}$$
$$\text{s.t.} \quad \nabla c_j(x_k)' p + c_j(k) \geq -t_i \quad j \in \mathcal{I} \tag{3.5b}$$
$$v, w, t \geq 0 \tag{3.5c}$$
$$\tag{3.5d}$$

Furthermore it has been discovered that the implementations of the ℓ_∞ norm is not necessary to solve the problems 1 and 2 but only for the 3 where thrust region is implemented. Contour plot of the HimmelBlau problem

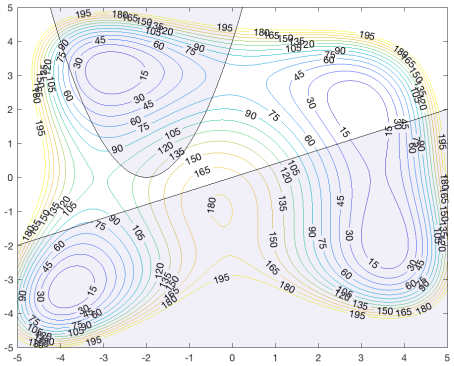

Figure 3.1: HimmelBlau

Question 1 Implement a SQP procedure with a damped BFGS approximation to the Hessian matrix. Make a table with the iteration sequence for different starting points. Plot the iteration sequence in a contour plot.

The implementation of the SQP with the BFGS approximation is almost identical to the implementation used in task two. This will bbe done with the intial points within the feasabe region and they are $x_0 \in (0,0)', (-4,0)', (3,4)'$.

Table 3.1: $X_0 = (0,0)'$

$x_0 = (0,0)'$	0	1	2	3	4	5	6	7	8	9	10
x_1	0.0	14.0	6.08	2.24	2.51	3.30	2.99	2.92	2.85	2.91	2.99
x_2	0.0	22.0	2.43	3.19	3.08	2.71	2.78	2.71	2.32	2.14	2.02

$x_0 = (0,0)'$	11	12	13	14	15
x_1	3.0	3.0	3.0	3.0	3.0
x_2	2.0	2.0	2.0	2.0	2.0

Table 3.2: $x_0 = (3,4)'$

Table 3.3: $x_0 = (3, 4)'$

$x_0 = (3, 4)'$	0	1	2	3	4	5	6	7	8	9	10	11
x_1	3.0	0.52	3.86	1.25	-0.40	-0.11	11.67	4.91	1.67	2.10	3.99	2.59
x_2	4.0	0.21	1.54	3.70	-0.16	0.40	15.11	1.96	3.03	2.85	2.06	2.61

$x_0 = (3, 4)'$	12	13	14	15	16	17	18	19	20	21	22
x_1	2.83	3.05	3.03	3.00	2.99	3.00	3.0	3.0	3.0	3.0	3.0
x_2	2.49	2.36	2.35	2.22	2.05	2.01	2.0	2.0	2.0	2.0	2.0

Table 3.4: $x_0 = (-4, 0)'$

$x_0 = (-4, 0)'$	0	1	2	3	4	5	6	7	8	9	10	11
x_1	-4.0	-2.73	-3.21	-3.78	-3.51	-3.54	-3.55	-3.55	-3.55	-3.55	-3.55	-3.55
x_2	0.0	-1.09	-1.28	-1.51	-1.40	-1.42	-1.42	-1.42	-1.42	-1.42	-1.42	-1.42

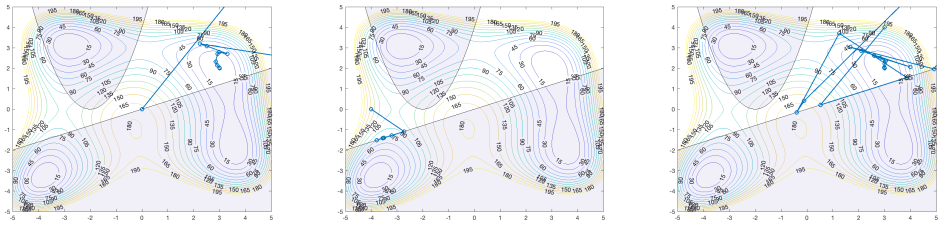

Figure 3.2: Contour plots for task 3.1

It is seen with all three starting point is the path that is chosen and therefor also the amount moved in that direction are very dependent on the gradient in the current iteration and in all tree examples it is seen that the iterations jump around alot. It is noticeable that the iterations go in the general direction of a optimum.

Question 2 Implement the procedure with a damped BFGS approximation to the Hessian matrix and line search. Make a table with the iteration sequence. Make a table with relevant statistics (function calls etc). Plot the iteration sequence in a contour plot.

Introducing the IQSQP with line search is again very similar to the implementation used in task 2 but now with slight change in the A matrix. The algorithm will be tested with the same initial points as in the task 3.1.

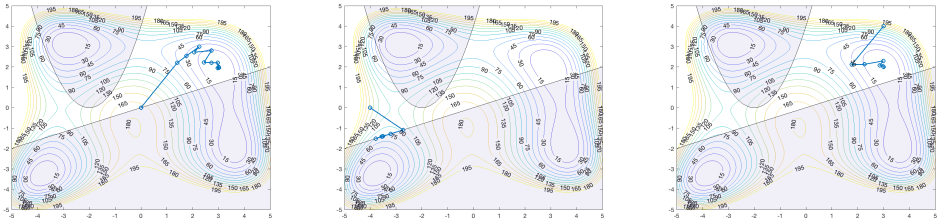

Figure 3.3: $x_0 = (0, 0)$ left, $x_0 = (-4, 0)$ center,$x_0 = (3, 4)$ right

Table 3.5: $x_0 = (0, 0)'$

$x_0 = (0, 0)'$	0	1	2	3	4	5	6	7	8	9	10
x_1	0.0	1.4	1.76	2.25	2.06	2.73	2.44	2.72	2.97	2.97	3.01
x_1	0.0	2.2	2.55	3.00	2.72	2.80	2.22	2.21	2.20	1.95	1.95

$x_0 = (0, 0)'$	11	12	13	14	15
x_1	3.0	3.0	3.0	3.0	3.0
x_2	2.0	2.0	2.0	2.0	2.0

Table 3.6: $x_0 = (3, 4)'$

$x_0 = (3, 4)'$	0	1	2	3	4	5	6	7	8	9	10	11
x_1	3.0	0.52	3.86	1.25	-0.40	-0.11	11.67	4.91	1.67	2.10	3.99	2.59
x_2	4.0	0.21	1.54	3.70	-0.16	0.40	15.11	1.96	3.03	2.85	2.06	2.61

Table 3.7: $x_0 = (-4, 0)'$

$x_0 = (-4, 0)'$	0	1	2	3	4	5	6	7	8	9	10	11
x_1	-4.0	-2.73	-3.21	-3.78	-3.51	-3.54	-3.55	-3.55	-3.55	-3.55	-3.55	-3.55
x_2	0.0	-1.09	-1.28	-1.51	-1.40	-1.42	-1.42	-1.42	-1.42	-1.42	-1.42	-1.42

In the line search method it is seen that the variance is quite a bit smaller and the interactions are much more well behaved. This is also expected and aligns with the tendency seen in task 2. It is also see when the starting point is very close to the optimum, then using the line search or not, is the same.

Question 3 Implement a Trust Region based SQP algorithm for this problem. Make a table with the iteration sequence. Make a table with relevant statistics (function calls etc). Plot the iteration sequence in a contour plot.

The trust region SQP method also works on the same principals as the other methods examined here for the IQSQP but the trust region adds a constraint to the sub problem defining the area of trust. This area of trust is the area where the model is considered to be reliable. The inclusion of the trust region can impose the sub-problem to become unfeasible and therefor it is necessary to introduce further procedures that resolve this issue. This means in practise that the amount of computations is increased, but with the benefit that the solver should not move around as drastically as seen in the previous two examples.

The trust region is implemented with the parameter Δ_k that imposes the size of the trust region. With this parameter included in problem the resulting formulation becomes:

$$\min_p \quad f_k + \nabla f_k' p + p' B_k p \tag{3.6a}$$
$$\text{s.t.} \quad \nabla c_i(x_k)' p + c_i(x_k) = 0 \quad i \in \mathcal{E} \tag{3.6b}$$
$$\nabla c_j(x_k)' p + c_j(k) \geq 0 \quad j \in \mathcal{I} \tag{3.6c}$$
$$||p||_\infty \leq \Delta_k \tag{3.6d}$$

In the formulation the IQP is introduced with the slight adjustment of the function and the implementation of the ℓ_∞ norm. This leads to the problem also disregarding the equality constrains as they are not needed in this problem:

$$\min_p \quad f_k + \nabla f_k' p + p' B_k p + \mu \sum_{i \in \mathcal{E}} t_i \tag{3.7a}$$
$$\text{s.t.} \quad \nabla c_j(x_k)' p + c_j(k) \geq -t_i \quad j \in \mathcal{I} \tag{3.7b}$$
$$v, w, t \geq 0 \tag{3.7c}$$
$$||p||_\infty \leq \Delta_k \tag{3.7d}$$

This can then be written in matrix notation as

$$\min_p \quad f_k + \begin{bmatrix} \nabla f_k \\ \mu \end{bmatrix}' \begin{bmatrix} p \\ t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} p \\ t \end{bmatrix}' \begin{bmatrix} B & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ t \end{bmatrix} \tag{3.8a}$$
$$\text{s.t.} \quad \begin{bmatrix} \nabla C(x_k) & I \\ 0 & I \\ I & 0 \\ -I & 0 \end{bmatrix} \begin{bmatrix} p \\ t \end{bmatrix} \geq \begin{bmatrix} -C(x_k) \\ 0 \\ -\Delta_k \\ -\Delta_k \end{bmatrix} \tag{3.8b}$$

Table 3.8: $x_0 = (0, 0)'$

$x_0 = (0,0)'$	0	1	2	3	4	5	6	7	8	9	10	11
x_1	0.0	1.0	2.0	2.0	2.25	2.50	2.75	3.00	3.02	2.99	3.0	3.0
x_2	0.0	1.0	2.0	2.0	2.25	2.22	1.97	2.22	1.97	2.00	2.0	2.0

It is seen in the contour plots 3.4 of the tree different iterations that the step length is much shorter and it the iterations do not "bounce" as much around as seen in the previous tree

Table 3.9: $x_0 = (-4, 0)'$

$x_0 = (-4,0)'$	0	1	2	3	4	5	6	7	8	9
x_1	-4.0	-3.0	-3.01	-3.17	-3.17	-3.42	-3.67	-3.78	-3.72	-3.97
x_2	0.0	-1.0	-2.00	-3.00	-3.00	-2.75	-2.50	-2.25	-2.00	-1.75

$x_0 = (-4,0)'$	10	11	12	13
x_1	-3.75	-3.60	-3.55	-3.55
x_2	-1.50	-1.44	-1.42	-1.42

Table 3.10: $x_0 = (3, 4)'$

$x_0 = (3,4)'$	0	1	2	3	4	5	6	7	8	9	10	11
x_1	3.0	2.0	3.00	3.00	3.00	2.94	2.99	3.0	3.0	3.0	3.0	3.0
x_2	4.0	3.0	2.17	2.17	2.17	2.11	2.05	2.0	2.0	2.0	2.0	2.0

examples. Yet there are still many iterations and therefor we can conclude even though the iterations are more stable, (caused by the trust region) the amount of iterations can be the same. Furthermore this method is very susceptible to the method that is used to set the μ values and if very small μ values are set initially then the result would be many more iterations. This in conjunction with the extra computations needed to change the size of the trust region and the re specification of μ in each iteration leas to a possible high computational load, compared to the line search and the basic IQSQP.

Also note that the integration sequence of for the initial starting point $x_0 = (-4, 0)'$ moves out of the feasible space, this is a consequence of the slack Incorporated in the objective function as the constraints do not need to be upheld. This behavior could be minimized by initializing at a higher μ value, keeping in mind that μ is updated dynamically and therefor the effects may be indistinguishable i a general case.

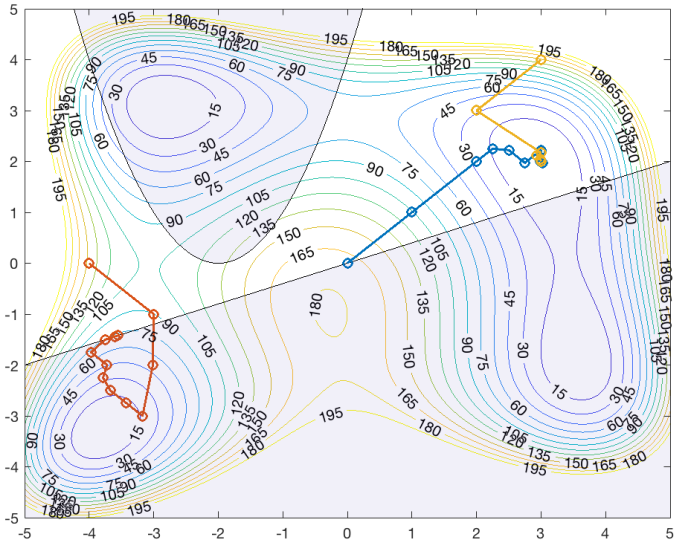

Figure 3.4: Task 3 all initial x_0 included

Bibliography

This page is intentionally left blank.

4

Appendix

4.1. Code for Assignment 2

4.1.1 Problem 2 Objective functions

```
% Problem 2 objective function
% returns the function, gradient and Hessian evaluated
% at a point `x`

function [f, df, d2f] = prob2_obj(x)

% Objective function
f = exp(prod(x)) - 0.5*( x(1)^3 + x(2)^3 + 1)^2;

% Gradient of the objective function
df = zeros(5,1);
df(1,1) = exp(prod(x))*prod(x([2,3,4,5])) - 3*x(1)^2*(1+x(1)^3+x(2)^3);
df(2,1) = exp(prod(x))*prod(x([1,3,4,5])) - 3*x(2)^2*(1+x(1)^3+x(2)^3);
df(3,1) = exp(prod(x))*prod(x([1,2,4,5]));
df(4,1) = exp(prod(x))*prod(x([1,2,3,5]));
df(5,1) = exp(prod(x))*prod(x([1,2,3,4]));

x1 = x(1);
x2 = x(2);
x3 = x(3);
x4 = x(4);
x5 = x(5);

d2f = [
```

```
    [             x2^2*x3^2*x4^2*x5^2*exp(x1*x2*x3*x4*x5) - 6*x1*(x1^3 + x2^3 + 1) - 9*x1^4,
    x3*x4*x5*exp(x1*x2*x3*x4*x5) - 9*x1^2*x2^2 + x1*x2*x3^2*x4^2*x5^2*exp(x1*x2*x3*x4*x5),
    x2*x4*x5*exp(x1*x2*x3*x4*x5) + x1*x2^2*x3*x4^2*x5^2*exp(x1*x2*x3*x4*x5),
    x2*x3*x5*exp(x1*x2*x3*x4*x5) + x1*x2^2*x3^2*x4*x5^2*exp(x1*x2*x3*x4*x5),
    x2*x3*x4*exp(x1*x2*x3*x4*x5) + x1*x2^2*x3^2*x4^2*x5*exp(x1*x2*x3*x4*x5)]
    [ x3*x4*x5*exp(x1*x2*x3*x4*x5) - 9*x1^2*x2^2 + x1*x2*x3^2*x4^2*x5^2*exp(x1*x2*x3*x4*x5),
    x1*x3*x5*exp(x1*x2*x3*x4*x5) + x1^2*x2*x3^2*x4*x5^2*exp(x1*x2*x3*x4*x5),
    x1*x3*x4*exp(x1*x2*x3*x4*x5) + x1^2*x2*x3^2*x4^2*x5*exp(x1*x2*x3*x4*x5)]
    [               x2*x4*x5*exp(x1*x2*x3*x4*x5) + x1*x2^2*x3*x4^2*x5^2*exp(x1*x2*x3*x4*x5),
    [               x2*x3*x5*exp(x1*x2*x3*x4*x5) + x1*x2^2*x3^2*x4*x5^2*exp(x1*x2*x3*x4*x5),
    [               x2*x3*x4*exp(x1*x2*x3*x4*x5) + x1*x2^2*x3^2*x4^2*x5*exp(x1*x2*x3*x4*x5),

% Hessian of the objective function (using Matlabs
% built in hessian functions and symbolics).
% syms x1 x2 x3 x4 x5;
% ff = exp(x1*x2*x3*x4*x5) - 0.5*(x1^3 + x2^3 + 1)^2;
% hhh = hessian(ff,[x1,x2,x3,x4,x5]);
% d2f = eval(subs(hhh));  % if we need to eval
end
```

4.1.2 Problem 2 Constrains

```
% Problem 2 constraint
% returns the function, gradient and Hessian evaluated
% at a point `x`

% x = [-1.8; 1.7; 1.9; -0.8; -0.8];

function [c, dc, d2c] = prob2_constr(x)
x1 = x(1);
x2 = x(2);
x3 = x(3);
x4 = x(4);
x5 = x(5);

c1 = x1^2 + x2^2 + x3^2 + x4^2 + x5^2 - 10;
c2 = x2*x3 - 5*x4*x5;
c3 = x1^3 + x2^3 + 1;

% Constraint function
c = [c1; c2; c3];

% Gradient of constraint function
dc = zeros(5, 3);
```

```
dc(:,1) = [  2*x1;   2*x2; 2*x3;  2*x4;  2*x5];
dc(:,2) = [     0;     x3;   x2; -5*x5; -5*x4];
dc(:,3) = [3*x1^2; 3*x2^2;    0;     0;     0];

% Hessian of the constraint function (using a Tensor)
d2c = zeros(5,5,3);
d2c(:,:,1) = 2*eye(5);   % constraint 1

d2c(2,3,2) = 1;          % constraint 2
d2c(3,2,2) = 1;
d2c(4,5,2) = -5;
d2c(5,4,2) = -5;

d2c(1,1,3) = 6*x1;       % constraint 3
d2c(2,2,3) = 6*x2;
end
```