THE TECHNICAL UNIVERSITY OF DENMARK

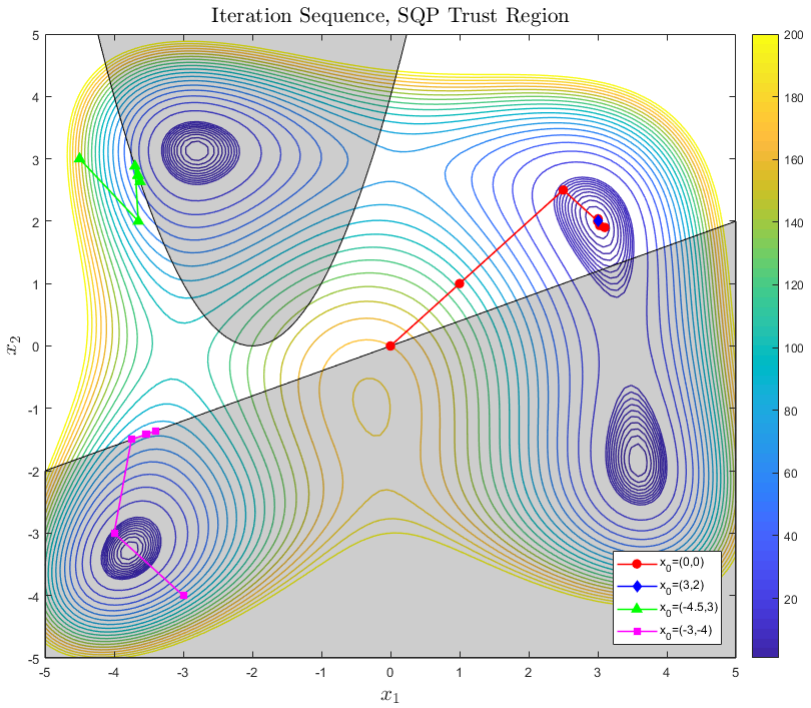
CONSTRAINED OPTIMIZATION

Exercises

Author:
Christian Dandanell GLISSOV,
S146996

Professor:
John BAGTERP JØRGENSEN

May 26, 2021
Kongens Lyngby



DTU Compute
Department of Applied Mathematics and Computer Science

Table of Contents

Problem 1

OBS: Please note that some drivers used to run the different functions throughout the report can be found in the Appendices.
Consider the problem

$$\min_x \quad f(x) = 3x_1^2 + 2x_1x_2 + x_1x_3 + 2.5x_2^2 + 2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3 \tag{1.1a}$$
$$\text{s.t.} \quad x_1 + x_3 = 3 \tag{1.1b}$$
$$x_2 + x_3 = 0 \tag{1.1c}$$

In the form

$$\begin{aligned} \min_x \quad & f(x) = \tfrac{1}{2}x'Hx + g'x \\ \text{s.t.} \quad & A'x = b \end{aligned} \tag{1.2}$$

P1.1

> What are H, g, A, b?

Consider the above quadratic problem in Equation 1.1. This problem can be written in the form of Equation 1.2. It is clear that we want to minimize over x_1, x_2 and x_3. Therefore $H \in \mathbb{R}^{3\times3}$ and $g \in \mathbb{R}^{3\times1}$. Two constraints can also be observed, therefore $A \in \mathbb{R}^{3\times2}$ and $b \in \mathrm{R}^{2\times1}$.

H and g is defined from (1.1a):

$$H = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix}, \quad g = \begin{bmatrix} -8 \\ -3 \\ -3 \end{bmatrix}$$

While A and b is defined from (1.1b) and (1.1c):

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, b = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

Now the above terms can be used to rewrite (1.1) to (1.2).

P1.2

> Write the KKT optimality conditions.

The KKT optimality conditions are given in Theorem 12.1 in (Nocedal & Wright, 1999, p. 321), first the Lagrangian is introduced:

$$\mathcal{L}(x,\lambda) = f(x) - \lambda' c(x) = \frac{1}{2}x'Hx + g'x - \lambda'\left(A'x - b\right)$$

The gradient of the Lagrangian can then be found:

$$\nabla_x \mathcal{L}\left(x,\lambda\right) = \nabla f(x) - \sum_{i\in\mathcal{E}\cup\mathcal{I}} \lambda_i \nabla c_i(x) \tag{1.3}$$

$$= Hx + g - A\lambda \tag{1.4}$$

Looking at the constraint qualification it is seen that $\nabla c_i(x)$, $i \in (1,2)$ where all constraints are active, due to the problem only consisting of equality constraints, Definition 12.1 in (Nocedal & Wright, 1999, p. 308). Hence the KKT conditions are:

$$Hx + g - A\lambda = 0 \tag{1.5a}$$

$$A'x - b = 0 \tag{1.5b}$$

$$\lambda\left(A'x - b\right) = 0 \tag{1.5c}$$

Rewriting (1.5) into a linear system of equations, (16.4) in (Nocedal & Wright, 1999) (notice the notation of the QP is slightly different):

$$\begin{bmatrix} G & -A \\ -A^T & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -g \\ -b \end{bmatrix} \tag{1.6}$$

Using the above system and the found defined H, g, A and b one can solve the problem in (1.1).

P1.3

> Make a function `[x,lambda]=EqualityQPSolver(H,g,A,b)` for solution of equality constrained quadratic programs.

To solve Equation 1.1, the KKT matrix, (1.6), can be used. The solution of the

system is then given by

$$\begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} H & -A \\ -A' & 0 \end{bmatrix}^{-1} \begin{bmatrix} -g \\ -b \end{bmatrix} \tag{1.7}$$

Implementation of this in MATLAB can be seen in Listing 1. To make the algorithm

Listing 1 EqualityQPSolver

```
function [x,lambda] = EqualityQPSolver(H,g,A,b)
% Solves convex quadratic program with equality constrains

%Dimensions
[s1,s2] = size(H);
[a1,a2] = size(A);

%KKT matrix
F = [H , -A; -A' , zeros(a2,a2)];


%RHS
h = [-g;-b];
eps=0;
% Use factorization, use cholesky if positive definite
if(all(eig(F) > eps))
    [L,p] = chol(F,'lower');
    x = L'\(L\h);
else % Use LDL else
    [L,D,p] = ldl(F,'vector');
    z = L'\(D\(L\h(p)));
    x = z(p);
end

%Get solution
lambda = x((s1+1):end);
x = x(1:s1);
```

more efficient, factorization of the KKT matrix is used. It is noticed that the KKT matrix is symmetric. The following factorizations are therefore taken into account:

1. Cholesky is used if the KKT matrix is positive definite and symmetric.
2. LDL factorization is used if the matrix is indefinite and symmetric.

P1.4

Test your program on the above problem.

The solution of (1.1) is found to be

$$x^* = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} \tag{1.8}$$

While the Langrange multipliers is found to be

$$\lambda^* = \begin{bmatrix} 3 \\ -2 \end{bmatrix} \tag{1.9}$$

Inserting the results satisfies the first order necessary conditions, the solution must be optimal. By inserting the values in (1.2) the lowest function value which is feasible is found as:

$$f(x^*) = \frac{1}{2} \begin{bmatrix} 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} + \begin{bmatrix} -8 & -3 & -3 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} = -3.5 \tag{1.10}$$

P1.5

Generate random convex quadratic programs (consider how this can be done) and test you program.

From (Bagterp Jørgensen, 2018b) it is seen that

$$b = A'x$$

$$g = A\lambda - Hx$$

Furthermore to ensure that the quadratic program (QP) is well-defined H has to be symmetric and for the QP to be convex H needs to be positive semi-definite. The code can be seen in Listing 2

A number of QP's are then generated randomly in `MATLAB` using a uniform random

Listing 2 RandomQP initialises a new random convex quadratic program.

```
function [H,g,A,b,x,lambda] = RandomQP(n,m)
% n is the dimensions of H
% m is the number of constraints
H = rand(n,n);
% Generate H based on random matrix
% Assure it's symmetric
H = 0.5*(H+H');
% Assure it's positive definite
H = H+n*eye(n);
% Generate A
A = rand(n,m);
%Generate true solution
x = rand(n,1);
% Construct b
b = A'*x;
% Define Lambda
lambda = rand(m,1);
% Construct g
g = A*lambda-H*x;
```

number generator. To test the program the error is simply calculated:

$$E(x^*) = \frac{1}{N}\sum_{i=1}^{N}(x - x^*)$$

Comparing the true solution x and the found solution x^*, for 1000 generated random convex QP's, the average error was found to be approximately in the order of 10^{-12} and approximately 0.08 for λ^* and λ. Please note, it was necessary to add a small noise to the KKT-matrix in the EqualityQPSolver to prevent singular matrices, this reduced the accuracy slightly.

P1.6

Write the sensitivity equations for the equality constrained convex QP.

To determine the sensitivity of the solution with respect to g and b, p is introduced in the problem. To investigate the sensitivity of the solution we let the solutions depend on p, $x = x(p)$ and $\lambda = \lambda(p)$:

$$f(x,p) = \frac{1}{2}x'Hx + (g + p_g)'x \tag{1.11}$$

and

$$c(x,p) = A'x - (b + p_b) \tag{1.12}$$

Where

$$p = \begin{bmatrix} p_g \\ p_b \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix}$$

The parameter sensitivities are given by (Bagterp Jørgensen, 2019f, p. 18):

$$\begin{bmatrix} \nabla x\left(p_0\right) & \nabla \lambda\left(p_0\right) \end{bmatrix} = \begin{bmatrix} W_{xp} & -\nabla_p c\left(x_0^*, p_0\right) \end{bmatrix} \begin{bmatrix} W_{xx} & -\nabla_x c\left(x_0^*, p_0\right) \\ -\nabla_x c\left(x_0^*, p_0\right)^T & 0 \end{bmatrix}^{-1} \tag{1.13}$$

First we determine $\nabla_p c(x,p)$ and $\nabla_x c(x,p)$:

$$\nabla_p c(x,p) = \nabla_p \left(A'x - (b+p)\right) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \tag{1.14}$$

$$\nabla_x c(x,p) = \nabla_x \left(A'x - b\right) = A \tag{1.15}$$

Now W_{xx} and W_{xp} is determined

$$W_{xx} = \nabla_{xx}^2 f(x,p) - \sum_{i \in \mathcal{E}} \lambda_i \nabla_{xx}^2 c_i(x,p) \tag{1.16}$$

$$= \nabla_{xx}^2 \left(\frac{1}{2} x'Hx + (g+p)'x\right) - \lambda' \nabla_{xx}^2 \left(A'x - (b+p)\right) \tag{1.17}$$

$$= \nabla_x (Hx + g + p) - \lambda' \nabla_x A \tag{1.18}$$

$$= H \tag{1.19}$$

And

$$W_{xp} = \nabla^2_{xp} f(x,p) - \sum_{i \in \mathcal{E}} \lambda_i \nabla^2_{xp} c_i(x,p) \tag{1.20}$$

$$= \nabla^2_{xp} \left(\frac{1}{2} x' H x + (g+p)' x \right) - \lambda' \nabla^2_{xp} \left(A' x - (b+p) \right) \tag{1.21}$$

$$= \nabla_p (Hx + g + p) - \lambda' \nabla_p A \tag{1.22}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{1.23}$$

Now everything is inserted into (1.13):

$$[\nabla x(p) \quad \nabla \lambda(p)] = - \begin{bmatrix} W_{xp} & - \nabla_p c(x,p) \end{bmatrix} \begin{bmatrix} W_{xx} & -\nabla_x c(x,p) \\ -\nabla_x c(x,p)^T & 0 \end{bmatrix}^{-1} \tag{1.24}$$

$$= \begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 6 & 2 & 1 & -1 & 0 \\ 2 & 5 & 2 & 0 & -1 \\ 1 & 2 & 4 & -1 & -1 \\ -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 \end{bmatrix}^{-1} \tag{1.25}$$

$$= \begin{bmatrix} -0.0769 & -0.0769 & 0.0769 & 0.4615 & -0.3846 \\ -0.0769 & -0.0769 & 0.0769 & -0.5385 & 0.6154 \\ 0.0769 & 0.0769 & -0.0769 & 0.5385 & 0.3846 \\ 0.4615 & -0.5385 & 0.5385 & 2.2308 & -0.6923 \\ -0.3846 & 0.6154 & 0.3846 & -0.6923 & 3.0769 \end{bmatrix} \tag{1.26}$$

The change in $x(p)$, $\nabla x(p)$, as the first 3 columns in (1.26) and $\nabla \lambda(p)$ is the last two columns.

P1.7

Make a function that returns the sensitivities of the solution with respect to g and b. Test your program and discuss how you can verify that the sensitivities you compute are correct.

Based on P1.6 a function is implemented in MATLAB as seen in Listing 3 Changing

Listing 3 SensitivitiesEQP returns the sensitivities and the Taylor approximation to the solutions.

```
function [dx, dlambda, x_approx, lambda_approx] = SensitivitiesEQP(H,g,A,b,p)
% Computing solution for p = 0, in order to approximate x(p) and lambda(p)
[x0,lambda0] = EqualityQPSolver(H,g,A,b);

 % s1 number of var, s2 number of constrains
[s1,s2] = size(A);

% Determined parameters in Q1.6
dxc = A;
Wxx = H;
z = zeros(s2,s2);

%Setup sensitivity matrix
K = [Wxx -dxc; -dxc' z];

%Sensitivity is calculated
Kinv = -inv(K);

% Sensitivity of x(p) and sensitivity of lambda(p)
dx = Kinv(:,1:s1);
dlambda = Kinv(:,s1+(1:s2));

% 1st order Taylor approximation to the solutions
x_approx = x0 + dx'*p;
lambda_approx = lambda0 + dlambda'*p;
```

p to

$$p = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

This will change the constraints in (1.1b) and (1.1c) to

$$x_2 + x_3 = 1 \tag{1.27}$$

$$x_1 + x_3 = 4 \tag{1.28}$$

Based on the Taylor approximative solutions one can find the solutions when introducing the change p, (Bagterp Jørgensen, 2019f, p.18). Looking at just the solution

x:

$$\begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.4615 \\ -0.5385 \\ 0.5385 \end{bmatrix} + \begin{bmatrix} -0.3846 \\ 0.6154 \\ 0.3846 \end{bmatrix} = \begin{bmatrix} 2.0769 \\ -0.9231 \\ 1.9231 \end{bmatrix} \tag{1.29}$$

This was also found running the code in Listing 3. One can test the function by comparing the sensitivity analysis against the actual change in the optimal solution with respect to g and b. Using EqualityQPSolver it is seen that the same optimal solution is found as with the Taylor approximated solution found in (1.29).

P1.8

Write the dual program of the equality constrained QP

Duality theory shows one can create an alternative problem from our original convex equality program. For certain cases this can make problems easier to solve (Nocedal & Wright, 1999). From (Bagterp Jørgensen, 2019d, p. 25) the Lagrange dual problem is stated as:

$$\max_{x,\lambda} \quad \mathcal{L}(x,\lambda) \tag{1.30a}$$

$$\text{s.t.} \quad \nabla_x \mathcal{L}(x,\lambda) = 0, \quad \lambda \geq 0 \tag{1.30b}$$

From P1.2 the Lagrangian function was found, it's written below for the purpose of duality:

$$\mathcal{L}(x,\lambda) = \frac{1}{2}x'Hx + g'x - \lambda'\left(A'x - b\right) \tag{1.31}$$

$$= [Hx + g - A\lambda]'x - \frac{1}{2}x'Hx + b'\lambda \tag{1.32}$$

It is noticed that

$$\nabla_x \mathcal{L}(x,\lambda) = Hx + g - A\lambda \tag{1.33}$$

For the dual problem the gradient of the Lagrangian is simply 0 based on the constraint in (1.30). Hence the Lagrangian is:

$$\mathcal{L}(x,\lambda) = \frac{1}{2}x'Hx - b'\lambda \tag{1.34}$$

The object function for the dual problem can now be set up:

$$\min_{x,\lambda} \quad \frac{1}{2}x'Hx - b'\lambda \tag{1.35a}$$

$$\text{s.t.} \quad Hx + g - A\lambda = 0, \quad \lambda \geq 0 \tag{1.35b}$$

P1.9

To see how the dual QP relates to primal QP the dual QP is formulated on standard form. This is done by introducing the following:

$$\hat{x} = \begin{bmatrix} x \\ \lambda \end{bmatrix}, \quad \hat{H} = \begin{bmatrix} H & 0 \\ 0 & 0 \end{bmatrix}, \quad \hat{g} = \begin{bmatrix} 0 \\ -b \end{bmatrix}$$

$$\hat{A} = \begin{bmatrix} H \\ -A' \end{bmatrix}, \quad \hat{b} = -g$$

The dual QP can now be formulated as:

$$\min_{\hat{x}} \quad \frac{1}{2}\hat{x}'\hat{H}\hat{x} + \hat{g}'\hat{x} \tag{1.36a}$$

$$\text{s.t.} \quad \hat{A}\hat{x} = \hat{b} \tag{1.36b}$$

It is seen that (1.36) is a convex equality constrained QP in standard form. Now the KKT-conditions can be set up:

$$\mathcal{L}(\hat{x}, \mu) = \frac{1}{2}\hat{x}'\hat{H}\hat{x} + \hat{g}'\hat{x} - \alpha'\left(\hat{A}'\hat{x} - \hat{b}\right) \tag{1.37}$$

Where α is Lagrange multiplier for the dual problem. The KKT matrix is given by (1.6)

$$\begin{bmatrix} \hat{H} & -\hat{A} \\ -\hat{A}' & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ \alpha \end{bmatrix} = - \begin{bmatrix} \hat{g} \\ \hat{b} \end{bmatrix} \tag{1.38}$$

Inserting the standard form parameters the KKT matrix becomes:

$$\begin{bmatrix} H & 0 & -H \\ 0 & 0 & A' \\ -H & A & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \\ \alpha \end{bmatrix} = - \begin{bmatrix} 0 \\ -b \\ -g \end{bmatrix} \tag{1.39}$$

Investigating (1.39) it is seen that it can be compared to the primal QP.

$$\begin{bmatrix} -H\alpha + Hx \\ A^T\alpha \\ A\lambda - Hx \end{bmatrix} = \begin{bmatrix} 0 \\ b \\ g \end{bmatrix} \tag{1.40}$$

The first row in (1.40) states:

$$Hx - H\alpha = 0 \Rightarrow x = \alpha \tag{1.41}$$

The Lagrangian multipliers of the dual problem relates therefore to the minimizers x of the problem QP. The same relations can be found for row 2 and 3 of (1.40):

$$A'\alpha = b \Leftrightarrow A'x = b \tag{1.42}$$

(1.42) is equal to the constraints of the primal QP. Finally the last row:

$$A\lambda - Hx = g \Leftrightarrow Hx - A\lambda + g = 0 \tag{1.43}$$

Equation 1.43 is simply the gradient of the Lagrangian set to 0 for the primal QP. It is easy to see how the dual problem and the optimality conditions of the dual QP can be related to the primal QP and its KKT-conditions. This is also in compliance with the symmetry of the primal-dual relationship, taking the dual of the dual problem recovers the primal problem (Nocedal & Wright, 1999, p. 360).

Problem 2

This problems illustrates how solution of the equality constrained convex quadratic program scales with problem size and factorization method applied.

Consider the convex quadratic optimization problem

$$\min_u \quad \frac{1}{2}\sum_{i=1}^{n+1}(u_i - \overline{u})^2 \tag{2.1a}$$

$$\text{s.t.} \quad -u_1 + u_n = -d_0 \tag{2.1b}$$

$$u_i - u_{i+1} = 0 \qquad i = 1, 2, \ldots, n-2 \tag{2.1c}$$

$$u_{n-1} - u_n - u_{n+1} = 0 \tag{2.1d}$$

$\overline{u}$ and d_0 are parameters of the problem. The problem size can be adjusted selecting $n \geq 3$. Let $\overline{u} = 0.2$ and $d_0 = 1$. The constraints models a recycle system as depicted by the directed graph in Figure 1.

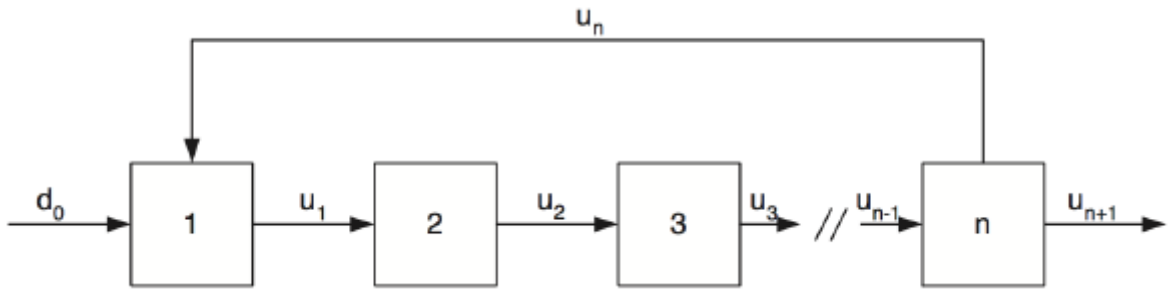


Figure 1: Directed graph representation of the constraints in (2.1). This graph represents a recycle system.

P2.1

Express the problem in matrix form, i.e. in the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \phi = \tfrac{1}{2} x' H x + g' x \\ \text{s.t.} \quad & A' x = b \end{aligned} \tag{2.2}$$

Let $n = 10$. What is x, H, g, A, and b.

First H is determined. It's given in the problem that $n = 10$. Expanding the expression in (2.1a) gives:

$$\frac{1}{2} \sum_{i=1}^{11} (u_i - \overline{u})^2 = \frac{1}{2} \sum_{i=1}^{11} \left(u_i^2 - 2 u_i \overline{u} + \overline{u}^2 \right) \tag{2.3}$$

$$= \frac{1}{2} \left(11 \overline{u}^2 + \sum_{i=1}^{11} \left(u_i^2 - 2 u_i \overline{u} \right) \right) \tag{2.4}$$

Clearly one can compare (2.4) with (2.2), the scalar does not change the minimisation problem and is ignored, therefore it follows directly that:

$$H = I, \quad g = -\overline{u} \cdot \mathbf{1} \tag{2.5}$$

Where $H \in \mathbb{R}^{11 \times 11}$, I being equal to the identity matrix, and $g \in \mathbb{R}^{11}$, with $\mathbf{1}$ being a vector of ones. From the constraints of (2.1), b is found from the RHS and A is defined by the variables u_i, notice A will be transposed to fit with (2.1), this leads

to:

$$b = \begin{bmatrix} -d_0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad A' = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \end{bmatrix} \quad x = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_8 \\ u_{10} \\ u_{11} \end{bmatrix} \tag{2.6}$$

All with the dimensions $x \in \mathbb{R}^{11}$, $A \in \mathbb{R}^{11\times 10}$ and $b \in \mathbb{R}^{10}$.

P2.2

> What is the Lagrangian function and the first order optimality conditions for the problem? Explain why the optimality conditions are both necessary and sufficient for this problem.

Once again the stated problem in (2.1) is a QP with equality constraints, therefore the Lagrangian is the following:

$$\mathcal{L}(x, \lambda) = \frac{1}{2} x' H x + g' x - \lambda' \left(A' x - b \right) \tag{2.7}$$

The first-order necessary conditions is given by Theorem 12.1 in (Nocedal & Wright, 1999, p.321). The gradient of the gradient is determined and set to 0:

$$\Delta_x \mathcal{L}(x, \lambda) = Hx + g - A\lambda = 0 \tag{2.8}$$

Finally the equality constraints, which also satisfy (12.34e) if equal to 0:

$$A'x - b = 0 \tag{2.9}$$

This can also be written using the KKT system:

$$\begin{bmatrix} H & -A \\ -A' & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = - \begin{bmatrix} g \\ b \end{bmatrix} \tag{2.10}$$

Here $\lambda \in \mathbb{R}^{10}$ based on the number of constraints seen from A.

If Lemma 16.1 in (Nocedal & Wright, 1999, p. 452-453) is satisfied then there is a unique vector pair (x^*, λ^*), which satisfies the KKT conditions, both the necessary and the sufficient condition. Because H is the identity matrix, H is positive definite, furthermore it's clear that A' has full row rank based on the diagonal elements in each row. Since we are dealing with a equality constrained QP, it's convex. This means that there is a unique solution, which is a minimum. This will make the first order optimality conditions necessary and sufficient.

P2.3

> Make a Matlab function that constructs H, g, A, and b as function of $n, \overline{u}$, and d_0.

The script in Listing 4 will take $n, \bar{u}, d_0$ as input and construct H, g, A, b:

Listing 4 ConstructEqQP constructs H, g, A, and b

```
function [H, g, A, b] = ConstructEqQP(n,ub,d0,sparsity)
% Construct g
g = -ones(n+1,1)*ub;
e1 = ones(n,1);
e2 = zeros(n,1);
% Construct H and A sparse or dense
if sparsity == 1
    H = sparse(eye(n+1,n+1));
    A = spdiags([e1 -e1 e2],[-1 0 1],n,n+1); %sparse
elseif sparsity == 0
    H = eye(n+1,n+1);
    A = spdiags([e1 -e1 e2],[-1 0 1],n,n+1);
    A = full(A); % dense
else
    disp('Choose 0 or 1 sparsity');
end
% Finalizing A
A(1,n) = 1;
A(n,n+1) = -1;
A = A';
% Constructing b
b = zeros(n,1);
b(1) = -d0;
```

P2.4

Make a Matlab function that constructs the KKT-matrix as function of $n, \overline{u}$, and d_0.

Below, in Listing 5, the function to construct the KKT-system matrix is seen.

Listing 5 KKTSystem constructs the KKT-matrix

```
function [K,h] = KKTSystem(n,u,d,sparsity)

% Construct H,g,A,b
[H, g, A, b] = ConstructEqQP(n,u,d,sparsity);
%Get dimensions of A
[~,a2] = size(A);
% Construct KKT matrix
K=[H , -A; -A' , zeros(a2,a2)];
%Create RHS
h = [-g;-b];
```

P2.5

Make a Matlab function that solves (2.1) using an LU factorization.

LU factorization assumes no properties to a quadratic matrix, it therefore handles indefinite and non-symmetric matrices rather well (Bagterp Jørgensen, 2019b). The code can be seen in Listing 6. Due to the flexibility of the LU factorization it can also be slower than other more restrictive methods.

Listing 6 Performs LU factorization on the KKT matrix using an inbuilt MATLAB function.

```
function [x,lambda] = KKTLUSolve(n,H,g,A,b,K,h)
% Make LU Factorization
[L,U,p] = lu(K,'vector');
z(p) = U\(L\h(p));
%Get solution and Langrangian multipliers
x=z(1:(n+1));
lambda=z((n+2):end);
```

P2.6

> Make a Matlab function that solves (2.1) using an LDL factorization.

The LDL factorization is a variant of Cholesky factorisation, seen in Listing 7. It requires that the matrix is symmetric and the method handles indefinite matrices well. Since the KKT-matrix is constructed satisfying symmetry, see (2.10), LDL factorization can be used.

Listing 7 Performs LDL factorization on the KKT matrix

```
function [x,lambda] = KKTLDLSolve(n,H,g,A,b,K,h)
z = zeros(2*n+1,1);
% Use LDL factorization
[L,D,p] = ldl(K,'lower','vector');
z(p) = L'\(D\(L\h(p)));
%Get solution and Langrangian multipliers
x=z(1:(n+1));
lambda=z((n+2):end);
```

P2.7

> Make a Matlab function that solves (2.1) using the Null-Space procedure based on QR-factorizations.

The algorithm for the Null-Space factorization can be seen in Listing (8). The Null-Space function looks at (2.10), where $A \in \mathbb{R}^{n \times m_a}$. The null-space method is based on QR-factorisation. It uses this factorisation in order to reduce the size of a problem from $(n+m) \times (n+m)$ to $(n-m) \times (n-m)$. The null-space method is very efficient when $n-m$ is small, but can also be computationally expensive (Rees & Scott, 2014). It does not require regularity of the KKT matrix in contrast to the range-space method.

P2.8

> Make a Matlab function that solves (2.1) using the Range-Space procedure.

The Range-Space method applies for symmetric positive definite matrices. If the matrix is well-conditioned and easy to invert, or if the inverted matrix is already

Listing 8 Performs Null-Space factorization on the constructed QP matrices.

```
function [x,lambda] = KKTNSSolve(n,H,g,A,b,K,h)
%Perform null space factorization
[Q,Rbar] = qr(A);
m1 = size(Rbar,2);
Q1 = Q(:,1:m1); Q2 = Q(:,m1+1:n+1); R = Rbar(1:m1,1:m1);
xy = R'\b;
xz = (Q2'*H*Q2)\(-Q2'*(H*Q1*xy+g));
x = Q1*xy+Q2*xz;
lamb = R\(Q1'*(H*x+g));
z = vertcat(x,lamb);
%Get solution and Langrangian multipliers
x=z(1:(n+1));
lambda=z((n+2):end);
```

known, the Range-Space method is very efficient. It also handles a low number of equality constraints well (Bagterp Jørgensen, 2019e). The method can be observed in Listing 9.

Listing 9 Performs Range-Space factorization on the constructed QP matrices.

```
function [x,lambda] = KKTRSSolve(n,H,g,A,b,K,h)
%Perform range-space factorization using a cholesky factorization
L = chol(H,'lower');
v = L'\(L\g);
Ha = A'*inv(L)'*inv(L)*A;
La = chol(Ha, 'lower');
lamb = La'\(La\(b+A'*v));
x = L'\(L\(A*lamb-g));
z = vertcat(x,lamb);
%Get solution and Langrangian multipliers
x=z(1:(n+1));
lambda=z((n+2):end);
```

P2.9, P2.10, P2.11, P2.12 & P2.13

Looking at the performance of the different QP solvers over an average of 40 iterations for the non-sparse solutions and around 80 to 800 iterations for the sparse, due to CPU time being rather sensitive. Both the wall-time and CPU-time can be seen in Figure 2. All solvers are close in computational cost when looking at the dense matrices. Null-space is the fastest, even though it can be computationally expensive, the gain is large, due to $n - m = 1$. The range space seem to compete fairly close

with the LU and LDL factorization. It seems LU is faster at the start, but start getting slow for high dimensions $n > 800$. Another surprise is that since H^{-1} is in this case $I^{-1} = I$, therefore very easy to invert, the range-space is supposed to do rather well, but it is not significantly faster than LU or LDL factorisation. Perhaps because of computational overhead.

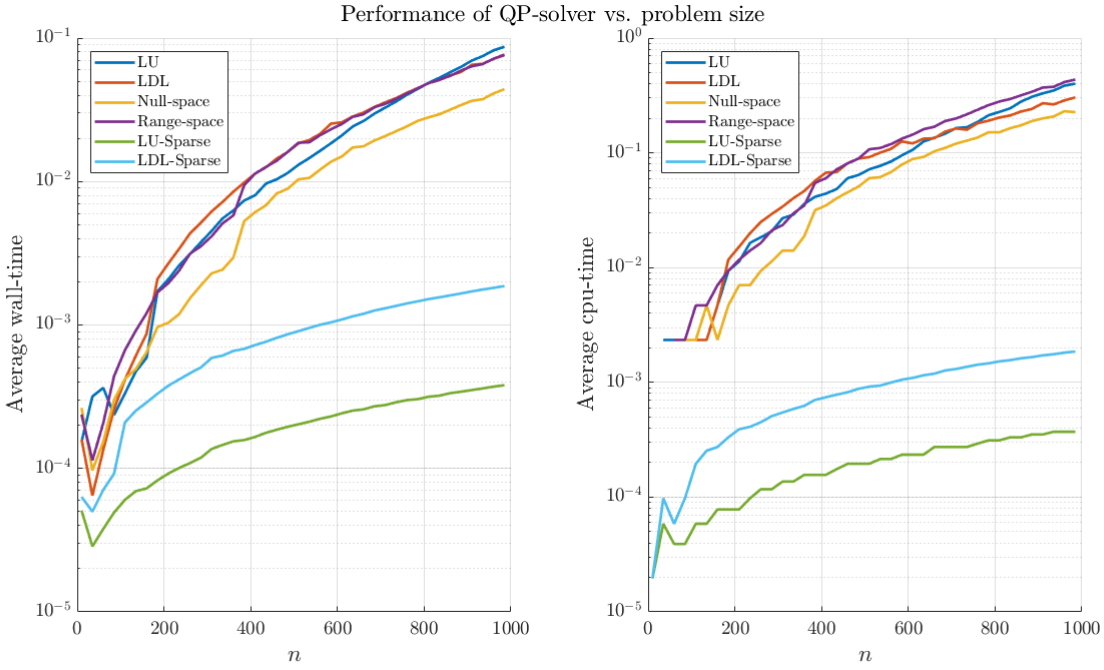


Figure 2: The performance for each of the implemented factorizations and the sparse LU and LDL-factorisation against problem size, n.

Clearly from Figure 3 the KKT matrix is full of zeros, with 503 nonzeros, accounting for 5.03% of the values. This will be utilized. Therefore A and the KKT matrix are made sparse. MATLAB is very good at handling sparse matrices, with optimized matrix operations, efficiently dealing with sparsity. From Figure 2 a large improvement to the computational cost can be observed. With LDL-sparse and LU-sparse being far superior. Still, surprisingly, the LU factorisation is faster, handling the structure better.

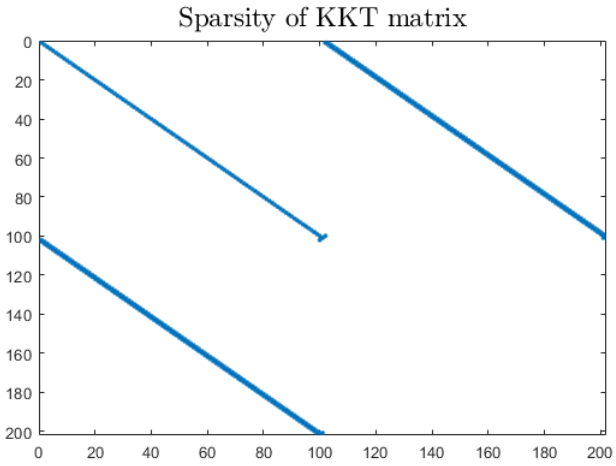


Figure 3: Sparsity of the KKT-matrix, lines indicate indices with a value different to 0. For $n = 100$ only 503 of the values are non-zero, this accounts for 5.03%

Problem 3

This exercise illustrates use of quadratic programming in a Financial application. By diversifying an investment into several securities it may be possible to reduce risk without reducing return. Identification and construction of such portfolios is called hedging. The Markowitz Portofolio Optimization problem is very simple hedging problem for which Markowitz was awarded the Nobel Price in 1990.
Consider a Financial market with 5 securities.

Security	Covariance					Return
1	2.30	0.93	0.62	0.74	-0.23	15.10
2	0.93	1.40	0.22	0.56	0.26	12.50
3	0.62	0.22	1.80	0.78	-0.27	14.70
4	0.74	0.56	0.78	3.40	-0.56	9.02
5	-0.23	0.26	-0.27	-0.56	2.60	17.68

P3.1

For a given return, R, formulate Markowitz' Portfolio optimization problem as a quadratic program.

The goal of the Markowitz problem is to find a portfolio that minimises the risk (variance) for the expected return. From the information given in problem, the minimum and maximum return, R, is known. Therefore an alternative formulation

of the Markowitz problem is used, (13a - 13d) in (Bagterp Jørgensen, 2018a, p. 8):

$$\min_{x \in \mathbb{R}^n} \quad F(x) = \frac{1}{2} x' H x \tag{3.1a}$$

$$\text{s.t.} \quad \mu' x = R \tag{3.1b}$$

$$\sum_{i=1}^{n} x_i = 1 \tag{3.1c}$$

$$x \geq 0 \tag{3.1d}$$

Where $R \in \begin{bmatrix} \min_i(r_i) & \max_i(r_i) \end{bmatrix}$, here r_i is the return of security i. From (3.1) it is seen that the expected return is given by:

$$E\{R\} = \mu' x$$

Hence (3.1b) and also that x_i is the fraction invested in security i, hence the equality constraint (3.1c) and inequality of (3.1d). Finally $H = \Sigma$, where Σ is the covariance matrix. In this case:

$$\Sigma = \begin{bmatrix} 2.30 & 0.93 & 0.62 & 0.74 & -0.23 \\ 0.93 & 1.40 & 0.22 & 0.56 & 0.26 \\ 0.62 & 0.22 & 1.80 & 0.78 & -0.27 \\ 0.74 & 0.56 & 0.78 & 3.40 & -0.56 \\ -0.23 & 0.26 & -0.27 & -0.56 & 2.60 \end{bmatrix}, \quad \mu = \begin{bmatrix} 15.10 \\ 12.50 \\ 14.70 \\ 9.02 \\ 17.68 \end{bmatrix}$$

Now the problem can be set up based on the standard form of the QP.

$$H = \Sigma, \quad b = [R\, 1]\prime, \quad A = [\mu\ \vec{1}], \quad C = I, \quad d = \vec{0}, \quad g = \vec{0}$$

Where C and d comes from the inequality constraints. In Listing (10) the code to constructing the Markowitz problem as a QP for a given return R can be observed.

P3.2

> What is the minimal and maximal possible return in this financial market?

The maximal possible return from the QP is defined by $R \in \begin{bmatrix} \min_i(r_i) & \max_i(r_i) \end{bmatrix}$. From the problem definition it's seen that $\min_i(r_i) = 9.02$ and $\max_i(r_i) = 17.68$.

Listing 10 Constructs the Markowitz problem.

```
function [H,g,A,b,C,d] = ConstructMarkowitz(R, rf);
%Construct minimizer
if rf==0
    H=[2.3, 0.93, 0.62, 0.74, -0.23;
        0.93, 1.4, 0.22, 0.56, 0.26;
        0.62, 0.22, 1.8, 0.78, -0.27;
        0.74, 0.56, 0.78, 3.4, -0.56;
        -0.23, 0.26, -0.27, -0.56, 2.6];
    A = [15.1, 12.5, 14.7, 9.02, 17.68;
    1 1 1 1 1]';
elseif rf==1
   H = [ 2.30 0.93 0.62 0.74 -0.23 0;
        0.93 1.40 0.22 0.56 0.26 0;
        0.62 0.22 1.80 0.78 -0.27 0;
        0.74 0.56 0.78 3.40 -0.56 0;
        -0.23 0.26 -0.27 -0.56 2.60 0;
        0 0 0 0 0 0];
    A = [15.10 12.50 14.70 9.02 17.68 2; 1 1 1 1 1 1]';
end
dim2 = size(H,1);
g=zeros(dim2,1);
%Construct equality constraint
dim1 = size(A,1);
b = [R;1];
%Construct inequality
C=eye(dim1);
d = zeros(dim1,1);
end
```

P3.3

> Use `quadprog` to find a portfolio with return, $R = 10.0$, and minimal risk. What is the optimal portfolio and what is the risk (variance)?

To use `quadprog` the inequality needs to be flipped to match the specified QP of the `quadprog` function, this is done by setting C to $-C$. The code to solving the Markowitz QP can be seen in Listing (11):

Listing 11 Solves the Markowitz QP for $R = 10$.

```
%% Opgave 3.3
%Specify return
R=10;
[H,g,A,b,C,d] = ConstructMarkowitz(R,0);
%Solve QP using quadprog
xhat = quadprog(H,[],-C,d,A',b);

%find variance
xhat'*H*xhat;

%% Opgave 3.4
colors = [0, 0.4470, 0.7410; 0.8500, 0.3250, 0.0980 ; 0.9290, 0.6940, 0.1250;
    0.4940, 0.1840, 0.5560; 0.4660, 0.6740, 0.1880; 0.3010, 0.7450, 0.9330];
%define max and minimum return
max1 = 17.68;
min1 = 9.02;
returns = [min1 9.5:0.5:17 max1];
xhats = zeros(5, length(returns));
risk = zeros(length(returns),1);

for i=1:length(returns)
    [H,g,A,b,C,d] = ConstructMarkowitz(returns(i),0);
    xhat = quadprog(H,[],-C,d,A',b);
    xhats(:,i)=xhat;
    risk(i) = xhat'*H*xhat;
end

%Risk as function of return
figure(1);
plot(returns, risk,'-*b', 'LineWidth', 1)
xlabel('Return','FontSize',11)
ylabel('Risk','FontSize',11)
title('Risk as function of Return', 'FontSize', 10)

% Optimal portfolio as function of return
figure(2);
h1=subplot(4,2,[1,2]);
bar(returns, xhats', 'stacked')
set(gca,'FontSize',10)
legend('Security 1','Security 2','Security 3','Security 4', 'security 5','Location','EastOutside'
xlim([min1-0.3, max1+0.3])
ylim([0,1.1])
xticks([9.02, 10:16, 17.68])
h2=subplot(4,2,[3 4, 5 ,6,7,8]);
plot(returns, xhats(1,:),'-*b' ,'LineWidth', 1)
hold on;
for i=2:5
    plot(returns, xhats(i,:),'-*', 'LineWidth', 1,'color', colors(i,:))
    set(gca,'FontSize',10)
end
hold off;
```

The results of the Markowitz QP giving the optimal fraction of securities for $R = 10$ is:

$$\hat{x} = [0 \quad 0.2816 \quad 0 \quad 0.7184 \quad 0]'$$

Clearly it can be seen that the above result sums to 1, in compliance to (3.1). Furthermore the risk is found by:

$$V[R] = \hat{x}' H \hat{x} = 2.0923$$

This means for a $R = 10$, a minimal risk of $V[R] = 2.0923$ is achieved when investing 28.16% of capital in security 2 and 71.84% in security 4.

P3.4

> Compute the efficient frontier, i.e. the risk as function of the return. Plot the efficient frontier as well as the optimal portfolio as function of return.

In Figure 4 it can be seen how the risk depends on the expected return, also known as the efficient frontier. Notice how both a low and high return have a high risk for this financial market. For Figure 5 the optimal portfolio can be seen for a given return, R, it's seen that for the maximum return or minimum return one should invest in a single security, however a combination of the securities in the portfolio seem optimal as a too high risk may not be desired. This is in accordance to the advice of Markowitz of diversifying portfolio to minimise risk.

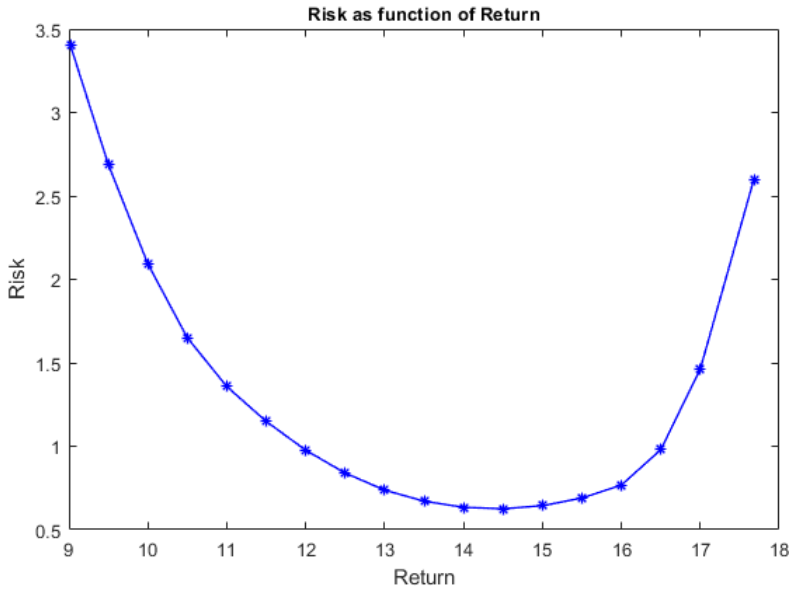


Figure 4: The efficient frontier. Around $R = 14.52$ the risk is minimized.

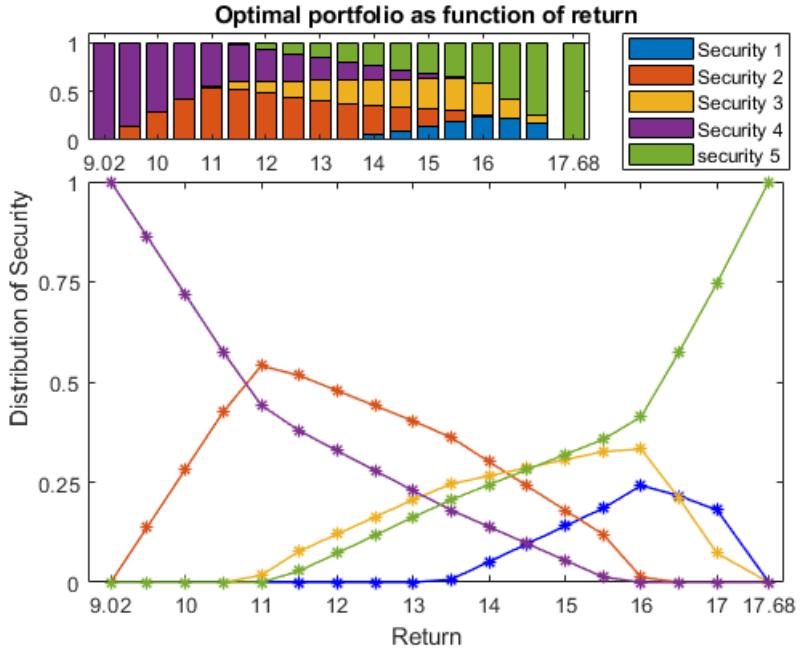


Figure 5: The optimal portfolio as function of return. The bar figure visualizes the fraction of each security.

Problem 3 Part 2

In the following we add a risk free security to the financial market. It has return $r_f = 2.0$.

P3.1.2

What is the new covariance matrix and return vector.

When adding a risk free security, this is the same as adding a variance free security with no correlations between other securities, the new market therefore becomes:

Security	Covariance						Return
1	2.3	0.93	0.62	0.74	-0.23	0	15.10
2	0.93	1.4	0.22	0.56	0.26	0	12.5
3	0.62	0.22	1.8	0.78	-0.27	0	14.7
4	0.74	0.56	0.78	3.4	-0.56	0	9.02
5	-0.23	0.26	-0.27	-0.56	2.6	0	17.68
6	0	0	0	0	0	0	2

P3.2.2

Compute the efficient frontier, plot it as well as the (return,risk) coordinates of all the securities. Comment on the effect of a risk free security. Plot the optimal portfolio as function of return.

In Figure 6 the efficient front can be seen. Notice that there is no risk when looking at the portfolio with $R = 2$. This is because the portfolio simply contain the risk free security, also seen as a light blue dot in the figure. The introduction of the new risk free security will make the new minimum return to 2.00 and will reduce the overall risk when combined with one of the other uncertain securities. This can be seen in Figure 6, except for the last portfolio, this is because if we want maximum return, all capital should be invested in security 5 with the maximal return and therefore security 6 will not be included in the portfolio.

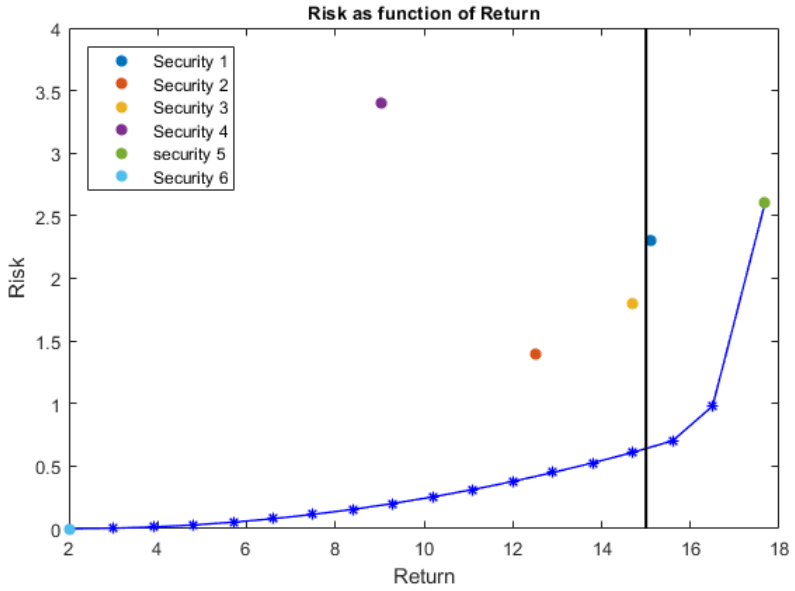


Figure 6: The efficient frontier. The black line is for $R = 15$ and the colored dots are the $(return, risk)$ coordinates

In Figure 7 the distribution of the securities in each portfolio can be observed. Once again securities with high risk, low return such as security 4 will not be invested heavily into in any of the portfolios. Once again if no risk is desired all capital should be invested into the risk free security, security 6. The fraction of security 6 in the portfolio is slowly decreasing, this is because of the low return, therefore if one wants a higher return, more risk has to be introduced and less should be invested in security 6.

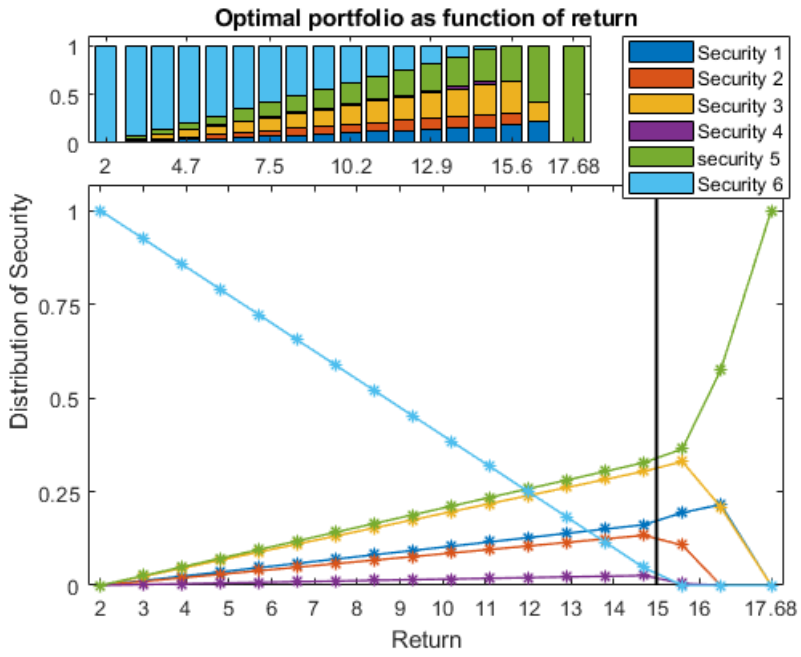


Figure 7: The optimal portfolio as function of return. The black line is for $R = 15$.

P3.3.2

> What is the minimal risk and optimal portfolio giving a return of $R = 15.00$. Plot this point in your optimal portfolio as function of return as well as on the efficient frontier diagram.

The optimal portfolio can be calculated using `quadprog` once again including the risk free security, here the fraction of each portfolio is given by:

$$\hat{x} = [0.1655 \quad 0.1365 \quad 0.3115 \quad 0.0266 \quad 0.3352 \quad 0.0247]'$$

The risk is found to be

$$V[R] = 0.6383$$

The risk is found to be slightly lower for $R = 15$ including the risk free security compared to when not including it, from Figure 4. The point of the optimal portfolio can be seen in Figure 6 and Figure 7.

Problem 4

P4.1 & P4.2

Write an interior-point algorithm on paper for solution of the convex quadratic program

$$\min_{x\in\mathbb{R}^n} \quad \phi = \frac{1}{2}x'Hx + g'x \tag{4.1a}$$
$$\text{s.t.} \quad A'x = b \tag{4.1b}$$
$$C'x \geq d \tag{4.1c}$$

and explain the Primal-Dual Interior Point Algorithm for convex QPs.

Please notice that P4.1 and P4.2 will overlap and is therefore combined. Primal-dual interior point methods can be described as a modified Newton's method solving a perturbed KKT system. The Lagrange function is given by:

$$L(x,y,z) = \frac{1}{2}x'Hx + g'x - y'\left(A'x - b\right) - z'\left(C'x - d\right)$$

This leads to the necessary and sufficient KKT conditions, see (Nocedal & Wright, 1999, Theorem 16.4).

$$\begin{aligned}
\nabla_x L(x,y,z) &= Hx + g - Ay - Cz = 0 \\
\nabla_y L(x,y,z) &= -\left(A'x - b\right) = 0 \\
\nabla_z L(x,y,z) &= -\left(C'x - d\right) \leq 0 \\
z &\geq 0 \\
\left(C'x - d\right)_i z_i &= 0 \quad i = 1,2,\ldots,m_c
\end{aligned} \tag{4.2}$$

Introducing a slack variable $s \geq 0$ and $s = C'x - d$ the optimality conditions can be rewritten to

$$\begin{aligned}
r_L &= Hx + g - Ay - Cz = 0 \\
r_A &= -A'x + b = 0 \\
r_C &= -C'x + s + d = 0 \\
s_i z_i &= 0 \quad i = 1,2,\ldots,m_c
\end{aligned} \quad , \tag{4.3}$$

where $z \geq 0$ and $s \geq 0$. The last condition can simply be written as $s_i z_i = r_{SZ} =$

$SZe = 0$, where:

$$S = \text{diag}(s_1, s_2, \ldots, s_{m_c}), \quad Z = \text{diag}(z_1, z_2, \ldots, z_{m_c}), \quad e = (1, 1, \ldots, 1)^T \tag{4.4}$$

The conditions are not complete, a modification is to set $s_i z_i = \tau$ rather than $s_i z_i = 0$. As explained in (Bagterp Jørgensen, 2019a, p. 10) a bias is introduced to the search direction toward the interior of the non-negative orthant $(z, s) \geq 0$.A search direction that reduces the duality measure μ will be chosen, which is the average value of the pairwise product $s_i z_i$

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \lambda_i = \frac{s'z}{m_c}, \tag{4.5}$$

Hence $\tau = \sigma\mu$ that is for some $\sigma \in [0, 1]$. Now when $\sigma > 0$ one is usually able to take longer Newton steps before violating the bound $(z, s) \leq 0$. Therefore the duality measure will use the pairwise product, (4.5), to decide if a computed step length has reduced the product enough. Otherwise σ will correct this, simply by scaling itself (Nocedal & Wright, 1999, p 396).

The above perturbed KKT conditions can then be used to express a non-linear system of equations

$$F(x_\tau, y_\tau, z_\tau, s_\tau) = \begin{bmatrix} r_L \\ r_A \\ r_C \\ r_{SZ} \end{bmatrix} = \begin{bmatrix} Hx_\tau + g - Ay_\tau - Cz_\tau \\ -A'x_\tau + b \\ -C'x_\tau + s_\tau + d \\ S_\tau Z_\tau e \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \sigma\mu e \end{bmatrix} = 0, \tag{4.6}$$

with $(z_\tau, s_\tau) \geq 0$. Now $F(x_\tau, y_\tau, z_\tau, s_\tau) = 0$ will yield the solution to (4.3). It is solved by using Newton's method, it is possible finding a global minimizer because (4.1) is a convex problem for H being positive semi-definite. By Newton's method (Nocedal & Wright, 1999, chapter 2) a search direction is found by the following system:

$$J(x, y, z, s) = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = -F(x, y, z, s), \tag{4.7}$$

J is the Jacobian of $F(x, y, z, s)$ and Δ indicating search directions. From this the

following system is found:

$$\begin{bmatrix} H & -A & -C & 0 \\ -A' & 0 & 0 & 0 \\ -C' & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = - \begin{bmatrix} r_L \\ r_A \\ r_C \\ r_{SZ} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \sigma\mu e \end{bmatrix} = \begin{bmatrix} -r_L \\ -r_A \\ -r_C \\ -r_{SZ} + \sigma\mu e \end{bmatrix} \tag{4.8}$$

A step can now be taken:

$$\begin{bmatrix} x_\tau \\ y_\tau \\ z_\tau \\ s_\tau \end{bmatrix} \leftarrow \begin{bmatrix} x_\tau \\ y_\tau \\ z_\tau \\ s_\tau \end{bmatrix} + \alpha \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} \quad (z_\tau, s_\tau) > 0 \tag{4.9}$$

For a step length $\alpha \in]0, \alpha_{\max}]$ determined as the largest step one can take such that the following is satisfied:

$$\begin{bmatrix} z^k \\ s^k \end{bmatrix} + \alpha^k_{\max} \begin{bmatrix} \Delta z^k \\ \Delta s^k \end{bmatrix} \geq 0 \tag{4.10}$$

First an outline of the primal-dual algorithm is introduced, see Algorithm 1. This gives an idea of the primal-dual framework. The algorithm is not computationally optimal. A good stopping criteria is missing and the LDL decomposition, is a computational expensive step, only used once in each of the iterations. Therefore it is desirable to converge to a solution faster by utilising the LDL decomposition in step 7 in Algorithm 1. Both in a predictor and corrector of the optimisation search direction. Finally a proper stopping criteria has to be derived. The ideas will briefly be discussed, and then the primal-dual predictor-corrector interior-point algorithm will be implemented.

To deal with the challenges Mehrotra made a few modifications (Bagterp Jørgensen, 2019a, p. 15).

1. Addition of a corrector step to the search direction
2. Adaptive choice of the centering parameter.

The method will then consist of a predictor step, adaptive choice of σ, centering

step and a corrector step. Looking at the predictor step.

$$\text{Predictor Step:} \quad \begin{bmatrix} H & -A & -C & 0 \\ -A' & 0 & 0 & 0 \\ -C' & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x^{aff} \\ \Delta y^{aff} \\ \Delta z^{aff} \\ \Delta s^{aff} \end{bmatrix} = - \begin{bmatrix} r_L \\ r_A \\ r_C \\ r_{SZ} \end{bmatrix}, \tag{4.11}$$

Algorithm 1 Primal-Dual Interior-Point for convex QP

procedure QPIPPD
  Given (x^0, y^0, z^0, s^0) such that $(z^0, s^0) > 0$
  while not STOP **do**
    Compute the duality measure: $\mu = \frac{s^{k\prime} z^k}{m_c}$
    Set centering parameter: $\sigma^k \in [0, 1]$
    Compute the residuals r_L, r_A, r_C and r_{SZ} as described in (4.6).
    Solve

$$\begin{bmatrix} H & -A & -C & 0 \\ -A' & 0 & 0 & 0 \\ -C' & 0 & 0 & I \\ 0 & 0 & S^k & Z^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta z^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} -r_L \\ -r_A \\ -r_C \\ -r_{SZ} + \sigma^k \mu^k e \end{bmatrix}$$

    Choose step-length $\alpha^k \in]0, \alpha^k_{\max}]$ with

$$\begin{bmatrix} z^k \\ s^k \end{bmatrix} + \alpha^k_{\max} \begin{bmatrix} \Delta z^k \\ \Delta s^k \end{bmatrix} \geq 0$$

    Update

$$\begin{bmatrix} x^{k+1} \\ y^{k+1} \\ z^{k+1} \\ s^{k+1} \end{bmatrix} \leftarrow \begin{bmatrix} x^k \\ y^k \\ z^k \\ s^k \end{bmatrix} + \alpha^k \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta z^k \\ \Delta s^k \end{bmatrix}$$

  end while
end procedure

Here we solve for an affine scaling direction and an affine step length α^{aff}. The duality gap of the affine step is found by:

$$\mu^{aff} = \frac{\left(z + \alpha^{aff} \Delta z^{aff}\right)' \left(s + \alpha^{aff} s^{aff}\right)}{m_c} \tag{4.12}$$

Now if the duality gap is close to 0, then we make the centering parameter, σ, small.

Otherwise $\sigma = 1$. Where the centering parameter, σ, can be calculated by:

$$\sigma = \left(\frac{\mu^{aff}}{\mu} \right)^3 \tag{4.13}$$

The centering parameter decides by how much the search direction needs to be centered by the predictor step. Finally α^{aff} is defined as the maximum step size for which the complementarity condition is satisfied:

$$\begin{bmatrix} z \\ s \end{bmatrix} + \alpha^{aff} \begin{bmatrix} \Delta z \\ \Delta s \end{bmatrix} \geq 0 \tag{4.14}$$

When the centering parameter has been found, the three directions from the corrector, affine and centering step will be combined. Where the correcter is found from:

$$\text{Corrector Step:} \quad \begin{bmatrix} H & -A & -C & 0 \\ -A' & 0 & 0 & 0 \\ -C' & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x^{cor} \\ \Delta y^{cor} \\ \Delta z^{cor} \\ \Delta s^{cor} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -\Delta S^{aff} \Delta Z^{aff} e \end{bmatrix}, \tag{4.15}$$

And the centering is given by:

$$\text{Center Step:} \quad \begin{bmatrix} H & -A & -C & 0 \\ -A' & 0 & 0 & 0 \\ -C' & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x^{cen} \\ \Delta y^{cen} \\ \Delta z^{cen} \\ \Delta s^{cen} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \sigma \mu e \end{bmatrix}, \tag{4.16}$$

This leads to Mehrotas Direction:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = \begin{bmatrix} \Delta x^{aff} \\ \Delta y^{aff} \\ \Delta z^{aff} \\ \Delta s^{aff} \end{bmatrix} + \begin{bmatrix} \Delta x^{cor} \\ \Delta y^{cor} \\ \Delta z^{cor} \\ \Delta s^{cor} \end{bmatrix} + \begin{bmatrix} \Delta x^{cen} \\ \Delta y^{cen} \\ \Delta z^{cen} \\ \Delta s^{cen} \end{bmatrix}, \tag{4.17}$$

Solving the resulting system

$$\begin{bmatrix} H & -A & -C & 0 \\ -A' & 0 & 0 & 0 \\ -C' & 0 & 0 & I \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_L \\ -r_A \\ -r_C \\ -r_{SZ} - \Delta X^{aff} \Delta \Lambda^{aff} e + \sigma \mu e \end{bmatrix}. \tag{4.18}$$

Finally the step updating the values:

$$\begin{bmatrix} x \\ y \\ z \\ s \end{bmatrix} \leftarrow \begin{bmatrix} x + \eta\alpha\Delta x \\ y + \eta\alpha\Delta y \\ z + \eta\alpha\Delta z \\ s + \eta\alpha\Delta s \end{bmatrix}, \tag{4.19}$$

Where once again largest α has to satisfy (4.20) and $\eta \in]0, 1[$

$$\begin{bmatrix} z \\ s \end{bmatrix} + \alpha \begin{bmatrix} \Delta z \\ \Delta s \end{bmatrix} \geq 0 \tag{4.20}$$

From this it is seen that the LDL decomposition of the system matrix happends twice, as opposed to once as before. Only introducing a slight increase in computational cost, compared to the simple primal-dual interior point algorithm, because it is only necessary to compute a few more products. The small overhead is expected to be insignificant, compared to the expected faster convergence of the modified method.

Hence this leads to the stopping criteria:

$$\begin{aligned} \|r_L\|_\infty &= \|Hx + g - Ay - Cz\|_\infty \leq \epsilon \cdot \max\left\{1, \| \begin{bmatrix} H & g & A & C \end{bmatrix} \|_\infty\right\} \\ \|r_A\|_\infty &= \left\|b - A'x\right\|_\infty \leq \epsilon \cdot \max\left\{1, \| \begin{bmatrix} A' & b \end{bmatrix} \|_\infty\right\} \\ \|r_C\|_\infty &= \left\|d + s - C'x\right\|_\infty \leq \epsilon \cdot \max\left\{1, \| \begin{bmatrix} I & d & C' \end{bmatrix} \|_\infty\right\} \end{aligned} \tag{4.21}$$

with $\mu \leq \epsilon \cdot 10^{-2} \cdot \mu^0$, where ϵ is some chosen tolerance.
A disadvantage of Mehrotra's primal-dual predictor-corrector interior-point algorithm is that while it has proven to be much faster than the simple primal-dual interior-point algorithm in both the computational time and the number of iterations, the global convergence or polynomial complexity can't be theoretically guaranteed (Cartis, 2009).

Further modifications of Mehrotra's primal-dual predictor-interior algorithm can be made. Especially exploiting the structure of linear systems and using matrix factorisation, augmenting the system. However, this will not be further elaborated, but rather referred to (Bagterp Jørgensen, 2019a).

P4.3

Implement the Primal-Dual Interior-Point Algorithm for this convex quadratic program

The implementation can be found in Appendix under Problem 4.

P4.4

What is H, g, A, C, b, and d for the Markowitz Portfolio Optimization Problem with $R = 15$ and the presence of a risk-free security?

Referring Problem 3, H, g, A, C, b, and d of the Markowitz portfolio opimisation problem can be seen below:

$$H = \begin{bmatrix} 2.30 & 0.93 & 0.62 & 0.74 & -0.23 & 0 \\ 0.93 & 1.40 & 0.22 & 0.56 & 0.26 & 0 \\ 0.62 & 0.22 & 1.80 & 0.78 & -0.27 & 0 \\ 0.74 & 0.56 & 0.78 & 3.40 & -0.56 & 0 \\ -0.23 & 0.26 & -0.27 & -0.56 & 2.60 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.22}$$

$$b = \begin{bmatrix} 15 & 1 \end{bmatrix}^T, \tag{4.23}$$

$$A = \begin{bmatrix} 15.10 & 12.50 & 14.70 & 9.02 & 17.68 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T, \tag{4.24}$$

and finally $C = I$, $d = \mathbf{0}$ and $g = \mathbf{0}$.

P4.5

Test this algorithm on the Markowitz Portfolio Optimization Problem, i.e. compute the efficient frontier and optimal portfolio for the situation with a risk-free security. Do your algorithm give the same solution as QUADPROG?

Using the Primal-Dual Interior-Point implementation the optimal portfolio is

$$\hat{x} = [0.1655 \quad 0.1365 \quad 0.3115 \quad 0.0266 \quad 0.3352 \quad 0.0247]^T \tag{4.25}$$

with a minimal risk of $V[15] = 0.6383$. In figure Figure 8 the optimal portfolio of different return values can be seen.

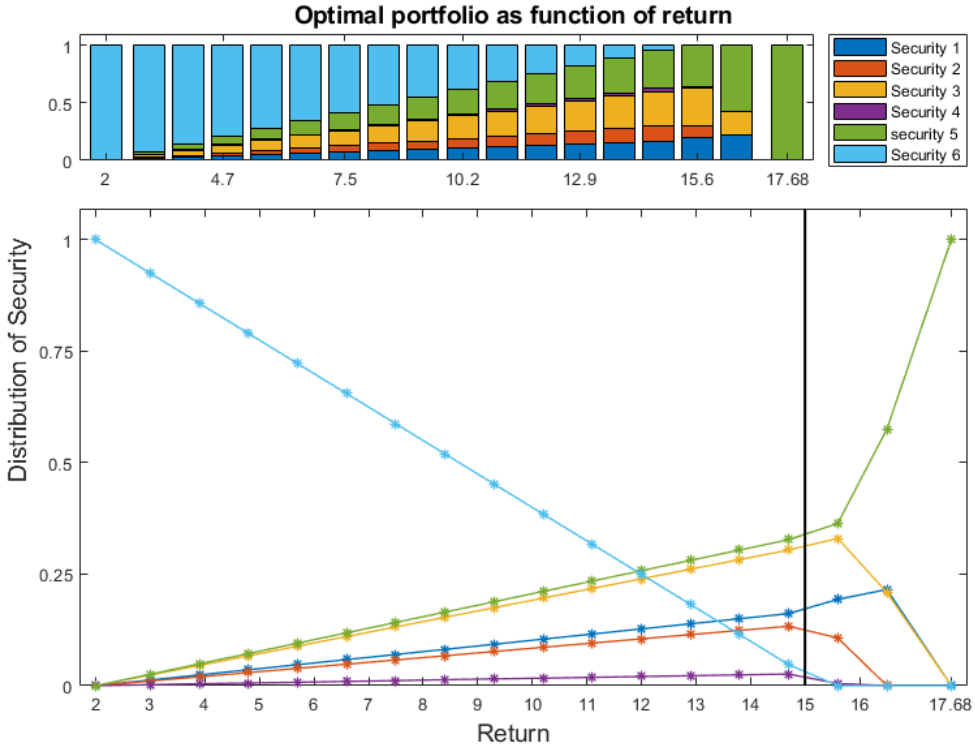


Figure 8: Optimal portfolio for different return values using the Primal-Dual Interior-Point implementation

Finally the efficient frontier can be seen in Figure 9

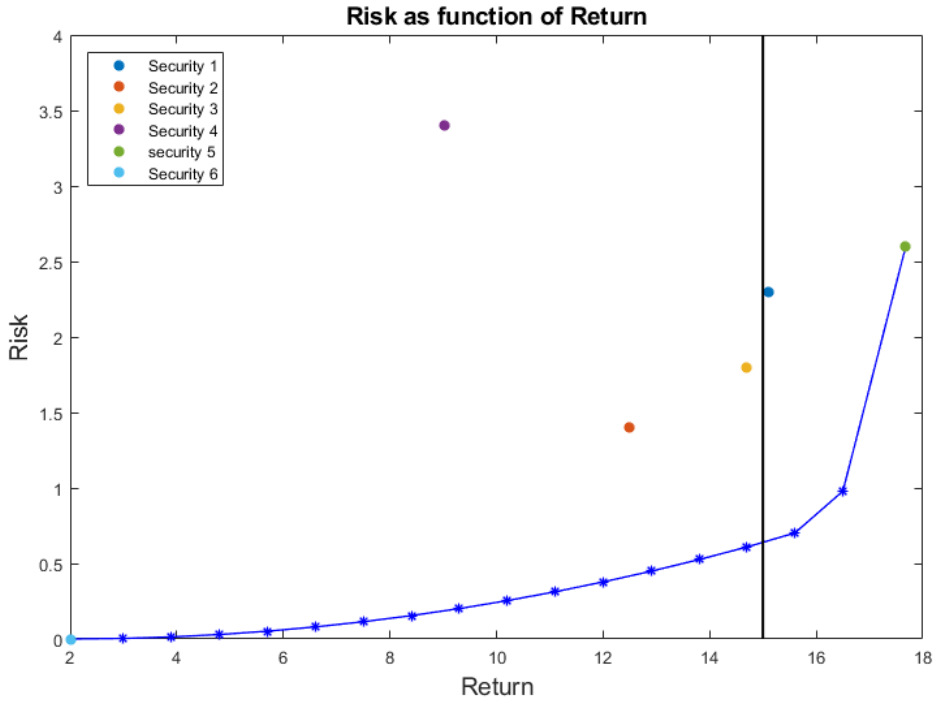


Figure 9: The efficient frontier using the Primal-Dual Interior-Point implementation

It is clear to see that the results match exactly the results found in Problem 3 using MATLAB's `quadprog`.

Problem 5

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.

P14.15

Program Algorithm 14.3 in Matlab. Choose $\eta = 0.99$ uniformly in (14.38). Test your code on a linear programming problem (14.1) generated by choosing A randomly, and then setting x, s, b, and c as follows:

$$
\begin{aligned}
x_i &= \begin{cases} \text{random positive number} & i = 1, 2, \ldots, m \\ 0 & i = m+1, m+2, \ldots, n \end{cases} \\
s_i &= \begin{cases} \text{random positive number} & i = m+1, m+2, \ldots, n \\ 0 & i = 1, 2, \ldots, m \end{cases} \\
\lambda &= \text{ random vector} \\
c &= A^T\lambda + s \\
b &= Ax
\end{aligned}
\tag{5.1}
$$

Choose the starting point $\left(x^0, \lambda^0, s^0\right)$ with the components of x^0 and s^0 set to large positive values.

P14.15 will follow a lot of the same principles as P4, however to make it clear it will be restated with small changes for the linear case. The general formulation of problem (14.1) is given by:

$$
\min c^T x, \quad \text{subject to } Ax = b, \quad x \geq 0 \tag{5.2}
$$

where c and x are vectors in $\mathbf{R}^n$, b is a vector in $\mathbf{R}^m$, and A is an $m \times n$ matrix with full row. The dual problem for (5.2) can be formulated as (Nocedal & Wright, 1999, p. 394):

$$
\max b^T\lambda, \quad \text{subject to } A^T\lambda + s = c, \quad s \geq 0 \tag{5.3}
$$

The Lagrangian is given by

$$
\mathcal{L}(x, \lambda, s) = c^T x - \lambda(Ax - b) - s^T x \tag{5.4}
$$

Now it is possible to set up the KKT-conditions of (5.2), (5.3):

$$
\begin{aligned}
A^T\lambda + s &= c \\
Ax &= b \\
x_i s_i &= 0, \quad i = 1, 2, \ldots, n \\
(x, s) &\geq 0
\end{aligned}
\tag{5.5}
$$

Once again this can be written as non-linear system:

$$F(x,\lambda,s) = \begin{bmatrix} A^T\lambda + s - c \\ Ax - b \\ XSe \end{bmatrix} = 0, \tag{5.6}$$

where $(x,s) \geq 0$ and

$$X = \operatorname{diag}(x_1, x_2, \ldots, x_n), \quad S = \operatorname{diag}(s_1, s_2, \ldots, s_n), \quad e = (1, 1, \cdots, 1)^T, \tag{5.7}$$

Now once again a duality measure is defined as follows:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i s_i = \frac{x^T s}{n} \tag{5.8}$$

The duality measure is the average value of the pairwise products $x_i s_i, i = 1, 2, ..., n$. As a result of the pairwise product $x_i s_i = 0$ in the KKT, one wants to make the duality measure as small as possible. This leads to the stopping criteria $|\mu| < \epsilon$, where ϵ is some specified tolerance. Now just like in the previous problem (P4), Newton's method can be used to compute $(\Delta x, \Delta\lambda, \Delta s)$, this is the search directions, by the Jacobian:

$$J(x,\lambda,s)\begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = -F(x,\lambda,s), \tag{5.9}$$

Where J is the Jacobian of F, this leads to the following linear system:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix}\begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ -XSe \end{bmatrix}, \tag{5.10}$$

Where

$$r_b = Ax - b, \quad r_c = A^T\lambda + s - c \tag{5.11}$$

A new iterate can then be defined as:

$$(x,\lambda,s) + \alpha(\Delta x, \Delta\lambda, \Delta s) \tag{5.12}$$

Where $\alpha \in]0, 1]$. Often α is small, so to not violate the condition $(x,s) > 0$.

To make a less aggressive Newton direction, we do not aim directly for a solution, but rather for a point whose pairwise products $x_i s_i$ are reduced to a lower average

value—not all the way to zero, this step is toward $x_i s_i = \sigma\mu$ where μ is the duality step and the centering parameter, $\sigma \in [0, 1]$, this leads to a modified step:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ -XSe + \sigma\mu e \end{bmatrix}. \tag{5.13}$$

As explained in Problem 4, P4.1 & P4.2, Mehrotra proposed a few modifications, one of them is to factorise the matrix in (5.13) twice, leading to an affine and a corrector step, the idea is to reduce the duality measure more than simply using an affine step (Nocedal & Wright, 1999). First the affine step is solved by the system:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{aff}} \\ \Delta\lambda^{\text{aff}} \\ \Delta s^{\text{aff}} \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ -XSe \end{bmatrix} \tag{5.14}$$

The above solution will cause the updated value of $x_i s_i$ to be $\Delta x_i^{2\pi} \Delta s_i^{2ff}$ rather than the ideal value 0. To correct the deviation the corrector step is found:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^{\text{cor}} \\ \Delta\lambda^{\text{cor}} \\ \Delta s^{\text{cor}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\Delta X^{\text{ aff}} \Delta S^{\text{ aff}} e \end{bmatrix} \tag{5.15}$$

Then the affine step-lengths are found:

$$\begin{aligned} \alpha_{\text{aff}}^{\text{pri}} &= \min\left(1, \min_{i:\Delta x_i^{\text{aff}}<0} -\frac{x_i}{\Delta x_i^{\text{aff}}}\right), \\ \alpha_{\text{aff}}^{\text{dual}} &= \min\left(1, \min_{i:\Delta s_i^{\text{aff}}<0} -\frac{s_i}{\Delta s_i^{\text{aff}}}\right), \end{aligned} \tag{5.16}$$

This is done to calculate the maximum allowable step lengths along the affine-scaling direction. This defines μ_{aff} to be the value of μ:

$$\mu_{\text{aff}} = \frac{\left(x + \alpha_{\text{aff}}^{\text{pri}} \Delta x^{\text{aff}}\right)^T \left(s + \alpha_{\text{atf}}^{\text{dual}} \Delta x^{\text{aff}}\right)}{n}. \tag{5.17}$$

and centering parameter:

$$\sigma = \left(\frac{\mu_{\text{ aff}}}{\mu}\right)^3. \tag{5.18}$$

Now the search direction can be found, by following the linear system:

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ -XSe - \Delta X^{\text{aff}} \Delta S^{\text{aff}} e + \sigma \mu e \end{bmatrix}. \tag{5.19}$$

Finally the step lengths are found from (14.36) in (Nocedal & Wright, 1999):

$$\alpha^{\text{pri}}_{k,\max} \overset{\text{def}}{=} \min_{i:\Delta x^k_i < 0} -\frac{x^k_i}{\Delta x^k_i}, \quad \alpha^{\text{dual}}_{k,\max} \overset{\text{def}}{=} \min_{i:\Delta s^k_i < 0} -\frac{s^k_i}{\Delta s^k_i}, \tag{5.20}$$

and the next iterate can be computed by:

$$\begin{bmatrix} x^{k+1} \\ \lambda^{k+1} \\ s^{k+1} \end{bmatrix} = \begin{bmatrix} x^k \\ \lambda^k \\ s^k \end{bmatrix} + \begin{bmatrix} \alpha^{\text{pri}}_k \Delta x \\ \alpha^{\text{dual}}_k \Delta \lambda \\ \alpha^{\text{dual}}_k \Delta s \end{bmatrix}. \tag{5.21}$$

From this the stopping criteria are given by:

$$\|r_c\| \leq \varepsilon \quad \|r_b\| \leq \varepsilon \quad |\mu| \leq \varepsilon \tag{5.22}$$

With ϵ being a user-specified tolerance. The algorithm can be seen in Algorithm 2. Once again one can do matrix factorisation of the linear systems, by augmenting the system. Once again this won't be elaborated, but rather referred to (Bagterp Jørgensen, 2019c, p. 20). Algorithm 14.3 from (Nocedal & Wright, 1999, p.411) is outlined below:
Please notice that the notation described is different to the slides in (Bagterp Jørgensen, 2019c), but follows the notation from (Nocedal & Wright, 1999). I.e in the slides the notation is g, λ, s, μ for respectively c, s, μ, λ in the book. The implementation of Algorithm 14.3 is based on the slides in (Bagterp Jørgensen, 2019c), therefore changes in the notation might occur. The implementation can be seen in Appendix Problem 5, Listing 14.

P14.15 Testing

Testing (5.1) with $n = 4$ and $m = 2$, where A, b and c are randomly chosen:

$$A = \begin{bmatrix} 2.3459 & 2.2103 & 0.6762 & 1.0007 \\ 0.0893 & 0.7440 & -0.4959 & -1.8874 \end{bmatrix}, \quad b = \begin{bmatrix} 1.5545 \\ 0.3461 \end{bmatrix}, \quad c = \begin{bmatrix} 2.1123 \\ 2.6391 \\ 0.8753 \\ -0.7947 \end{bmatrix}$$

Algorithm 2 Primal-Dual Interior-Point LP Solver

procedure LPIPPD
  Require (x^0, λ^0, s^0) where $(x^0, s^0) > 0$
  while not CONVERGED **do**
    Set$(x, \lambda, s) = \left(x^k, \lambda^k, s^k\right)$ and solve (5.14) for $\left(\Delta x^{\text{ alf}}, \Delta\lambda^{\text{ aff}}, \Delta s^{\text{ aff}}\right)$;
    Calculate largest $\alpha^{\text{pri}}_{\text{aff}}, \alpha^{\text{dual}}_{\text{aff}}$ so following is satisfied, (5.16):

$$x + \alpha^{\text{pri}}_{\text{aff}} \Delta x^{aff} \geq 0 \quad s + \alpha^{\text{dual}}_{\text{aff}} \Delta s^{aff} \geq 0$$

    Find affine duality gap: μ_{aff} (5.17);
    Set centering parameter to $\sigma = \left(\mu_{\text{aff}}/\mu\right)^3$;
    Solve (5.19) for $(\Delta x, \Delta\lambda, \Delta s)$;
    Calculate largest α^{pri}_k and α^{dual}_k from (5.20)
    Set

$$x^{k+1} = x^k + \alpha^{\text{pri}}_k \Delta x$$
$$\left(\lambda^{k+1}, s^{k+1}\right) = \left(\lambda^k, s^k\right) + \alpha^{\text{dual}}_k (\Delta\lambda, \Delta s)$$

  end while
end procedure

The results found are:

$$x^* = \begin{bmatrix} 0.2530 \\ 0.4348 \\ 0 \\ 0 \end{bmatrix}, \quad \lambda^* = \begin{bmatrix} 0.8630 \\ 0.9834 \end{bmatrix}, \quad s^* = \begin{bmatrix} 0.0000 \\ 0.0000 \\ 0.7794 \\ 0.1977 \end{bmatrix}$$

To test if the implementation is correct, one can use two methods, check the maximum difference between the true solution and the estimated solution or the estimated solution has to satisfy the KKT conditions in (5.6). For the second approach the result is

$$A \cdot x^* - b = \begin{bmatrix} 0.1413 \cdot 10^{-10} \\ -0.0573 \cdot 10^{-10} \end{bmatrix}$$

And

$$A^T\lambda^* + s^* - c = \begin{bmatrix} -0.2110 \cdot 10^{-11} \\ -0.3110 \cdot 10^{-11} \\ 0.0237 \cdot 10^{-11} \\ 0.3405 \cdot 10^{-11} \end{bmatrix}$$

Finally

$$X^*S^*e = 2.4360 \cdot 10^{-10}$$

All close to 0, but because of the tolerance set to $\epsilon = 10^{-9}$, the convergence criteria will cause a minor deviation from 0. If a more precise result is desired, the tolerance

can be lowered. Therefore (5.6) is satisfied and x^* is a solution.

Problem 6

P6.1

> Problem 18.3 in Nocedal and Wright (p .562, note that they have interchanged the starting point x_0 and the solution x^* . 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 $f(x), \nabla f(x), \nabla^2 f(x), c_i(x), \nabla c_i(x), \nabla^2 c_i(x)$

The problem in focus is given by Nocedal and Wright, 1999:

$$\begin{aligned} \min_{x\in\mathbb{R}^5} \quad & e^{x_1x_2x_3x_4x_5} - \frac{1}{2}\left(x_1^3 + x_2^3 + 1\right)^2 \\ \text{s.t.} \quad & x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ & x_2x_3 - 5x_4x_5 = 0 \\ & x_1^3 + x_2^3 + 1 = 0 \end{aligned} \tag{6.1}$$

It is an equality constrained problem, where the Jacobian and gradients have been implemented in OBJ.M, OBJ1.M, OBJ2.M and NLPCON.M, Appendix Problem 7. Problem (6.1) will be solved by using sequential quadratic programming (SQP). It is based on iteratively determining some solution, by Newton's method using a computed step length and by linearising the constraints, where the KKT conditions are found from the non-linear program. The method uses a QP-solver for each step, which also was implemented in Problem 1. To start with a simple SQP-algorithm will be implemented. For the simple SQP-algorithm to work, it is assumed the true Hessian is known and non-singular.

Once again, we will start by looking at the Lagrangian function:

$$L(x, y) = f(x) - y'h(x)$$

Which for this particular problem will be:

$$\mathcal{L}(x, \lambda) = e^{x_1x_2x_3x_4\tau_5} - \begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 \end{bmatrix} \begin{bmatrix} x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 \\ x_2x_3 - 5x_4x_5 \\ x_1^3 + x_2^3 + 1 \end{bmatrix}$$

The explicit calculations of the gradients and Jacobians will not be shown, but rather we refer to the implementations. The KKT conditions are then given by:

$$\begin{aligned}\nabla_x L(x,y) &= \nabla f(x) - \nabla h(x) y = 0 \\ \nabla_y L(x,y) &= -h(x) = 0\end{aligned}$$

Once again we can write this as a system of non-linear equations:

$$F(x,y) = \begin{bmatrix} \nabla_x L(x,y) \\ \nabla_y L(x,y) \end{bmatrix} = \begin{bmatrix} \nabla f(x) - \nabla h(x) y \\ -h(x) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0$$

To find the step for the solution of $F(x) = 0$, we can use by Newton's method:

$$J\left(x^k\right) \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = -F\left(x^k\right), \quad \text{where } J\left(x^k\right) = \left[\nabla F\left(x^k\right)\right]' \tag{6.2}$$

The gradient of $F(x)$ then gives

$$\nabla F\left(x^k, \lambda^k\right) = \begin{bmatrix} \nabla_x F\left(x^k, \lambda^k\right) \\ \nabla_\lambda F\left(x^k, \lambda^k\right) \end{bmatrix} = \begin{bmatrix} \nabla^2_{xx} \mathcal{L}\left(x^k, \lambda^k\right) & -\nabla h \\ -\nabla h' & 0 \end{bmatrix} \tag{6.3}$$

From (6.2), (6.3) and a few adjustments, Newton's method gives:

$$\begin{bmatrix} \nabla^2_{xx} L\left(x^k, \lambda^k\right) & -\nabla h\left(x^k\right) \\ -\nabla h\left(x^k\right)' & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla_x L\left(x^k, \lambda^k\right) \\ -h\left(x^k\right) \end{bmatrix}. \tag{6.4}$$

Now (6.4) has a unique solution that satisfies the standard form (Bagterp Jørgensen, 2019g):

$$\begin{aligned} \min_{\Delta x \in \mathbb{R}^n} \quad & \tfrac{1}{2} \Delta x' H \Delta x + g' \Delta x \\ \text{s.t.} \quad & A' \Delta x = b \end{aligned} \tag{6.5}$$

Where $H = \nabla^2_{xx} L\left(x^k, y^k\right)$, $g = \nabla f\left(x^k\right)$, $A = \nabla h\left(x^k\right)$ and $b = -h\left(x^k\right)$. Where like regular QPs assumption 18.1 in Nocedal and Wright, 1999 should be satisfied, i.e. $\nabla^2_{xx} \mathcal{L}(x, \lambda)$ is positive definite on the tangent space of the constraints and therefore the problem is convex, even when x^k is close to the solution and that the constraint Jacobian $A(x)$ has full row rank, linear independence between the constraints and the Jacobian. Furthermore Theorem 12.6 (Nocedal & Wright, 1999) is also satisfied for the KKT conditions being sufficient for x^* being a strict local solution. Quadratic convergence applies to the Newton iterations, if the assumptions are fulfilled (Nocedal & Wright, 1999, p. 531)

Now one can solve for Δx and λ_{old}^{k+1} to get the step length updates of x^{k+1} and λ_{new}^{k+1}:

$$\begin{bmatrix} x^{k+1} \\ \lambda_{new}^{k+1} \end{bmatrix} = \begin{bmatrix} x^{k} \\ \lambda_{old}^{k+1} \end{bmatrix} + \begin{bmatrix} \Delta x \\ 0 \end{bmatrix} \tag{6.6}$$

This is done iteratively by the function EqualityQPSolver.m implemented in Problem 1 yielding $(\Delta x_1 \Delta\lambda)$. Finally recomputing A, g, b, H, until convergence (the stopping criteria) given by:

$$\begin{aligned} \left\| \nabla f\left(x^k\right) - \nabla h\left(x^k\right)^T \lambda \right\|_\infty &< \epsilon \\ \left\| h\left(x^k\right) \right\|_\infty &< \epsilon \end{aligned} \tag{6.7}$$

Where ϵ is again a user specified tolerance. The whole process can be seen in the flow chart, Figure 10. Notice that λ_0 could be initialised by using the EqualityQPSolver and the initial positive definite Hessian B, however in this case it was just set to some arbitrary value.

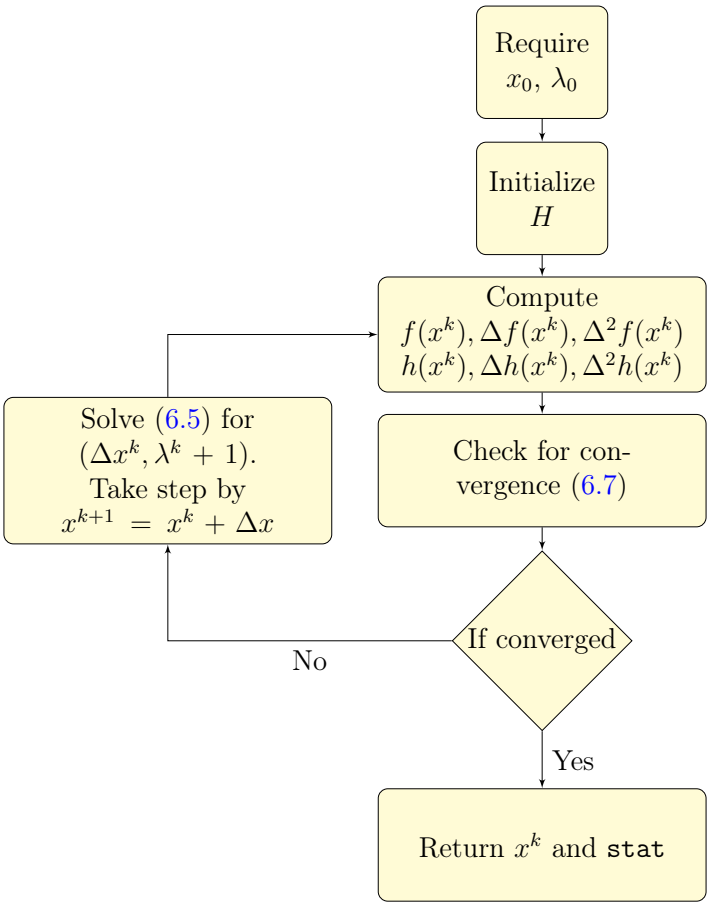


Figure 10: Flow chart of the SQP algorithm.

In Listing 15 in Appendix, Problem 6, one can see the implementation of the SQP algorithm in MATLAB. The implementation takes an initial guess of x_0 and λ_0 as input, where:

$$x_0 = (-1.8, 1.7, 1.9, -0.8, -0.8)^T$$

Solving (6.1) the sequence of iterations below is achieved.

Iteration	x_1	x_2	x_3	x_4	x_5	λ_1	λ_2	λ_3	F
1	-1.8	1.7	1.9	-0.8	-0.8	0.0024952	0.019985	-0.082223	0.02093
2	-1.6829	1.5594	1.8943	-0.76914	-0.76914	-0.034542	0.033611	-0.0043357	0.05249
3	-1.7231	1.6027	1.8179	-0.76381	-0.76381	-0.039848	0.037528	-0.0064712	0.053455
4	-1.7171	1.5957	1.8273	-0.76366	-0.76366	-0.040155	0.037949	-0.0052262	0.05394
5	-1.7171	1.5957	1.8272	-0.76364	-0.76364	-0.040163	0.037958	-0.0052226	0.05395

Table 1: Iteration sequence for the simple SQP

Within only 4 iterations the algorithm converges. The solution is found to be close to the true solution given by:

$$x^* - (-1.71, 1.59, 1.82, -0.763, -0.763)^T$$

The algorithm is not very practical. It might have a fast convergence. However if H is not positive definite on the tangent space of the constraints (Nocedal & Wright, 1999, Theorem 18.1), there is no guarantee of convergence.

P6.2

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

At times the hessian, H, can be difficult and expensive to compute, this is where Quasi-Newton approximation methods arise. Full Quasi-Newton approximation will use a positive semi-definite matrix B_k and update it, resulting from the step k to $k+1$, where the following vectors are used:

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1}) \tag{6.8}$$

Each new approximation, B_{k+1}, has to remain positive definite around the minimum, this allows for a robust convergence. This also requires s_k and y_k satisfy the curvature condition

$$s_k' y_k > 0 \tag{6.9}$$

This is why the following is introduced (Nocedal & Wright, 1999, p. 537):

$$r_k = \theta_k y_k + (1 - \theta_k) B_k s_k \tag{6.10}$$

Where θ_k is a scalar and defined as:

$$\theta_k = \begin{cases} 1 & \text{if } s_k^T y_k \geq 0.2 s_k^T B_k s_k \\ \left(0.8 s_k^T B_k s_k\right) / \left(s_k^T B_k s_k - s_k^T y_k\right) & \text{if } s_k^T y_k < 0.2 s_k^T B_k s_k \end{cases} \tag{6.11}$$

B_k can then be updated using the following update:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{r_k r_k^T}{s_k^T r_k}. \tag{6.12}$$

One can show that for $\theta_k \neq 1$, then B_{k+1} is positive definite from:

$$s_k^T r_k = 0.2 s_k^T B_k s_k > 0 \tag{6.13}$$

Therefore the choice of θ_k ensures that the new approximation stays close enough to B_k to ensure positive definiteness, this is a value of $\theta_k \in (0, 1)$.

The SQP with damped BFGS can be seen in Listing 16, in Appendix 6, NEWTONSQP_BFGS.M. The algorithm doesn't need the true Hessian of the Lagrange function, but only needs a guess of a Hessian that is positive definite. The identity matrix is used, assuring a positive definite matrix. Using once again problem (6.1) the iteration sequence in Table 2 is seen. It is clear that it converges slower (9 iterations against 4). This is understandable given the true Hessian adds a lot more information, compared to the approximated initial Hessian, but the real benefit of SQP with damped BFGS is that no true Lagrangian Hessian is needed, which makes one avoid a potential computationally expensive Hessian.

Iteration	x_1	x_2	x_3	x_4	x_5	λ_1	λ_2	λ_3	F
1	-1.8	1.7	1.9	-0.8	-0.8	0.0024952	0.019985	-0.082223	0.02093
2	-1.7269	1.6087	1.8132	-0.76362	-0.76362	-0.044088	0.019613	-0.084656	0.052928
3	-1.7243	1.6041	1.8138	-0.76282	-0.76282	-0.039379	0.037364	-0.019088	0.053973
4	-1.7207	1.5998	1.8207	-0.76325	-0.76325	-0.040605	0.038049	-0.0047908	0.053951
5	-1.7171	1.5957	1.8274	-0.76366	-0.76366	-0.039015	0.037338	-0.018446	0.053946
6	-1.7171	1.5957	1.8273	-0.76365	-0.76365	-0.04014	0.037929	-0.0055048	0.05395
7	-1.7172	1.5958	1.8271	-0.76363	-0.76363	-0.040108	0.037958	-0.0056514	0.05395
8	-1.7171	1.5957	1.8272	-0.76364	-0.76364	-0.040138	0.037981	-0.005331	0.05395
9	-1.7171	1.5957	1.8272	-0.76364	-0.76364	-0.040162	0.037957	-0.0052308	0.05395
10	-1.7171	1.5957	1.8272	-0.76364	-0.76364	-0.040163	0.037958	-0.0052227	0.05395

Table 2: Iteration sequence for the SQP with damped BFGS algorithm

In large scale cases the Quasi-Newton approximation might require a lot of memory,

due to being dense with $n \times n$ dimension. Another setback is that damped BFGS updating can fail, such as if the Lagrangian Hessian is not positive definite. This is where we need to use line search or trust-region methods (Nocedal & Wright, 1999, p.538).

P6.3

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

An improvement of the SQP methods ability to take optimal steps is to use the line search method complementary to the damped BFGS method. In this problem it will only be shown for case of equality constraints. The line search method as seen in Algorithm 3 will use a "regularisation" parameter α to take appropriate step lengths, $\alpha\Delta x^k$. This is done by using a merit function, which in this case is Powell's l_1-merit function (Bagterp Jørgensen, 2019g):

$$P(x, \lambda, \mu) - f(x) + \lambda' |h(x)| \tag{6.14}$$

Where $\lambda \geq |y|$. The penalty, λ, is then updated by:

$$\lambda - \max\left\{|y|, \frac{1}{2}(\lambda + |y|)\right\} \tag{6.15}$$

We then adjust the new iterate by $\alpha\Delta x^k$:

$$x^{k+1} - x^k + \alpha\Delta x^k \tag{6.16}$$

And the merit function for α as

$$\begin{aligned} \phi(\alpha) = P(x, \lambda) &= P\left(x^k + \alpha\Delta x^k, \lambda\right) \\ &= f\left(x^k + \alpha\Delta x^k\right) + \lambda' \left| h\left(x^k + \alpha\Delta x^k\right)\right| \end{aligned} \tag{6.17}$$

To accept $\alpha_k\Delta x$ the Armijo condition has to be satisfied (Nocedal & Wright, 1999, p. 540).

$$\phi(\alpha) \leq \phi(0) + c_1\alpha\frac{d\phi}{d\alpha}(0) \tag{6.18}$$

Where $c_1 \in (0, 1)$, $c = 0.1$ in this case. By making a quadratic approximation:

$$\phi(\alpha) \approx q(\alpha) = a\alpha^2 + b\alpha + c \tag{6.19}$$

With a bit of algebra we get (Bagterp Jørgensen, 2019g, p. 24):

$$a - \frac{\phi(\alpha_1) - \left(\phi(0) + \phi'(0)\alpha_1\right)}{\alpha_1^2} \tag{6.20}$$

Because of quadratic nature

$$\alpha_{\min} = -\frac{b}{2a} \tag{6.21}$$

α_{min} is only accepted if $\alpha_{\min} \in [0.1\alpha, 0.9\alpha]$ due to α_{min} being an approximation of the optimal α, this leads to: $\alpha - \min\left\{0.9\alpha, \max\left\{\alpha_{\min}, 0.1\alpha\right\}\right\}$ We continue this approach until the Armijo condition is satisfied.

Algorithm 3 Line Search (Bagterp Jørgensen, 2019g, p. 25)

procedure LINE SEARCH
  Require $f\left(x^k\right), \nabla f\left(x^k\right), h\left(x^k\right), g\left(x^k\right), \lambda, \Delta x^k$
  $\alpha = 1, i = 1$, STOP = false
  $c = \phi(0) = f\left(x^k\right) + \lambda \left|h\left(x^k\right)\right|$
  $b = \phi'(0) = \nabla f\left(x^k\right)' \Delta \dot{x^k} - \lambda' \left|h\left(x^k\right)\right|$
  while not STOP **do**
    $x = x^k + \alpha \Delta x^k$
    Evaluate $f(x), h(x), g(x)$
    Compute $\phi(\alpha) = f(x) + \lambda' |h(x)|$
    if $\phi(\alpha) \leq \phi(0) + 0.1\phi'(0)\alpha$ **then**
      STOP = true
    else
      Compute $a = \frac{\phi(\alpha) - (c + b\alpha)}{\alpha^2}$ and $\alpha_{\min} = \frac{-b}{2a}$
      $\alpha = \min\left\{0.9\alpha, \max\left\{\alpha_{\min}, 0.1\alpha\right\}\right\}$
    end if
  end while
end procedure

Once again the SQP using damped BFGS and line search is implemented and can be seen in Appendix P6, Listing 17. Once again problem (6.1) is solved with the same starting point as previously used. This gives the following iteration sequence:

Iteration	x_1	x_2	x_3	x_4	x_5	λ_1	λ_2	λ_3	F
1	-1.8	1.7	1.9	-0.8	-0.8	0.0024952	0.019985	-0.082223	0.02093
2	-1.7269	1.6087	1.8132	-0.76362	-0.76362	-0.044088	0.019613	-0.084656	0.052928
3	-1.7243	1.6041	1.8138	-0.76282	-0.76282	-0.039379	0.037364	-0.019088	0.053973
4	-1.7207	1.5998	1.8207	-0.76325	-0.76325	-0.040605	0.038049	-0.0047908	0.053951
5	-1.7171	1.5957	1.8274	-0.76366	-0.76366	-0.039015	0.037338	-0.018446	0.053946
6	-1.7171	1.5957	1.8273	-0.76365	-0.76365	-0.04014	0.037929	-0.0055048	0.05395
7	-1.7171	1.5957	1.8272	-0.76364	-0.76364	-0.040133	0.037935	-0.0055349	0.05395
8	-1.7171	1.5957	1.8272	-0.76364	-0.76364	-0.040163	0.037958	-0.0052226	0.05395

Table 3: Iteration sequence of SQP with damped BFGS and line search.

Again, we are close to the true solution x^*. It converges in 7 iterations and is therefore converging faster than the damped BFGS. Another thing to notice is that we are very close to the true solution after a few steps, this might be because of the optimal step length chosen.

P6.4

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

The optimal point is already known.

$$x^* - (-1.71, 1.59, 1.82, -0.763, -0.763)^T$$

All the algorithms was run, with a tolerance set to 10^-6. Each iteration can then be compared to the true solution x^*. The convergence can then be investigated by calculating:

$$\epsilon_k = \|x_k - x^*\|_2 \,. \tag{6.22}$$

Looking at the convergence in Figure 11 it is seen that all of the methods seem to have superlinear convergence rate. The local SQP with the true Hessian seem to have a convergence rate slightly better, also given Figure 12, this makes sense due to the Lagrangian Hessian adds additional information about the optimization problem and not having to approximate. From Figure 12 it can be seen that the local SQP and SQP with line search and damped BFGS seem to converge quickly to the solution, with the damped BFGS using line search almost getting close to the solution as quickly as the local SQP, however it takes a few more iterations to get to the set tolerance for convergence.

Problem 7

Consider the problem

$$\min_x \quad f(x) = \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + x_2^2 - 7\right)^2 \tag{7.1a}$$

$$\text{s.t.} \quad c_1(x) = (x_1 + 2)^2 - x_2 \geq 0 \tag{7.1b}$$

$$c_2(x) = -4x_1 + 10x_2 \geq 0 \tag{7.1c}$$

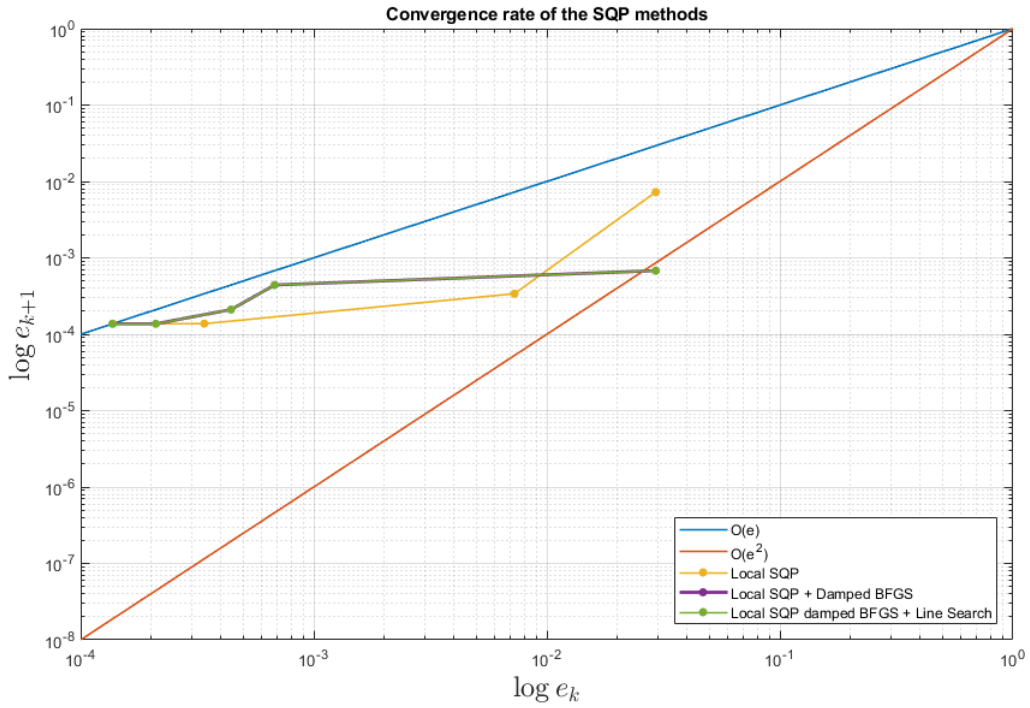


Figure 11: Convergence rate of the different SQP algorithms. $\mathcal{O}(e)$ and $\mathcal{O}(e^2)$ is used for comparison reasons.

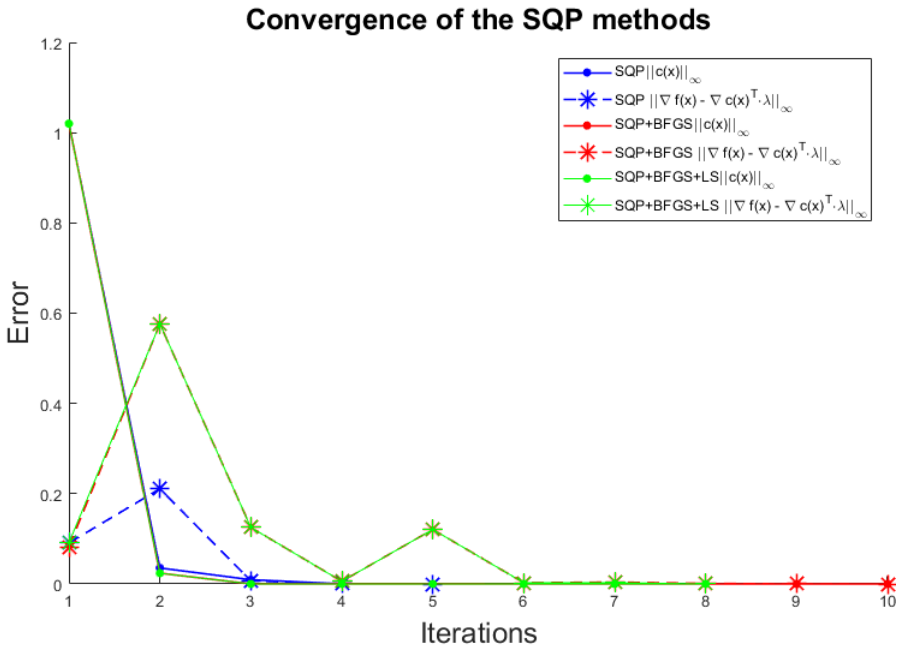


Figure 12: Error against iteration for the two stopping criterias.

P7.1

> 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.

In contrast to the previous problem, an inequality constrained problem is given. This requires the SQP framework to be extended, luckily this is easily done (Nocedal & Wright, 1999, p. 533). Given a general nonlinear programming problem

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & c_i(x) = 0, \quad i \in \mathcal{E} \\ & c_i(x) \geq 0, \quad i \in \mathcal{I} \end{aligned}$$

This can then be linearized to enable a QP solver to solve it:

$$\begin{aligned} \min_p \quad & f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla^2_{xx} \mathcal{L}_k p \\ \text{s.t.} \quad & \nabla c_i \left(x_k\right)^T p + c_i \left(x_k\right) = 0, \quad i \in \mathcal{E} \\ & \nabla c_i \left(x_k\right)^T p + c_i \left(x_k\right) \geq 0, \quad i \in \mathcal{I} \end{aligned} \tag{7.2}$$

One might notice that the above can be written in standard form, f_k being omitted, due to it just being a constant:

$$\begin{aligned} \min_{p \in \mathbb{R}^2} \quad & \tfrac{1}{2} p^T H p + g p \\ \text{s.t.} \quad & A^T p - b \geq \end{aligned} \tag{7.3}$$

Where for (7.1)

$$A(x_k)^T = \begin{bmatrix} \nabla c_1(x_k) & \nabla c_2(x_k) \end{bmatrix}, \quad b(x_k) = - \begin{bmatrix} c_1(x_k) & c_2(x_k) \end{bmatrix} \tag{7.4}$$

and

$$g(x_k) = \nabla f(x_k)^T, \quad H = \nabla^2_{xx} \mathcal{L}_k \tag{7.5}$$

To solve (7.1), it is therefore necessary to find the derivatives, so it can be rewritten into a quadratic problem. The gradient ,$\nabla f(x)$, of (7.1) is given by:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 4x_1 \left(x_1^2 + x_2 - 11\right) + 2\left(x_1 + x_2^2 - 7\right) \\ 2\left(x_1^2 + x_2 - 11\right) + 4x_2 \left(x_1 + x_2^2 - 7\right) \end{bmatrix}. \tag{7.6}$$

It's not necessary to find the Hessian of $f(x)$, since it will be approximated by the Quasi-Newton approximation, B_k, by the damped BFGS algorithm as explained in

Section P6.2. Therefore $B_k \approx \nabla^2_{xx}\mathcal{L}_k$. The gradient of the constraint functions are given by:

$$\nabla c_1(x) = \begin{bmatrix} \frac{\partial c_1}{\partial x_1} \\ \frac{\partial c_1}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2\,(x_1 + 2) \\ -1 \end{bmatrix}, \quad \nabla c_2(x) = \begin{bmatrix} \frac{\partial c_2}{\partial x_1} \\ \frac{\partial_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} -4 \\ 10 \end{bmatrix} \tag{7.7}$$

Now the problem in (7.3) can be solved by the SQP framework.

Algorithm 4 SQP ICQ with damped BFGS (Nocedal & Wright, 1999, p.532)

procedure LOCAL SQP
  Choose an initial pair (x_0, λ_0); set $k \leftarrow 0$
  while not converged **do**
    Evaluate $f_k, \nabla f_k, \nabla^2_{xx}\mathcal{L}_k, c_k$, and A_k
    Solve (7.3) using a QP solver (such as `quadprog` to find p_k and l_k.
    Update $x_{k+1} = x_k + p_k$ and $\lambda_{k+1} = l_k$
    Approximate B_k using damped BFGS
    Check convergence, $\|p\|_\infty < \epsilon$
  end while
end procedure

The algorithm has been implemented in `SQP_BFGS_ineq.m` and can be seen in Appendix Problem 7, Listing 18. The iteration sequence for problem (7.1) can be seen in Figure 13. The starting points used are $x_0 = (0, 0)$, $x_0 = (3, 2)$, $x_0 = (-4.5, 3)$, $x_0 = (-3, -4)$. A tolerance of $\epsilon = 10^{-6}$ was used. The iteration sequence can also be seen in Table 4. It is clear that for starting points far away from the minima and close to the constraints or in the constraints makes the algorithm behave strange, where when using $x_0 = (-4.5, 3)$ the algorithm does not converge and ends at maximum iterations of 200. However a clear convergence can be observed at the minima of $x^* = (3, 2)$ when the starting guess is not too far away or within the constraint. It can also be observed that sometimes the steps taken are very large and far from the solution. To improve the steps a line search algorithm will be implemented in P7.2.

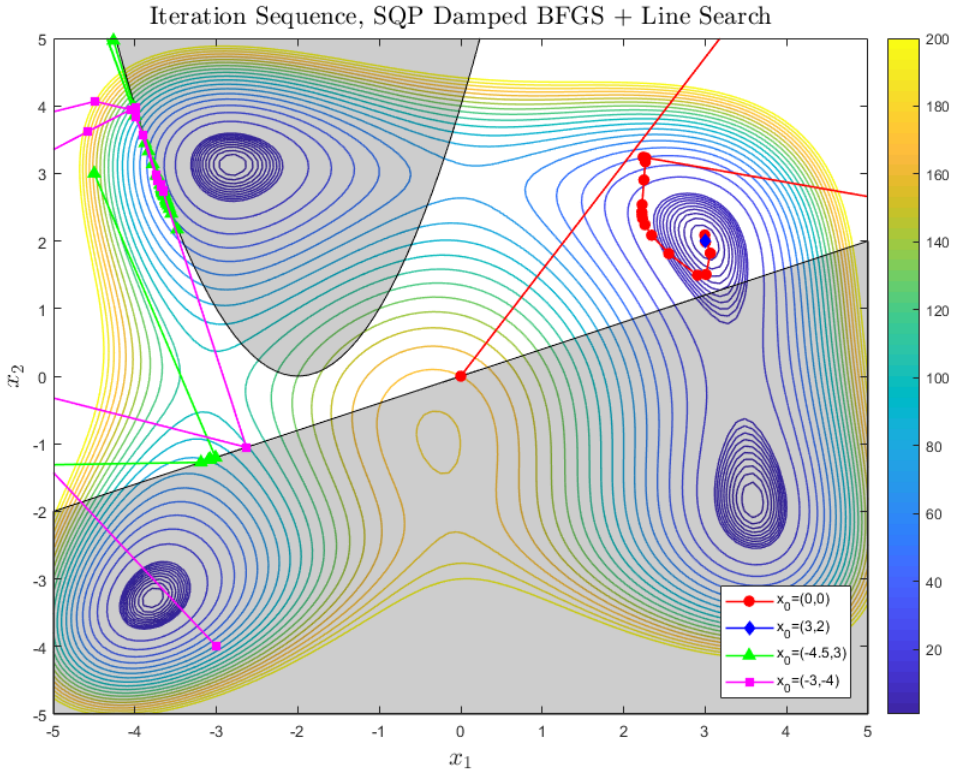


Figure 13: Iteration sequence plotted on the contour of the problem in (7.1), SQP algorithm uses damped BFGS

Iteration	x_1	x_2	x_1	x_2	x_1	x_2	x_1	x_2
0	0	0	3	2	-4.5	3	-3	-4
1	14	22	3	2	-3.0093	-1.2037	-87	104
2	6.0759	2.4304			-3.0746	-1.2299	-44.396	-17.758
3	2.2386	3.2405			-3.0943	-1.2377	-23.12	-6.56
4	2.2701	3.2319			-3.1896	-1.2758	-18.835	-7.5341
5	2.2659	3.1725			-2952.8	-57.521	-16.591	-6.6366
6	2.2481	2.9042			-1477.3	-590.92	-9.3175	0.63584
7	2.2264	2.5383			-824.59	2.5061e+05	-8.5244	1.1209
8	2.2226	2.426			-1476.2	-590.46	-5.4961	3.0376
9	2.2243	2.3826			-1475.8	-590.33	-4.5854	3.6152
10	2.2319	2.3334			-1474.2	-589.67	-4.0553	3.9435
11	2.2606	2.2423			-737.99	-285.09	-3.9938	3.9714
12	2.3426	2.0858			-685.34	-263.31	-3.9886	3.9544
13	2.5532	1.8142			-474.6	-176.13	-3.987	3.8261
14	2.9057	1.4945			-370.25	-132.97	-3.7203	2.8883
15	3.0114	1.5045			-276.35	-94.12	-3.6839	2.8341
16	3.0628	1.8206			-209.9	-66.634	-3.6753	2.8066
17	2.995	2.0893			-158.43	-45.339	-2.6316	-1.0526
18	3.0049	1.9866			-119.9	-29.403	-2.6322	-1.0529
19	3.0006	1.9989			-90.662	-17.307	-1012.4	310.06
20	3	2			-68.565	-8.1664	-507.11	-202.85
21	3	2			-51.818	-1.2385	-381.51	1485.7
22	3	2			-39.13	4.0102	-583.31	-201.57
23	3	2			-29.538	7.9781	⋮	⋮
63					⋮	⋮	-3.6546	2.7377
200					-3.6546	2.7378		

Table 4: Iteration sequence of the SQP with damped BFGS.

P7.2

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.

A line search algorithm will now be implemented. The framework of the line search algorithm itself will not be shown here, but we rather refer to Algorithm 18.3, Line Search SQP Algorithm in (Nocedal & Wright, 1999, p. 545), which is the implemented algorithm.
It is based on merit functions to control the size of the step. The l_1 merit function for (7.2) is defined as (Nocedal & Wright, 1999, p.540):

$$\phi_1(x;\mu) = f(x) + \mu\|c(x)\|_1 \tag{7.8}$$

This allows to decide whether a step, $\alpha_k p_k$, will be accepted or not, by the following sufficient decrease condition, just as for the Armijo condition for unconstrained problems:

$$\phi_1\left(x_k + \alpha_k p_k; \mu_k\right) \leq \phi_1\left(x_k, \mu_k\right) + \eta\alpha_k D\left(\phi_1\left(x_k;\mu\right); p_k\right) \tag{7.9}$$

Where $D\left(\phi_1\left(x_k;\mu\right); p_k\right)$ denotes a directional derivative of ϕ_1 in the direction p_k. By Theorem 18.2 in (Nocedal & Wright, 1999, p. 541) the directional derivative is defined as:

$$D\left(\phi_1\left(x_k;\mu\right); p_k\right) = \nabla f_k^T p_k - \mu\left\|c_k\right\|_1 \tag{7.10}$$

The penalty parameter μ has to be large enough to make the descent condition hold and is defined by:

$$\mu \geq \frac{\nabla f_k^T p_k + (\sigma/2) p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k}{(1-\rho)\left\|c_k\right\|_1} \tag{7.11}$$

Based on (7.9) the step size $\alpha_k p_k$ can now be adjusted by resetting the parameter α.
The algorithm has been implemented in `SQP_BFGS_LS_ineq.m` and can be seen in Appendix Problem 7, Listing 19. The iteration sequence for problem (7.1) can be seen in Figure 14. A clear improvement to the steps can be seen. With a lot fewer iterations and a faster convergence.

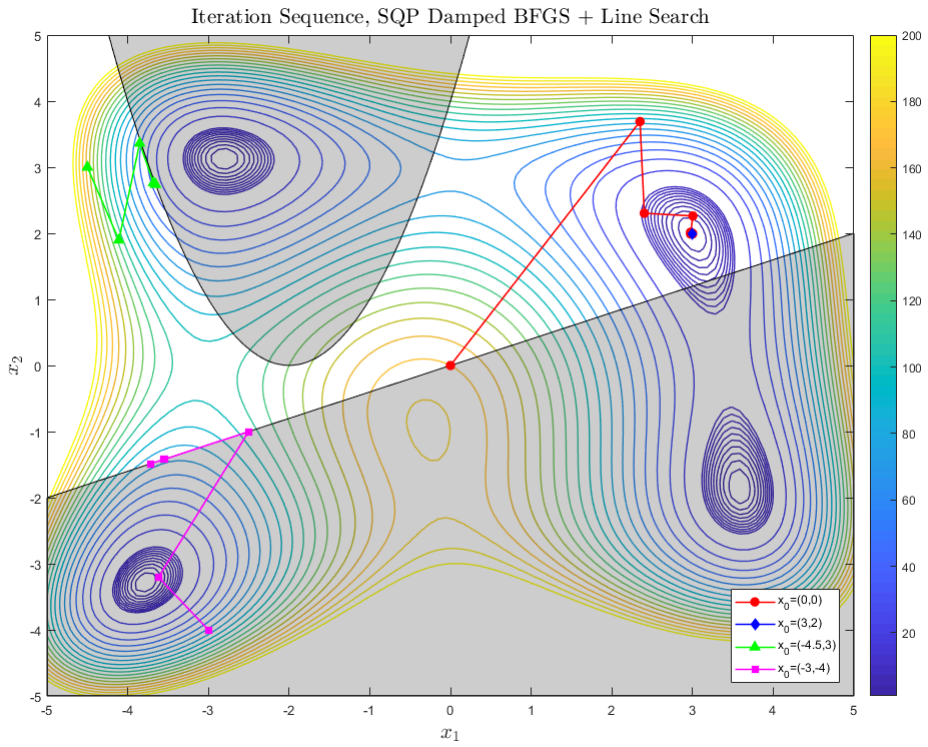


Figure 14: Iteration sequence plotted on the contour of the problem in (7.1), SQP algorithm uses damped BFGS and Backtracking Line Search

The same starting points are used as in the previous problem. All the points seem to converge to their nearest minima, however some converge to a local minima given by the inequality constraints.

Iteration	x_1	x_2	x_1	x_2	x_1	x_2	x_1	x_2
0	0	0	3	2	-4.5	3	-3	-4
1	8.96	14.08	3	2	-4.1092	1.898	-3.6198	-3.2031
2	5.4942	5.9762			-3.8517	3.3624	-2.5011	-1.0004
3	3.3476	5.9221			-3.6809	2.7639	-3.7125	-1.485
4	2.6762	2.7204			-3.6556	2.7402	-3.5461	-1.4184
5	2.8152	2.5859			-3.6551	2.7394	-3.549	-1.4196
6	2.9877	2.2854			-3.6546	2.7377	-3.549	-1.4196
7	2.995	2.133			-3.6546	2.7377	-3.5489	-1.4195
8	2.9946	2.0461			-3.6546	2.7377	-3.5485	-1.4194
9	2.9979	2.0129					-3.5485	-1.4194
10	2.9995	2.0029						
11	2.9999	2.0006						
12	3	2.0001						
13	3	2						
14	3	2						
15	3	2						
16	3	2						

Table 5: Iteration sequence of the SQP with damped BFGS and Backtracking Line Search.

As seen only with a maximum of 16 iterations each of the starting points have converged. Once again a tolerance $\epsilon = 10^{-6}$ was used. Below a few statistics can be

seen:

x_0	**Function Value**	**Iterations**	**Function Calls**
$(0, 0)$	$7.9896 \cdot 10^{-18}$	16	122
$(3, 2)$	0	1	330
$(-4.5, 3)$	35.9298	8	200
$(-3, -4)$	72.8555	9	100

Table 6: Statistics of the SQP with line search algorithm

Notice all the function calls around $(3, 2)$. It seems the sufficient decrease condition is getting stuck, making a lot of function calls to find an optimal step size.

P7.3

> 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.

Trust-region SQP methods do not require $\nabla^2_{xx}\mathcal{L}_k$ to be positive definite in (7.2). Furthermore they control the quality of steps even when Hessian and Jacobian introduce singularities and can enforce global convergence (Nocedal & Wright, 1999, p. 546). As (7.1) only have inequalities, this will be the focus. Trust regions introduce a new constraint to (7.2)

$$\|p\| \leq \Delta_k \tag{7.12}$$

It forces the step p to lie outside the trusted region which can be indicated by some circle with radius δ_k (Nocedal & Wright, 1999, p. 546).
In this case the *Penalty Update and Step Computation* algorithm 18.5 from (Nocedal & Wright, 1999) was implemented. It starts with requiring a solution of

$$\begin{aligned} \min_{p,v,w,t} \quad & f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla^2_{xx}\mathcal{L}_k p + \mu \textstyle\sum_{i\in\mathcal{I}} t_i \\ & \nabla c_i\left(x_k\right)^T p + c_i\left(x_k\right) \geq -t_i, \quad i \in \mathcal{I} \\ & t \geq 0 \\ & \|p\|_\infty \leq \Delta_k \end{aligned} \tag{7.13}$$

Here t_i is a slack variable, to solve this we need to the program into standard form to solve it using `quadprog`. Now since $t_i \in \mathbb{R}^2$ is introduced to each of the equations, it will increase the dimensions. Therefore

$$H = \begin{bmatrix} B & 0 \\ 0 & 0 \end{bmatrix}, \quad A = -\begin{bmatrix} \nabla c^T & I \end{bmatrix} \tag{7.14}$$

g is simply being extended by μ, $b = c$ and $\hat{p} = \begin{bmatrix} p \\ t \end{bmatrix}$

$$g = \begin{bmatrix} \nabla f_k \\ \mu \end{bmatrix} \tag{7.15}$$

Finally due to the trust region $-\Delta_k \leq p \leq \Delta_k$. It's now possible to set up the system in standard form, just like in (7.3) and can be easily solved using `quadprog`. The next step in Algorithm 18 (Nocedal & Wright, 1999, p. 554) is to use a piecewise linear model of constraint violation:

$$m_k(p) = \sum_{i \in \mathcal{I}} \left[c_i\left(x_k\right) + \nabla c_i\left(x_k\right)^T p \right]^- \tag{7.16}$$

The goal is to achieve linearized feasibility of the constraints, meaning we want the constraints to be satisfied with the respective slack variables. To do this we will adjust the penalty parameter μ by looking at a number of conditions. Increasing μ by a fixed factor until the conditions of $m_k(p)$ are satisfied. When satisfied with μ we update p_k and μ_k. The last step is to look at acceptability of the step p_k this is done by the nonsmooth ℓ_2 merit functions

$$\phi_2(x; \mu) = f(x) + \mu \|c(x)\|_2 \tag{7.17}$$

And equation (18.48) in (Nocedal & Wright, 1999, p.548). The whole implementation can be seen in SQP_TRUST_REG.M and can be seen in Appendix Problem 7, Listing 20. The same problem is used with $\epsilon = 10^{-6}$. Looking at Figure 15 it can be seen that the method effectively finds the local solution.

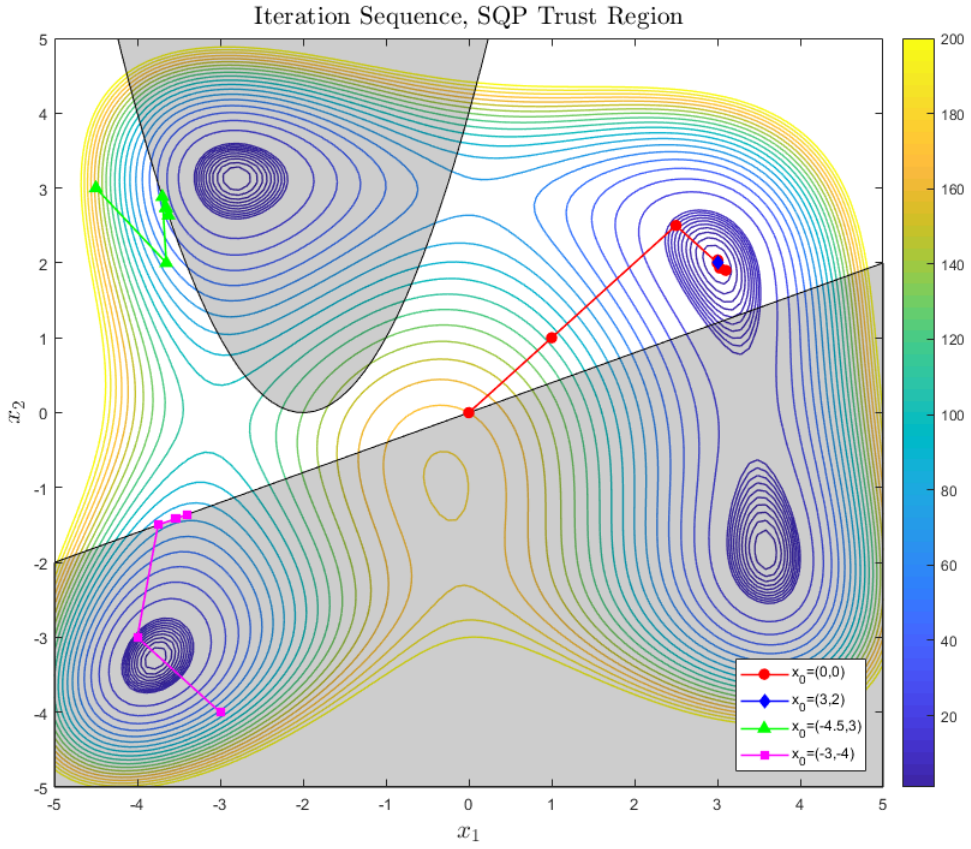


Figure 15: Iteration sequence plotted on the contour of the problem in (7.1), SQP algorithm uses damped BFGS and Backtracking Line Search

From Table 7 we see that SQP with Trust Region converges with less iterations toward the minimum than the SQP with damped BFGS and line search.

Iteration	x_1	x_2	x_1	x_2	x_1	x_2	x_1	x_2
0	0	0	3	2	-4.5	3	-3	-4
1	1	1	3	2	-3.65	2	-4	-3
2	2.5	2.5			-3.6984	2.8823	-3.75	-1.5
3	2.5	2.5			-3.6243	2.633	-3.75	-1.5
4	3.1	1.9			-3.6523	2.7295	-3.399	-1.3596
5	3.1	1.9			-3.6548	2.7382	-3.5345	-1.4138
6	3.029	1.931			-3.6546	2.7377	-3.5496	-1.4198
7	3.0044	2.0376			-3.6546	2.7377	-3.5485	-1.4194
8	2.9985	1.998			-3.6546	2.7377	-3.5485	-1.4194
9	3.0001	1.9998					-3.5485	-1.4194
10	3	2						
11	3	2						
12	3	2						

Table 7: Iteration sequence of the SQP with Trust Region.

It is also clear from Table 8 that it has less function calls than any of the methods. The method does require additional quadratic programs, but it will reduce number of

iterations and the total number of QP solves using an appropriate penalty parameter, based on the iterations and function calls this is accordance with said (Nocedal & Wright, 1999, p. 554).

x_0	**Function Value**	**Iterations**	**Function Calls**	$\|\|\nabla_x L\|\|$
$(0, 0)$	$5.2583e-15$	12	42	$6.4743e-07$
$(3, 2)$	0	1	2	1
$(-4.5, 3)$	35.9298	8	30	$1.0158e-06$
$(-3, -4)$	72.8555	9	32	$3.0290e-07$

Table 8: Statistics of the SQP with trust region

References

Bagterp Jørgensen, J. (2018a). Quadratic Optimization, Introduction and Equality Constrained QP (Slides Week 5, 02612). Department of Applied Mathematics and Computer Science.

Bagterp Jørgensen, J. (2018b). Quadratic Programming, Equality Constrained QP (Slides Week 5). Department of Applied Mathematics and Computer Science.

Bagterp Jørgensen, J. (2019a). Convex Quadratic Programming Primal-Dual Interior Point Algorithm (Slides Week 6). Department of Applied Mathematics and Computer Science.

Bagterp Jørgensen, J. (2019b). Introduction to Constrained Optimization (Slides Week 1, 02612). Department of Applied Mathematics and Computer Science.

Bagterp Jørgensen, J. (2019c). Linear Programming Lecture 07B - Primal-Dual Interior-Point Algorithm (Slides Week 7). Department of Applied Mathematics and Computer Science.

Bagterp Jørgensen, J. (2019d). Numerical Methods for Constrained Optimization (Lecture Notes for 02612 Constrained Optimization). Springer.

Bagterp Jørgensen, J. (2019e). Quadratic Programming, Equality Constrained QP (Slides Week 5, 02612). Department of Applied Mathematics and Computer Science.

Bagterp Jørgensen, J. (2019f). Sensitivity Lecture 3 (Slides Week 3). Department of Applied Mathematics and Computer Science.

Bagterp Jørgensen, J. (2019g). Sequential Quadratic Programming (SQP), Introduction (Slides Week 8, 02612). Department of Applied Mathematics and Computer Science.

Cartis, C. (2009). Some disadvantages of a mehrotra-type primal-dual corrector interior point algorithm for linear programming. Applied Numerical Mathematics, 59(5), 1110–1119. doi:https://doi.org/10.1016/j.apnum.2008.05.006

Nocedal, J. & Wright, S. J. (1999). Numerical Optimization (Second Edition). Springer.

Rees, T. & Scott, J. (2014). The null-space method and its relationship with matrix factorizations for sparse saddle point systems.

Appendix

Problem 1

SensitivitiesEQP.m

EqualityQPSolver.m

RandomQP.m

Listing 12 Driver1 is acting as the main function for Problem 1

```
%% Problem 1.3: EqualityQPSolver and RandomQP
H = [6,2,1; 2,5,2;1,2,4];
g = [-8;-3;-3];
A = [1,0;0,1;1,1];
b = [3;0];

[x,lambda] = EqualityQPSolver(H,g,A,b);
eig(H)
%% Problem 1.5: Random QP generator
merr=zeros(1000,2);
%Run 1000 times
for i=1:1000
    n=randi(6);
    m=randi(6);
    [H,g,A,b, xT, lambdaT] = RandomQP(n,m);
    [x,lambda] = EqualityQPSolver(H,g,A,b);

    %Calculate error
    errx = abs(xT-x);
    errlambda = abs(lambdaT-lambda);
    merr(i,1) = mean(errx);
    merr(i,2) = mean(errlambda);
end
mean(merr)
%% Problem 1.7 Sensitivities
p = zeros(5,1);
[dx, dlambda, x_approx, lambda_approx] = SensitivitiesEQP(H,g,A,b,p);

% Try with [p1, p2, p3, p4, p5] = [0, 0, 1, 0, 1] and compare with new
% corresponding b and g.
p = [0, 0, 0, 1, 1]';
gp = g+p(1:3);
bp = b+p(4:5);

% First order Taylor approximations of solution is exact
[dx, dlambda, x_approx, lambda_approx] = SensitivitiesEQP(H,g,A,b,p);
[x,lambda] = EqualityQPSolver(H,gp,A,bp);
```

Problem 2

ActiveSetConc.m

ConstructEqQP.m

driver.m

KKTLDLSolve.m

KKTLUSolve.m

KKTRSSolve.m

KKTSystem.m

Problem 3

ConstructMarkowitz.m

SolveMarkowitz.m

Problem 4

driver.m

The implementation of Primal-Dual Interior-Point algorithm.

```
function [x,y,z,s,info,mu,iter] = PDPCIP(H,g,A,C,b,d,x,y,z,s)

%%
[n,m] = size(A);
% n variables
% m equality contrains
nc = length(y); % Number of equality constrains
mc = length(z); % Number of inequality consrains


maxit = 100;
tolL = 1.0e-5;
tolA = 1.0e-5;
tolC = 1.0e-5;
tolSZ = 1.0e-5;
tolmu = 1.0e-5;

eta = 0.995;

% Residuals
rL = H*x+g-A*y-C*z;
rA = b-A'*x;
rC = s+d-C'*x;
rSZ = diag(s)*diag(z)*ones(length(z),1);
mu = (z'*s)/mc;

% Converged
Converged = (norm(rL,inf) <= tolL) && ...
            (norm(rA,inf) <= tolA) && ...
            (norm(rC,inf) <= tolC) && ...
            (norm(rSZ,inf) <= tolSZ) && ...
            (abs(mu) <= tolmu);
```

```
33
34  %%
35  iter = 0;
36
37  while ~Converged && (iter<maxit)
38      iter = iter+1;
39
40      % =================================================================
41      % Form and Factorize Matrix
42      % =================================================================
43      zdivs = z./s;
44      sdivz = s./z;
45      H1 = H + C*diag(zdivs)*C';
46      K = [H1 -A; -A' zeros(m,m)];
47      [L,D,p] = ldl(K,'vector');      % factorization
48
49      % =================================================================
50      % Affine Step
51      % =================================================================
52      % Solve
53      xyaff = zeros(m+nc,1);
54      temp = rSZ./z;
55      temp2 = diag(zdivs)*(rC-temp);
56      rL1 = rL - C*temp2;
57      rhs = -[rL1; rA];
58      xyaff(p) = L'\(D\(L\rhs(p)));       % back substitution
59      xaff = xyaff(1:n);
60
61      % Find z and s
62      zaff = -diag(zdivs)*C'*xaff+temp2;
63      saff = -temp-diag(sdivz)*zaff;
64
65      % Step length
66      izaff = find(zaff < 0.0);
67      alpha1 = min([1.0; -z(izaff,1)./zaff(izaff,1)]);
68
69      isaff = find(saff < 0.0);
70      beta1 = min([1.0; -s(isaff,1)./saff(isaff,1)]);
71
72      alpha = min(alpha1,beta1);
73
74      % =================================================================
75      % Center Parameter and duality gap
76      % =================================================================
77      muaff = (z+alpha*zaff)'*(s+alpha*saff)/mc;
78      sigma = (muaff/mu)*(muaff/mu)*(muaff/mu);
79
80      % =================================================================
81      % Affine-Centering-Correction Direction
82      % =================================================================
83      rSZ1 = rSZ + saff.*zaff - sigma*mu*ones(mc,1);
84
85      % Solve
86      dxy = zeros(m+nc,1);
87      temp = rSZ1./z;
88      temp2 = diag(zdivs)*(rC-temp);
89      rL1 = rL - C*temp2;
```

```
    rhs = -[rL1; rA];

    dxy(p) = L'\(D\(L\rhs(p)));       % back substitution
    dx = dxy(1:n);
    dy = dxy(n+1:end);

    % Find z and s
    dz = -diag(zdivs)*C'*dx+temp2;
    ds = -temp-diag(sdivz)*dz;

    % Step length
    iz = find(dz < 0.0);
    alpha1 = min([1.0; -z(iz,1)./dz(iz,1)]);

    is = find(ds < 0.0);
    beta1 = min([1.0; -s(is,1)./ds(is,1)]);

    alpha = min(alpha1,beta1);

    % =============================================================
    % Update iteration
    % =============================================================
    alpha1 = eta*alpha;
    x = x + alpha1*dx;
    y = y + alpha1*dy;
    z = z + alpha1*dz;
    s = s + alpha1*ds;

    % Residuals
    rL = H*x+g-A*y-C*z;
    rA = b-A'*x;
    rC = s+d-C'*x;
    rSZ = diag(s)*diag(z)*ones(length(z),1);
    mu = (z'*s)/mc;

    % Converged
    Converged = (norm(rL,inf) <= tolL) && ...
            (norm(rA,inf) <= tolA) && ...
            (norm(rC,inf) <= tolC) && ...
            (norm(rSZ,inf) <= tolSZ) && ...
            (abs(mu) <= tolmu);
end

%%

% Return solution
info = Converged;
if ~Converged
    x=[];
    mu=[];
    lambda=[];
end
```

Listing 13: PDPCIP is a function of the Primal-Dual Interior-Point program

Problem 5

TestLPsolver.m

The implementation of Primal-Dual Interior-Point algorithm for the linear constrained problem.

```
function [x,info,mu,lambda,iter] = LPippd(g,A,b,x)
% LPIPPD   Primal-Dual Interior-Point LP Solver
%
%          min  g'*x
%           x
%          s.t. A x  = b      (Lagrange multiplier: mu)
%                 x >= 0      (Lagrange multiplier: lamba)
%
% Syntax: [x,info,mu,lambda,iter] = LPippd(g,A,b,x)
%
%         info = true   : Converged
%              = false  : Not Converged

% Created: 04.12.2007
% Author : John Bagterp Jørgensen
%          IMM, Technical University of Denmark

%%
[m,n]=size(A);

maxit = 100;
tolL = 1.0e-9;
tolA = 1.0e-9;
tols = 1.0e-9;

eta = 0.99;

lambda = ones(n,1);
mu = zeros(m,1);

% Compute residuals
rL = g - A'*mu - lambda;    % Lagrangian gradient
rA = A*x - b;               % Equality Constraint
rC = x.*lambda;             % Complementarity
s = sum(rC)/n;              % Duality gap

% Converged
Converged = (norm(rL,inf) <= tolL) && ...
            (norm(rA,inf) <= tolA) && ...
            (abs(s) <= tols);
%%
iter = 0;
while ~Converged && (iter<maxit)
    iter = iter+1;

    % ======================================================================
    % Form and Factorize Hessian Matrix
    % ======================================================================
```

```
    xdivlambda = x./lambda;
    H = A*diag(xdivlambda)*A';
    L = chol(H,'lower');

    % =====================================================================
    % Affine Step
    % =====================================================================
    % Solve
    tmp = (x.*rL + rC)./lambda;
    rhs = -rA + A*tmp;

    dmu = L'\(L\rhs);
    dx = xdivlambda.*(A'*dmu) - tmp;
    dlambda = -(rC+lambda.*dx)./x;

    % Step length
    idx = find(dx < 0.0);
    alpha = min([1.0; -x(idx,1)./dx(idx,1)]);

    idx = find(dlambda < 0.0);
    beta = min([1.0; -lambda(idx,1)./dlambda(idx,1)]);

    % =====================================================================
    % Center Parameter
    % =====================================================================
    xAff = x + alpha*dx;
    lambdaAff = lambda + beta*dlambda;
    sAff = sum(xAff.*lambdaAff)/n;

    sigma = (sAff/s)^3;
    tau = sigma*s;

    % =====================================================================
    % Center-Corrector Step
    % =====================================================================
    rC = rC + dx.*dlambda - tau;

    tmp = (x.*rL + rC)./lambda;
    rhs = -rA + A*tmp;

    dmu = L'\(L\rhs);
    dx = xdivlambda.*(A'*dmu) - tmp;
    dlambda = -(rC+lambda.*dx)./x;

    % Step length
    idx = find(dx < 0.0);
    alpha = min([1.0; -x(idx,1)./dx(idx,1)]);

    idx = find(dlambda < 0.0);
    beta = min([1.0; -lambda(idx,1)./dlambda(idx,1)]);

    % =====================================================================
    % Take step
    % =====================================================================
    x = x + (eta*alpha)*dx;
    mu = mu + (eta*beta)*dmu;
```

```
    lambda = lambda + (eta*beta)*dlambda;

    % ======================================================================
    % Residuals and Convergence
    % ======================================================================
    % Compute residuals
    rL = g - A'*mu - lambda;    % Lagrangian gradient
    rA = A*x - b;               % Equality Constraint
    rC = x.*lambda;             % Complementarity
    s = sum(rC)/n;              % Duality gap

    % Converged
    Converged = (norm(rL,inf) <= tolL) && ...
                (norm(rA,inf) <= tolA) && ...
                (abs(s) <= tols);
end

%%
% Return solution
info = Converged;
if ~Converged
    x=[];
    mu=[];
    lambda=[];
end
```

Listing 14: LPIPPD is a function of the Primal-Dual Interior-Point program for a linear constrained problem

Problem 6

DRIVER.M

EQQP.M

NLPCON.M

OBJ.M

OBJ1.M

OBJ2.M

The implementation of a simple SQP algorithm.

```
function [x, stat] = NewtonSQP(fundfun,consfun,x0,y0,varargin)
% y is Lagrange multiplier

maxit = 100*length(x0);
tol   = 1e-5;


stat.converged = false; % converged
stat.nfun = 0; % number of function calls
stat.iter = 0; % number of iterations

% Initial iteration
```

```
x = x0;
it = 0;
B = eye(numel(x0));
[f,df,d2f] = feval(fundfun,x,varargin{:});
[c,dc,d2c] = feval(consfun,x,varargin{:});
[~,y0] = EQQP(B,df,dc,c);
y=y0;
dL_2 = df - dc*y;
converged = (norm(dL_2,'inf') <= tol && norm(c) <= tol);
stat.nfun = 2;

% Store data for plotting
stat.X = x;
stat.Y = y;
stat.F = f;
stat.C = c;
stat.dF = df;
stat.dC = dc;
stat.d2F = d2f;
stat.d2C = d2c;
stat.Errc = norm(c, "inf");
stat.ErrL = norm(dL_2, "inf");


    while ~converged && (it < maxit)
        % updating the iteration number
        it = it + 1;

        % Computing the Hessian
        H = d2f;
        for i = 1:length(y0)
            H = H - y(i)*d2c(:,:,i);
        end

        % Solve equality constriant
        [p,y] = EQQP(H,df,dc,-c);

        % Take step
        x = x + p;

        % Function evaluation
        [f,df,d2f] = feval(fundfun,x,varargin{:});
        [c,dc,d2c] = feval(consfun,x,varargin{:});
        stat.nfun = stat.nfun + 2;

        dL = df - dc*y;

        converged = (norm(dL,'inf') <= tol && norm(c,'inf') <= tol);
        stat.X   = [stat.X x];
        stat.Y   = [stat.Y y];
        stat.F   = [stat.F f];
        stat.C   = [stat.C c];
        stat.dF  = [stat.dF df];
        stat.dC  = [stat.dC dc];
        stat.d2F = [stat.d2F d2f];
        stat.d2C = [stat.d2C d2c];
```

```
        stat.Errc = [stat.Errc norm(c, "inf")];
        stat.ErrL = [stat.ErrL norm(dL,"inf")];
    end


if ~converged
    x = [];
end
stat.converged = converged;
stat.iter = it;
```

Listing 15: NewtonSQP is a function of the SQP program to solve the non-linear equality constrained problem in Problem 6.

The implementation of a SQP with damped BFGS algorithm.

```
function [x, stat] = NewtonSQP_BFGS(fundfun,consfun,x0,y0,varargin)
% y is Lagrange multiplier

maxit = 100*length(x0);
tol   = 1e-6;


stat.converged = false; % converged
stat.nfun = 0; % number of function calls
stat.iter = 0; % number of iterations

% Initial iteration
x = x0;
it = 0;
B = eye(numel(x0));
[f,df,d2f] = feval(fundfun,x,varargin{:});
[c,dc,d2c] = feval(consfun,x,varargin{:});
[~,y0] = EQQP(B,df,dc,c);
y=y0;
%Compute dL(x,y_new)
dL_2 = df - dc*y;
converged = (norm(dL_2,'inf') <= tol && norm(c) <= tol);
stat.nfun = 2;


% Store data for plotting
stat.X = x;
stat.Y = y;
stat.F = f;
stat.C = c;
stat.B = B;
stat.dF = df;
stat.dC = dc;
stat.d2F = d2f;
stat.d2C = d2c;
stat.Errc = norm(c, "inf");
stat.ErrL = norm(y, "inf");


    while ~converged && (it < maxit)
```

```
        % updating the iteration number
        it = it + 1;


        % Solve equality constriant
        [p,y] = EQQP(B,df,dc,-c);


        % Take step
        x = x + p;


        % Function evaluation
        [f,df,d2f] = feval(fundfun,x,varargin{:});
        [c,dc,d2c] = feval(consfun,x,varargin{:});
        stat.nfun = stat.nfun + 2;

        dL = df - dc*y;

        %compute q
        q = dL - dL_2;

        %Update Hessian by modified BFGS
        if ( p'*q >= 0.2*p'*B*p)
            theta = 1;
        else
            theta = ( 0.8*p'*B*p ) /(p'*B*p - p'*q );
        end

        %Weighting
        r = theta*q + (1-theta)*B*p;

        dL_2=dL;
        %Approximate hessian
        B = B + (r*r')/(p'*r) - ((B*p)*(B*p)')/(p'*B*p);

        converged = (norm(dL,'inf') <= tol && norm(c) <= tol);
        stat.X   = [stat.X x];
        stat.Y   = [stat.Y y];
        stat.F   = [stat.F f];
        stat.C   = [stat.C c];
        stat.B   = [stat.B B];
        stat.dF  = [stat.dF df];
        stat.dC  = [stat.dC dc];
        stat.d2F = [stat.d2F d2f];
        stat.d2C = [stat.d2C d2c];
        stat.Errc = [stat.Errc norm(c, "inf")];
        stat.ErrL = [stat.ErrL norm(dL,"inf")];
    end

stat.converged = converged;
stat.iter = it;
if ~converged
    x = [];
end
stat.converged = converged;
```

```
97  stat.iter = it;
```

Listing 16: NewtonSQP_BFGS.m is a function of the SQP program to solve the non-linear equality constrained problem in Problem 6 using damped BFGS.

The implementation of a SQP with damped BFGS and line search algorithm.

```
1   function [x, stat] = NewtonSQP_lineSearch(fundfun,consfun,x0,y0,varargin)
2   % l_opt is Lagrange multiplier
3   rng(2)
4   maxit = 100*length(x0);
5   tol   = 1e-6;
6   k = 0;
7   reg1 = 0.9;
8   stat.converged = false; % converged
9   stat.nfun = 0; % number of function calls
10  stat.iter = 0; % number of iterations
11
12  % Initial iteration
13  x = x0;
14
15  B = eye(numel(x0));
16  [f,df,d2f] = feval(fundfun,x,varargin{:});
17  [c,dc,d2c] = feval(consfun,x,varargin{:});
18  [~,y0] = EQQP(B,df,dc,c);
19  y=y0;
20  lambda = abs(y);
21  %Initialise gradient lagrangian
22  dL_2 = df - dc*y;
23  converged = (norm(dL_2,'inf') <= tol && norm(c, 'inf') <= tol);
24  stat.nfun = 2;
25
26  % Store data for plotting
27  stat.X = x;
28  stat.Y = y;
29  stat.F = f;
30  stat.C = c;
31  stat.B = B;
32  stat.dF = df;
33  stat.dC = dc;
34  stat.d2F = d2f;
35  stat.d2C = d2c;
36  stat.Errc = norm(c, "inf");
37  stat.ErrL = norm(dL_2, "inf");
38
39  while ~converged && (k < maxit)
40
41      % updating the iteration number
42      k = k + 1;
43
44      % Solve equality constriant
45      [p,l_opt] = EQQP(B,df,dc,-c);
46
47      pl = l_opt - y;
48
49      %%%%  Line Search
```

```
    stop = 0;
    alpha = 1;
    %Powell
    lambda = max(abs(l_opt),1/2*(lambda+abs(l_opt)));
    cls = f + lambda'*abs(c);
    b = df'*p - lambda'*abs(c);

    %START LINE SEARCH
    while stop == 0
        xk = x + alpha*p;

        [f,~,~] = feval(fundfun,xk,varargin{:});
        [c,~,~] = feval(consfun,xk,varargin{:});
        stat.nfun = stat.nfun + 2;
        phi_alpha = f + lambda'*abs(c);
        if phi_alpha <= (cls + (1-reg1)*b*alpha )
            stop = 1;
        else
            %Find good alpha for step length
            a = (phi_alpha - (cls + b*alpha))/(alpha^2);
            alpha_min = -b/(2*a);
            alpha = min( reg1*alpha,max(alpha_min,(1-reg1)*alpha));
        end
    end

    x = xk; %new

    %update lagrangian
    y = y + alpha*pl;
%       % Function evaluation
    [f,df,d2f] = feval(fundfun,x,varargin{:});
    [c,dc,d2c] = feval(consfun,x,varargin{:});
%       stat.nfun = stat.nfun + 2;

    %%%% BFGS approximation
    dL = df - dc*y;

%       %compute q
    q = dL - dL_2;
%
%       %Update Hessian by modified BFGS
    if ( p'*q > 0.2*p'*B*p)
         theta = 1;
    else
         theta = ( 0.8*p'*B*p ) /(p'*B*p - p'*q );
    end
%
    r = theta*q + (1-theta)*B*p;
%Hessian approximation
    B = B + (r*(r'))/(p'*r) - ((B*p)*((B*p)'))/(p'*B*p);
    dL_2 = dL;

    converged = (norm(dL,'inf') <= tol && norm(c, 'inf') <= tol);
    stat.X   = [stat.X x];
    stat.Y   = [stat.Y y];
    stat.F   = [stat.F f];
```

```
    stat.C   = [stat.C c];
    stat.B   = [stat.B B];
    stat.dF  = [stat.dF df];
    stat.dC  = [stat.dC dc];
    stat.d2F = [stat.d2F d2f];
    stat.d2C = [stat.d2C d2c];
    stat.Errc = [stat.Errc norm(c, "inf")];
    stat.ErrL = [stat.ErrL norm(dL_2,inf)];
end

if ~converged
    x = [];
end
stat.converged = converged;
stat.iter = k;
```

Listing 17: NewtonSQP_lineSearch.m with damped BFGS and line search.

Problem 7

ConFun.m

Driver.m

ObjFun.m

PlotMe.m

```
function [x, stat] = SQP_BFGS_ineq(ObjFun1,ConFun1,x0,y0)
% y is Lagrange multiplier

maxit = 100*length(x0);
tol   = 1e-6;
setQP = optimoptions('quadprog','display','off');

stat.converged = false; % converged
stat.nfun = 0; % number of function calls
stat.iter = 0; % number of iterations

% Initial iteration
x = x0;
it = 0;
B = eye(numel(x0));

[f,df,d2f] = feval(ObjFun1,x);
[c,dc] = feval(ConFun1,x);
y=y0;
%Compute dL(x,y_new)
dL_2 = df - dc*y;
converged = 0;
stat.nfun = 2;

% Store data for plotting
stat.X = x;
stat.Y = y;
```

```
stat.F = f;
stat.C = c;
stat.B = B;
stat.dF = df;
stat.dC = dc;
stat.d2F = d2f;
stat.Errc = norm(c, "inf");
stat.ErrL = norm(y, "inf");


    while ~converged && (it < maxit)
        % updating the iteration number
        it = it + 1;

        % Solve equality constriant
        [p,~,~,~,l] = quadprog(B,df,-dc',c,[],[],[],[],[],setQP);
        y=l.ineqlin;

        % Take step
        x = x + p;

        % Function evaluation
        [f,df,d2f] = feval(ObjFun1,x);
        [c,dc] = feval(ConFun1,x);
        stat.nfun = stat.nfun + 2;

        dL = df - dc*y;

        %compute q
        q = dL - dL_2;

        %Update Hessian by modified BFGS
        if ( p'*q >= 0.2*p'*B*p)
            theta = 1;
        else
            theta = ( 0.8*p'*B*p ) /(p'*B*p - p'*q );
        end

        r = theta*q + (1-theta)*B*p;
        dL_2=dL;
        %Approximate Hessian
        B = B + (r*r')/(p'*r) - ((B*p)*(B*p)')/(p'*B*p);



        converged =(norm(p,'inf') < tol);
        stat.X   = [stat.X x];
        stat.Y   = [stat.Y y];
        stat.F   = [stat.F f];
        stat.C   = [stat.C c];
        stat.B   = [stat.B B];
        stat.dF  = [stat.dF df];
        stat.dC  = [stat.dC dc];
        stat.d2F = [stat.d2F d2f];
        stat.Errc = [stat.Errc norm(c, "inf")];
        stat.ErrL = [stat.ErrL norm(dL,"inf")];
```

```
    end

stat.converged = converged;
stat.iter = it;
if ~converged
    x = [];
end
stat.converged = converged;
stat.iter = it;
```

Listing 18: SQP ICQ WITH DAMPED BFGS

```
function [x, stat] = SQP_BFGS_LS_ineq2(ObjFun1,ConFun1,x0,y0)
% y is Lagrange multiplier

maxit = 100*length(x0);
tol   = 1e-6;
setQP = optimoptions('quadprog','display','off');

stat.converged = false; % converged
stat.nfun = 0; % number of function calls
stat.iter = 0; % number of iterations


% Initial iteration
x = x0;
it = 0;
eta = 0.4; % (0,0.5)
tau = 0.8; % (0,1)
rho = 0.8; % (0,1)
B = eye(numel(x0));

[f,df,d2f] = feval(ObjFun1,x);
[c,dc] = feval(ConFun1,x);
l_k=y0;
%Compute dL(x,y_new)
converged =0;
stat.nfun = 2;

% Store data for plotting
stat.X = x;
stat.Y = l_k;
stat.F = f;
stat.C = c;
stat.B = B;
stat.dF = df;
stat.dC = dc;
stat.d2F = d2f;


while ~converged && (it < maxit)
    % updating the iteration number
    it = it + 1;

    % Solve equality constriant
    [p,~,~,~,l] = quadprog(B,df,-dc',c,[],[],[],[],[],setQP);
```

```
45
46
47      l_hat=l.ineqlin;
48
49      %P_lambda:
50      pl = l_hat - l_k;
51
52      %%%%%  Back Tracking Line Search
53      % Choose mu k
54      mu = max(((df'*p+(1/2)*p'*B*p)/((1-rho)*norm(c,1))),0); %new
55
56      %Set alpha
57      alpha = 1;
58
59      [f1,~,~] = feval(ObjFun1,x+alpha*p);
60      [c1,~] = feval(ConFun1,x+alpha*p);
61      %Calculate phi, l1 merit function, see week 8, slide 23:
62      phi_1 = f1+mu*norm(c1,1); %perhaps update c
63      %directional derivative D is found p. 542
64      phi_2 = f + mu*norm(c,1) + eta*alpha*(df'*p-mu*norm(c,1));
65      %Scale alpha until criterion is satisfied
66      while phi_1 > phi_2
67          alpha = tau*alpha;
68          stat.nfun = stat.nfun + 2;
69          [f1,~,~] = feval(ObjFun1,x+alpha*p);
70          [c1,~] = feval(ConFun1,x+alpha*p);
71          phi_1 = f1+mu*norm(c1,1);
72          phi_2 = f + mu*norm(c,1) + eta*alpha*(df'*p-mu*norm(c,1));
73      end
74
75      %compute gradient lagrangian
76      dL_old = df - dc*l_k;
77
78      %update lagrangian
79      l_k = l_k + alpha*pl;
80
81      %Take a step
82      x=x+alpha*p;
83
84       % Function evaluation
85      [f,df,d2f] = feval(ObjFun1,x);
86      [c,dc] = feval(ConFun1,x);
87      stat.nfun = stat.nfun + 2;
88
89      dL_new = df - dc*l_k;
90
91
92      %compute q
93      q = dL_new - dL_old;
94      p=alpha*p;
95      %Update Hessian by modified BFGS
96      if ( p'*q >= 0.2*p'*B*p)
97          theta = 1;
98      else
99          theta = ( 0.8*p'*B*p ) /(p'*B*p - p'*q );
100     end
101
```

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

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


    converged =(norm(p,'inf') < tol);
    stat.X   = [stat.X x];
    stat.Y   = [stat.Y l_k];
    stat.F   = [stat.F f];
    stat.C   = [stat.C c];
    stat.B   = [stat.B B];
    stat.dF  = [stat.dF df];
    stat.dC  = [stat.dC dc];
    stat.d2F = [stat.d2F d2f];
end

stat.converged = converged;
stat.iter = it;
if ~converged
    x = [];
end
stat.converged = converged;
stat.iter = it;
```

Listing 19: SQP ICQ with damped BFGS and Line Search

The implementation of a SQP with damped BFGS and line search algorithm.

```
function [x, stat] = SQP_TRUST_REG(ObjFun1,ConFun1,x0,y0)
%SQP TRUST REGION
debug=0;
maxit = 100*length(x0);
setQP = optimoptions('quadprog','display','off');
setLP = optimoptions('linprog','display','off');
stat.converged = false; % converged
stat.nfun = 0; % number of function calls
stat.iter = 0; % number of iterations

%Initialise
x = x0;
[f,df,d2f] = feval(ObjFun1,x);
[c,dc] = feval(ConFun1,x);
stat.nfun = 2;
it = 0;
%Set hyper params
mu1=1e-5; %1e-4 also seems to work great
tol   = 1e-6;
dkmax=2;
dk = 1;  %set this for different solution, best for 1?
eps1 = 0.4;
eps2 = 0.4;
eta = 0.2; %0.1-0.3
gamma = 0.4; %0.3?
converged = 0;
B = eye(numel(x0));
m = size(c,1);
```

```
n = size(x,1);
t=ones(n,1);

% Store data for plotting
stat.X = x;
stat.F = f;
stat.C = c;
stat.B = B;
stat.dF = df;
stat.dC = dc;
stat.d2F = d2f;
stat.Errc = norm(c, "inf");
stat.ErrL = norm(y0, "inf");
%mk is a piecewise linear model (18.56)
mk = @(c,dc,p) sum(max(0,-(c + dc'*p)));

while ~converged && (it < maxit)
    % updating the iteration number
    it = it + 1;
    %Used to solve 18.50
    B2 = [B, zeros(n,m); zeros(m,n), zeros(m,m)];
    %Add slack to g
    g = [df; mu1 * t];
    %t adds extra dimensions, multiply with - to swap inequality
    A = -[dc', eye(n)];
    %inequality limits
    b=c;
    if(debug==1)
        A
        B2
        g
    end
    % ||p||_inf < delta_k -> can go from -dk to dk
    % slack variables can go from t>=0 -> 0 to inf
    lb = [-dk*t; 0*t];
    ub = [dk*t; Inf*t];

    % Solve 18.50, to get slack t and p
    [pslack,~,~,~,l] = quadprog(B2,g,A,b,[],[],lb,ub,[],setQP);
    %get solution for p
    l_hat=l.ineqlin(1:2);
    p = pslack(1:2);
    converged =(norm(p,'inf') < tol);
    if(converged==1)
        stat.X   = [stat.X x];
        stat.F   = [stat.F f];
        stat.C   = [stat.C c];
        stat.B   = [stat.B B];
        stat.dF  = [stat.dF df];
        stat.dC  = [stat.dC dc];
        stat.d2F = [stat.d2F d2f];
        break;
    end
    m_k=mk(c,dc,p);
    %%% PENALTY UPDATE AND STEP COMPUTATION %%%
    if abs(m_k) <= tol
```

```
85          mu = mu1;
86      else
87          p_lp = linprog([zeros(n,1); ones(m,1)], A, b, [], [],lb,ub,[], ...
88                      setLP);
89          pinf = p_lp(1:2);
90          mkinf=mk(c,dc,pinf);
91          if abs(mkinf) <= tol
92              %new constrained inequality problem
93              mkpos=m_k;
94              mu = mu1;
95              %keep doing until below is satisfied
96              while(abs(mkpos)>tol)
97                  %multiply mu by a factor, thats how we get muk>muk-1
98                  mu = mu*5;
99                  %solve the quadprog to make p(mu^pos) and mk(mu^+) satisfy
100                 [muslack,~,~,~,~] = quadprog(B2,[df; mu * t],A,b,[],[], ...
101                                 lb,ub,[],setQP);
102                 pmupos = muslack(1:2);
103                 mkpos = mk(c,dc,pmupos);
104                 if(debug==1)
105                     pmupos
106                     mkpos
107                     mu
108                 end
109             end
110         else
111             mk0 = mk(c,dc,zeros(2,1));
112             mkpos = m_k;
113             mu = mu1;
114             %Keep doing until below is satisfied
115             while((mk0-mkpos) < (eps1*(mk0-mkinf)))
116                 %scale
117                 mu = mu*5;
118                 [muslack,~,~,~,~] = quadprog(B2,[df; mu * t],A,b,[],[], ...
119                                 lb,ub,[],setQP);
120                  %solve the quadprog to make p(mu^pos) and mk(mu^+) satisfy
121                  pmupos = muslack(1:2);
122                  mkpos = mk(c,dc,pmupos);
123                 if(debug==1)
124                     pmupos
125                     mkpos
126                     mu
127                 end
128             end
129         end
130     end
131     %Quadprog again...
132     [muslack,~,~,~,~] = quadprog(B2,[df; mu * t],A,b,[],[], ...
133                                 lb,ub,[],setQP);
134     pmupos = muslack(1:2);
135     %find these variables piece wise linear
136     mk0 = mk(c,dc,zeros(2,1));
137     mkpos = mk(c,dc,pmupos);
138     %Calculate (18.57)
139     q0 = f + mu*mk0;
140     qp = f + df'*pmupos + 0.5*pmupos'*B*pmupos + mu*mkpos;
```

```
    if(debug==1)
        q0
        qp
    end
    %need to satisfy this scale mu until satisfied
    while((q0-qp) < eps2*mu*(mk0-mkpos))
        mu=mu*5;
        [muslack,~,~,~,~] = quadprog(B2,[df; mu * t],A,b,[],[], ...
                        lb,ub,[],setQP);
        %update
        pmupos=muslack(1:2);
        mkpos = mk(c,dc,pmupos);
        q0 = f + mu*mk0;
        qp = f + df'*pmupos + 0.5*pmupos'*B*pmupos + mu*mkpos;
    end

    %set mu_k = mupos and p=p(mu+)
    mu1=mu;
    p=pmupos;
    %%%% COMPLETE %%%%
    [f1,~,~] = feval(ObjFun1,x+p);
    [c1,~] = feval(ConFun1,x+p);
    stat.nfun = stat.nfun + 2;
    %%% MERIT FUNCTIONS %%%%
    % (18.48) to judge our step
    phi1 = f + mu1*sum(max(-c, 0));
    phi1p = f1 + mu1*sum(max(-c1, 0));
    if(debug==1)
        phi1
        phi1p
    end
    rhok = (phi1 - phi1p) / (q0 - qp);
    if rhok > eta
        %Update parameters
        dLold = df - dc*l_hat;
        x = x + p;

        % Function evaluation
        [f,df,d2f] = feval(ObjFun1,x);
        [c,dc] = feval(ConFun1,x);
        stat.nfun = stat.nfun + 2;

        dLnew = df - dc*l_hat;
        %compute q
        q = dLnew - dLold;

        %Update Hessian by modified damped BFGS
        if ( p'*q >= 0.2*p'*B*p)
            theta = 1;
        else
            theta = ( 0.8*p'*B*p ) /(p'*B*p - p'*q );
        end

        r = theta*q + (1-theta)*B*p;
        %approximate hessian
        B = B + (r*r')/(p'*r) - ((B*p)*(B*p)')/(p'*B*p);
```

```
197
198         %update radius of circle
199         dk = min(dk * 1.5,dkmax);
200     else
201         %update radius of circle
202         dk = min(gamma*norm(p, 'inf'), dkmax);
203     end
204     if(debug==1)
205         dk
206     end
207     %Save stuff
208     converged =(norm(p,'inf') < tol);
209     stat.X   = [stat.X x];
210     stat.F   = [stat.F f];
211     stat.C   = [stat.C c];
212     stat.B   = [stat.B B];
213     stat.dF  = [stat.dF df];
214     stat.dC  = [stat.dC dc];
215     stat.d2F = [stat.d2F d2f];
216     stat.Errc = norm(c, "inf");
217     stat.ErrL = norm(dLnew, "inf");
218 end
219
220 stat.converged = converged;
221 stat.iter = it;
222 if ~converged
223     x = [];
224 end
225 stat.converged = converged;
226 stat.iter = it;
```

Listing 20: SQP ICQ with trust region