DTU Compute
Department of Applied Mathematics and Computer Science

Exercises

02612 Constrained Optimization

Anton Ruby Larse (s174356)

Kongens Lyngby 2021

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Contents

CHAPTER 1

Mangler at blive lavet

1. ex referencer er ikke automatiske så hvis man bytter om på opgave rækkefølgen dør det
2. ex2.9
3. ex2.10
4. ex2.11
5. ex2.12
6. ex2.13
7. ex3.10
8. ex4.2
9. Tilføj notationsforklaring så λ er for all lagrange multi for equality, μ er for alle inequality, x er for variable, s er for slack osv.
10. I LP int. method mangler der at redegøres for om η kan vælges adaptivt, normal equations og cholesky fac(At det man skal factorize er symmetrisk) samt en pseudokode.
11.
12. Jeg kunne godt tænke mig at det bliver delt op i active set methods hvor der gives en generel intro, så bliver metoder for LP og QP gennemgået og til sidst et eksempel(makowitch).
13. afsnittet til interior point methods skal have et generelt teori afsnit til at starte med der forklare man kan gå barrier method eller primal dual vejen.

Forslag til opbygning

1. Først generelt om optimering som exercise 1.
2. Så om factorizations og effiktiv implementering

3. Så equality constrained hvor de forskellige factorizations bliver brugt. Dette kan evt være samme afsnit som factorizations
4. Så et afsnit om active set methods som bliver brugt på inequality
5. så et afsnit om interior methods

KUNNE VÆRE MEGA FEDT AT LAVE EN SERIE MED JUPYTER NOTEBOOKS I JULIA

CHAPTER 2

Quadratic Optimization, Sensitivity and Duality

Consider the problem

$$\min_x \quad f(x) = 3x_1^2 + x_1x_2 + 2.5x_2^2 + 2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3 \tag{2.1a}$$
$$\text{s.t.} \quad x_1 + x_3 = 3, \tag{2.1b}$$
$$x_2 + x_3 = 0 \tag{2.1c}$$

In the form

$$\min_x \quad f(x) = \frac{1}{2}x^T Hx + g^T x \tag{2.2a}$$
$$\text{s.t.} \quad A^T x = b \tag{2.2b}$$

2.1 Exercise 1.1

What are H, g, A and b

First we will consider H. H consists of all the second order terms in 1.1a where the diagonal is multiplied by 2, i.e.

$$H = \begin{bmatrix} 2ax_1^2 & dx_1x_2 & ex_1x_3 \\ dx_1x_2 & 2bx_2^2 & fx_2x_3 \\ ex_1x_3 & fx_2x_3 & 2cx_3^2 \end{bmatrix} = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix} \tag{2.3}$$

Next we have g. It consists of all first order term in 1.1a.

$$g = \begin{bmatrix} -8 \\ -3 \\ -3 \end{bmatrix} \tag{2.4}$$

Then we have A which consists of the lhs of the constraints 1.1b and 1.1c.

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \tag{2.5}$$

Lastly we have b which consists of the rhs of the constraints 1.1b and 1.1c.

$$b = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \tag{2.6}$$

2.2 Exercise 1.2

Write the KKT optimality conditions.

From the lecture notes [Jør21] page 44 we know the KKT conditions for a constrained optimization problem is equivalent to the first order condition for a unconstrained problem which follows proposition 2.10. Before we can state the first order conditions we need the Lagrangian.

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \varepsilon} \lambda_i c_i(x) \tag{2.7}$$

We are working with a equality constrained problem and hence no inequality constraints are present. Therefore we only need to consider the two first conditions.

$$\nabla_x \mathcal{L}(x, \lambda) = \nabla_x f(x) - \sum_{i \in \varepsilon} \lambda_i \nabla_x c_i(x) = 0 \tag{2.8}$$

$$c_i(x) = 0 \tag{2.9}$$

For our problem 2.8 becomes

$$\begin{aligned} \nabla_x \mathcal{L}(x, \lambda) &= Hx + g - A\lambda \\ &= \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix} + \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} -8 \\ -3 \\ -3 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \end{aligned} \tag{2.10}$$

The second condition 2.9 is derived from setting $\nabla_\lambda \mathcal{L}(x, \lambda) = 0$. Therefore we need to consider the sign in front of the second term when deriving the second condtion for our problem.

$$\begin{aligned} \nabla_\lambda \mathcal{L}(x, \lambda) &= -c_i(x) \\ &= -A^T x + b \\ &= -\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned} \tag{2.11}$$

We can then combine 2.10 and 2.11

$$\begin{bmatrix} \nabla_x \mathcal{L}(x,\lambda) \\ \nabla_\lambda \mathcal{L}(x,\lambda) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} Hx - A\lambda \\ -A^T x \end{bmatrix} = \begin{bmatrix} -g \\ -b \end{bmatrix}$$
$$\begin{bmatrix} H & -A \\ -A^T & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} -g \\ -b \end{bmatrix} \tag{2.12}$$

2.3 Exercise 1.3

Make a function for [x,lambda]=EqualityQPSolver(H,g,A,b) **solution of equality constrained convex quadratic programs. hint: Consider and discuss which factorization to use when you factorize the KKT-matrix.**

We observe that H is positive definite and A has full column rank. Therefore by theorem 16.2 [NW06] we know the solution to 2.12 is a unique global solution.
Next we need to solve the KKT system 2.12. We know from section 16.2 in [NW06] that if $m \geq 1$ then the KKT matrix will be indefinite. We have 2 constraints so our KKT matrix is indefinite meaning we are not able to use the Cholesky factorization for example.
One factorization we can use when working with indefinite matrices is the gauss elimination(LU-factorization) but here we do not make use of the symmetry of the KKT matrix and we can also have problems with numerical stability as one can see in this Stack Exchange post.

Therefore we will choose to use the block LDL-factorization. It gives numeric stability and half the computation cost compared to the gauss elimination with pivoting. The following function is following page 455 in [NW06].

```
1  function [x,lambda]=EqualityQPSolver(H,g,A,b)
2  nulls = zeros(size(A,2),size(A,2));
3  KKT = [H -A;-A' nulls];
4  rhs = [-g; -b];
5  [L, B, P] = ldl(KKT);
6  z = L\(P'*rhs);
7  y = B\z;
8  zbar = L'\y;
9  sol = P*zbar;
10 x = sol(1:size(A,1));
11 lambda = sol(size(A,1)+1:size(A,2)+size(A,1));
12 end
```

Listing 2.1: The EqualityQPSolver

2.4 Exercise 1.4

Test your program on the above problem

We define H, g, A and b in matlab and run our function.

```
[x,lambda]=EqualityQPSolver(H,g,A,b)

x =
    2.0000
   -1.0000
    1.0000

lambda =
    3.0000
   -2.0000
```

We compare with the backslash operator

```
>> KKT\rhs

ans =

    2.0000
   -1.0000
    1.0000
    3.0000
   -2.0000
```

We get the same result so we conclude our program is correct.

2.5 Exercise 1.5

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

As earlier mentioned we know from theorem 16.1 and 16.2 in [NW06] that if H is symmetric and positive definite, and A has full column rank then our program will be convex and the solution to 2.12 will be a unique the global solution.
Therefore we will produce a procedure which can generate such symmetric and positive definite matricies.

```
function M = generateSymPosDef(n)
M = rand(n); % Random n by n matrix
M = 0.5*(M+M'); % Makes M symmetric
M = M + n*eye(n); % Makes M positive definite
end
```

We will now create a program which generates a H and an A. Then we generate a random x and λ for which we calculate g and b. This way we can test our program.

```
function [H,g,A,b,x,lambda] = generateRandomEQP(n,m)

H = rand(n); %Random n by n matrix
H = 0.5*(H+H') + n*eye(n); %Makes H symmetric and pos def

A = rand(n);
A = 0.5*(A+A')+n*eye(n); %Pos def to ensure full rank
A = A(:,1:m);

x = rand(n,1);
lambda = rand(m,1);

KKT = [H -A;-A' zeros(m)];
sol = [x; lambda];
rhs = KKT*sol;

g = -rhs(1:n);
b = -rhs(n+1:n+m);
end
```

We have then generate generate 100 programs with 6 variables 6 times where each time we increase the number of constrains by 1. Then we calculate the difference from the real x and λ, and our solvers output.

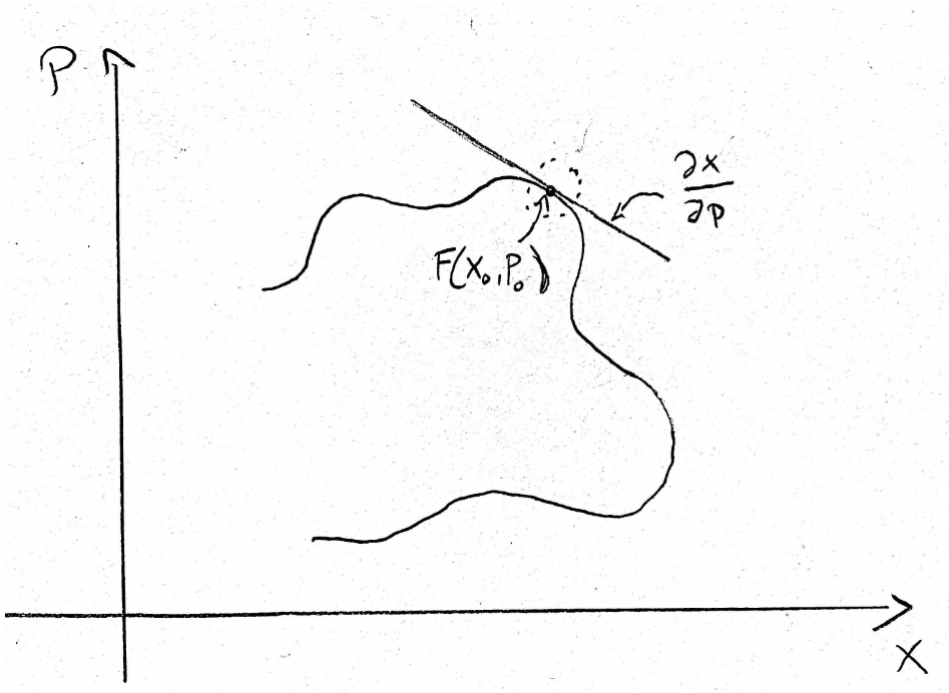
```
test_size = 100;
x_err = 0;
lambda_err = 0;
n = 6;
for j = 1:6
for i = 1:test_size
[H,g,A,b,x,lambda] = generateRandomEQP(n,j);
[xhat,lambdahat]=EqualityQPSolver(H,g,A,b);
x_err = x_err + sum(abs(xhat-x));
lambda_err = lambda_err + sum(abs(lambdahat-lambda));
end
end

error_x = x_err/test_size;
error_lambda = lambda_err/test_size;
```

The error we got in x was 3.7535e-15 and the error for λ was 3.0348e-15. With our generation of the EQP's we also ensure that $det(H)$ is always large and therefore far away from singular. If $det(H)$ was close to zero this method of solving the EQP would give a larger error.

2.6 Exercise 1.6

Write the sensitivity equations for the equality constrained convex QP.

We get the sensitivity equation for our equality constrained optimization problem by use of the implicit function theorem stated in section 2.8.1 [Jør21]. This theorem is based on the concept of implicit differentiation described in [3bl].
We parameterizes the solution of our problem by x and the parameters p as $F(x, p) = 0$ which conceptually can be visualized as follows:



We now want to find out how our current solution $F(x_0, p_0)$ changes when we change p by some small amount, i.e. we want to find $\frac{\partial x}{\partial p}$. This we know from implicit differentiation [3bl]

$$\frac{\partial x}{\partial p} = -xp^{-1} \tag{2.13}$$

So the implicit function theorem gives us a linear approximation of how x changes locally when changing the parameters p.
The specific situation of a equality constrained optimization problem is derived in

section 2.8 in [Jør21] and is given as

$$\begin{bmatrix}\nabla_p x(p) & \nabla_p \lambda(p)\end{bmatrix} = -\begin{bmatrix}\nabla^2_{xp}\mathcal{L}(x,\lambda;p) & -\nabla_p c(x,p)\end{bmatrix}\begin{bmatrix}\nabla^2_{xx}\mathcal{L}(x,\lambda;p) & -\nabla_x c(x,p) \\ -\nabla_x c(x,p) & 0\end{bmatrix}^{-1} \tag{2.14}$$

where

$$\nabla^2_{xp}\mathcal{L}(x,\lambda;p) = \nabla^2_{xp} f(x,p) - \sum_{i\in\varepsilon} \lambda_i \nabla^2_{xp} c_i(x,p) \tag{2.15}$$

$$\nabla^2_{xx}\mathcal{L}(x,\lambda;p) = \nabla^2_{xx} f(x,p) - \sum_{i\in\varepsilon} \lambda_i \nabla^2_{xx} c_i(x,p) \tag{2.16}$$

In the next exercise we see we are to return sensitivities w.r.t. g and b which means $p = \begin{bmatrix} g \\ b \end{bmatrix}$. Therefore now solve 2.14 with such p. First we find 2.15 and 2.16 for our problem.

$$\begin{aligned}\nabla^2_{xp}\mathcal{L}(x,\lambda;p) &= \nabla_p(Hx+g) - \nabla_p(A\lambda) \\ &= \begin{bmatrix} I \\ 0 \end{bmatrix}\end{aligned}$$

$$\begin{aligned}\nabla^2_{xx}\mathcal{L}(x,\lambda;p) &= \nabla_x(Hx+g) - \nabla_x(A\lambda) \\ &= H\end{aligned}$$

We proceed with $-\nabla_p c(x,p)$ and $-\nabla_x c(x,p)$

$$\begin{aligned}-\nabla_p c(x,p) &= \nabla_p(A^T x - b) \\ &= \begin{bmatrix} 0 \\ I \end{bmatrix}\end{aligned}$$

$$\begin{aligned}-\nabla_x c(x,p) &= \nabla_x(A^T x - b) \\ &= -A\end{aligned}$$

We can now insert everything in 2.14

$$\begin{aligned}\begin{bmatrix}\nabla_p x(p) & \nabla_p \lambda(p)\end{bmatrix} &= -\begin{bmatrix}\begin{bmatrix} I \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ I \end{bmatrix}\end{bmatrix}\begin{bmatrix} H & -A \\ -A^T & 0 \end{bmatrix}^{-1} \\ &= -\begin{bmatrix} H & -A \\ -A^T & 0 \end{bmatrix}^{-1}\end{aligned} \tag{2.17}$$

We observe that 2.17 is the inverse of our KKT matrix in 2.12. This is convenient if one is to implement a program where one both solves 2.12 and wants to return the sensitivities of x and λ because when solving 2.12 we invert the KKT matrix and then the sensitivities are almost free to obtain.

OBS OBS !!!! I NOTERNE FRA LECTURE 10 I SciML ER FORHOLDET MELLEM SENSITIVITET OG INVERS JOCOBIAN BESKREVET !!!!! OBS OBS

2.7 Exercise 1.7

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.

We calculate the sensitivities implementing what we found in ex 1.6. It is implemented in the program below.

```
function [x_sens,lam_sens] = sensitivity(H,A)
sol = -inv([H -A; -A' zeros(size(A,2),size(A,2))]);
x_sens = sol(:,1:size(A,1));
lam_sens = sol(:,size(A,1)+1:size(A,1)+size(A,2));
end
```

Listing 2.2: Program for calculating the sensitivities of x and λ w.r.t. g and b

The sensitivities can be interpreted as the gradient of $\begin{bmatrix} x \\ \lambda \end{bmatrix}$ w.r.t. $\begin{bmatrix} g \\ b \end{bmatrix}$ meaning

$$\begin{bmatrix} \nabla_p x(p) & \nabla_p \lambda(p) \end{bmatrix} = \begin{bmatrix} \frac{\partial x_1}{\partial g_1} & \frac{\partial x_2}{\partial g_1} & \frac{\partial x_3}{\partial g_1} & \frac{\partial \lambda_1}{\partial g_1} & \frac{\partial \lambda_2}{\partial g_1} \\ \frac{\partial x_1}{\partial g_2} & \frac{\partial x_2}{\partial g_2} & \frac{\partial x_3}{\partial g_2} & \frac{\partial \lambda_1}{\partial g_2} & \frac{\partial \lambda_2}{\partial g_2} \\ \frac{\partial x_1}{\partial g_3} & \frac{\partial x_2}{\partial g_3} & \frac{\partial x_3}{\partial g_3} & \frac{\partial \lambda_1}{\partial g_3} & \frac{\partial \lambda_2}{\partial g_3} \\ \frac{\partial x_1}{\partial b_1} & \frac{\partial x_2}{\partial b_1} & \frac{\partial x_3}{\partial b_1} & \frac{\partial \lambda_1}{\partial b_1} & \frac{\partial \lambda_2}{\partial b_1} \\ \frac{\partial x_1}{\partial b_2} & \frac{\partial x_2}{\partial b_2} & \frac{\partial x_3}{\partial b_2} & \frac{\partial \lambda_1}{\partial b_2} & \frac{\partial \lambda_2}{\partial b_2} \end{bmatrix} \tag{2.18}$$

So if we for example change b_1 with 1, $\begin{bmatrix} x \\ \lambda \end{bmatrix}$ should change by 1 times row 4 in 2.18. We check.

```
bny = [4;0];
[x, lambda] = EqualityQPSolver(H,g,A,bny)

[x; lambda] =
    2.4615
   -1.5385
    1.5385
    5.2308
   -2.6923

[x_sens,l_sens] = sensitivity(H,A);
b1_grad = [x_sens(4,:) l_sens(4,:)];

[x; lambda] + b1_grad' =
    2.4615
   -1.5385
    1.5385
    5.2308
   -2.6923
```

It is as we expected.

2.8 Exercise 1.8

Write the dual program of the equality constrained QP.

Since U in our primal problem is positive definite, $\mathcal{L}(., \lambda)$ is strictly convex and we can hence write our dual as a wolfe dual according to theorem 12.14 [NW06]. The wolfe dual is given as

$$\begin{aligned} \underset{x,\lambda}{\text{maximize}} \quad & \mathcal{L}(x, \lambda) \\ \text{subject to} \quad & \nabla_x \mathcal{L}(x, \lambda) = 0 \end{aligned} \tag{2.19}$$

We write our program as a wolfe dual

$$\begin{aligned} \underset{x,\lambda}{\text{maximize}} \quad & \frac{1}{2} x^T H x + g^T x - \lambda^T (A^T x - b) \\ \text{subject to} \quad & Hx + g - A\lambda = 0 \end{aligned}$$

We can rewrite the objective to

$$\frac{1}{2} x^T H x - g^T - \lambda^T A^T) x + \lambda^T b$$

Then we rewrite the constraint

$$\begin{aligned} Hx + g - A\lambda = 0 \quad &\Leftrightarrow \\ g - A\lambda = -Hx \quad &\Leftrightarrow \\ g^T - \lambda^T A^T = -x^T H^T \quad &\Leftrightarrow \\ g^T - \lambda^T A^T = -x^T H \end{aligned}$$

We substitute in to the objective and get the rewritten wolfe dual

$$\begin{aligned} \underset{x,\lambda}{\text{maximize}} \quad & -\frac{1}{2}x^T Hx + \lambda^T b \\ \text{subject to} \quad & Hx + g - A\lambda = 0 \end{aligned}$$

Lastly we multiply by -1 to get a minimization problem

$$\begin{aligned} \underset{x,\lambda}{\text{minimize}} \quad & \frac{1}{2}x^T Hx - \lambda^T b \\ \text{subject to} \quad & Hx + g - A\lambda = 0 \end{aligned} \tag{2.20}$$

2.9 Exercise 1.9

From section 2.7 in [Jør21] we know that for our convex equality constrained program, the first order conditions for the primal and dual problem are the same. We write up the Lagrangian and the KKT conditions for the dual where μ denotes the new Lagrange multipliers for the dual program.

$$\mathcal{L}_D(x, \lambda, \mu) = \frac{1}{2}x^T Hx - \lambda^T b - \mu^T(Hx + g - A\lambda) \tag{2.21}$$

and the KKT conditions

$$\nabla_x \mathcal{L}_D(x, \lambda, \mu) = Hx - \mu^T H \quad = 0 \tag{2.22}$$
$$\nabla_\lambda \mathcal{L}_D(x, \lambda, \mu) = -b + \mu^T A \quad = 0 \tag{2.23}$$
$$\nabla_x \mathcal{L}_D(x, \lambda, \mu) = -Hx - g + A\lambda = 0 \tag{2.24}$$

We can rewrite the system as

$$\begin{bmatrix} H & 0 & -H \\ 0 & 0 & A^T \\ -H & A & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \\ \mu \end{bmatrix} = \begin{bmatrix} 0 \\ b \\ g \end{bmatrix} \tag{2.25}$$

We solve the system

$$\begin{bmatrix} H & 0 & -H \\ 0 & 0 & A^T \\ -H & A & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ b \\ g \end{bmatrix} \begin{bmatrix} x \\ \lambda \\ \mu \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 1 \\ 3 \\ -2 \\ 2 \\ -1 \\ 1 \end{bmatrix}$$

We can see from 2.22 that x must equal μ at the optimal solution which we also see from the solved program. This also follows what theorem 12.13 in [NW06] says which is also called strong duality.

2.10 Exercise 1.10

We can solve the dual problem by applying the same technique used for the primal.

```
function [x,lambda,mu] = dualEQPsolver(H,g,A,b)
smallnull = zeros(size(A,2),size(A,2));
bignull = zeros(size(A,1),size(A,1));
Anull = 0*A;
KKT = [H Anull -H; Anull' smallnull A'; -H A bignull];
rhs = [zeros(length(g),1) ; b; g];
[L, B, P] = ldl(KKT);
z = L\(P'*rhs);
y = B\z;
zbar = L'\y;
sol = P*zbar;
x = sol(1:length(g));
lambda = sol(length(g)+1:length(g)+length(b));
mu = sol(length(g)+length(b)+1:length(g)*2+length(b));
end
```

Listing 2.3: Solver for the dual program of 1.2

For our specific program there is no advantage to get from solving the dual instead of the primal. We will only get a slower algorithm due the the larger matrix we would need to solve in the dual program.
Had H been semi definite instead of positive definite though can the dual solve the program where the primal would fail according to page 349 in [NW06].

CHAPTER 3

Equality Constrained Quadratic Optimization

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

$$\begin{array}{ll} \min_u & \frac{1}{2}\sum_{i=1}^{n+1}(u_i-\bar{u})^2 \\ \text{s.t.} & -u_1+u_n=-d_0 \\ & u_i-u_{i+1}=0 \quad i=1,2,\ldots,n-2 \\ & u_{n-1}-u_n-u_{n+1}=0 \end{array} \tag{3.1}$$

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

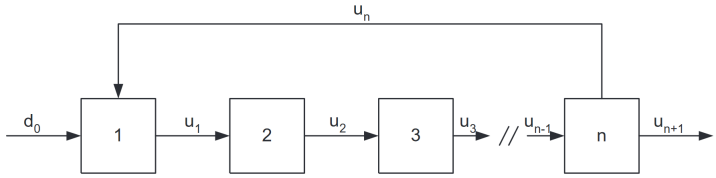


Figure 3.1: Directed graph representation of the constraints in 3.1. This graph represents a recycle system.

3.1 Exercise 2.1

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

$$\begin{array}{ll} \min_{x\in R^n} & \phi = \frac{1}{2}x^T H x + g^T x \\ \text{s.t.} & A^T x = b \end{array} \tag{3.2}$$

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

We write up ϕ for $n = 10$, $\bar{u} = 0.2$ and $d_0 = 1$

$$\begin{aligned} \phi &= \frac{1}{2}\sum_{i=1}^{11}(u_i - \bar{u})^2 \\ &= \frac{1}{2}\left((u_1 - \bar{u})^2 + (u_2 - \bar{u})^2 + \ldots + (u_{11} - \bar{u})^2\right) \\ &= \frac{1}{2}\left(u_1^2 + \bar{u}^2 - 2u_1\bar{u} + u_2^2 + \bar{u}^2 - 2u_2\bar{u} + \ldots + u_{11}^2 + \bar{u}^2 - 2u_{11}\bar{u}\right) \\ &= \frac{1}{2}\left(\begin{bmatrix} u_1 & u_2 & \ldots & u_{11}\end{bmatrix} I_{11} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{11}\end{bmatrix} - 2\begin{bmatrix}\bar{u} & \bar{u} & \ldots & \bar{u}\end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{11}\end{bmatrix} + 11\bar{u}^2\right) \\ &= \frac{1}{2}u^T I^{[11\times 11]}u - \bar{u}^{[1\times 11]}u + \frac{11}{2}\bar{u}^2 \end{aligned} \tag{3.3}$$

So

$$x = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{11}\end{bmatrix} \tag{3.4}$$

$$H = I^{[11\times 11]} \tag{3.5}$$

$$g = -\bar{u}^{[1\times 11]} \tag{3.6}$$

where we can omit the constant because it has no effect on the solution of 3.1.

Next we look at the constraints

$$-u_1 + u_{10} = -d_0 \tag{3.7}$$

$$u_i - u_{i+1} = 0, \quad i = 1, 2, ..8 \tag{3.8}$$

$$u_9 - u_{10} - u_{11} = 0 \tag{3.9}$$

giving

$$A = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ & & & \ddots & & \\ 0 & 0 & 0 & & 1 & 0 \\ 0 & 0 & 0 & \cdots & -1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & -1 \\ 0 & 0 & 0 & \cdots & 0 & -1 \end{bmatrix}^{[1\times 10]} \tag{3.10}$$

$$b = \begin{bmatrix} -d_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}^{[11\times 10]} \tag{3.11}$$

3.2 Exercise 2.2

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.

As in exercise 1 we also have an EQP so we can use the same first order conditions stated in proposition 2.10, [Jør21].

$$\nabla_x \mathcal{L}(x,\lambda) = \nabla_x f(x) - \sum_{i\in\varepsilon} \lambda_i \nabla_x c_i(x) = 0 \tag{3.12}$$

$$c_i(x) = 0 \tag{3.13}$$

First we state the Lagrangian for our specific problem.

$$\mathcal{L}(x,\lambda) = \frac{1}{2}x^T H x + g^T x - \sum_{i=1}^{10} \lambda_i c_i(x) \tag{3.14}$$

Next we find 3.12 and 3.13 for our problem.

$$\begin{aligned} \nabla_x \mathcal{L}(x,\lambda) &= Hx + g - A\lambda \\ &= x - \bar{u}^{[1\times 11]} - A\lambda \\ &= 0^{[1\times 11]} \end{aligned} \tag{3.15}$$

$$\begin{aligned} c_i(x) &= \nabla_\lambda \mathcal{L}(x,\lambda) \\ &= -A^T x + b \\ &= 0^{[1\times 10]} \end{aligned} \tag{3.16}$$

which as in exercise 1.2 can be written as a KKT system

$$\begin{bmatrix} I & -A \\ -A^T & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} \bar{u} \\ -b \end{bmatrix}$$

From proposition 2.10 we know the first order conditions are necessary. Furthermore it is stated in section 2.5, [Jør21], that if our optimization problem is convex the first order conditions are also sufficient. A QP is convex if the H matrix is positive semi definite. H is positive semi definite if all its eigenvalues are non-negative. Because H is the identity this is trivially fulfilled. Therefore we know 3.12 and 3.12 are both necessary and sufficient for this problem.

3.3 Exercise 2.3

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

```
function [H,g,A,b] = recycleSystem(n,ubar,d0)
b = zeros(n,1);
b(1) = -d0;

H = eye(n+1);

g = -ubar*ones(n+1,1);

A1 = -eye(n);
A2 = [zeros(n-1,1) eye(n-1)];
bot = zeros(1,n);
bot(1) = 1;
A = [A2;bot]+A1;
bot(1) = 0;
bot(n) = -1;
A = [A;bot];
end
```

Listing 3.1: The program constructs an EQP of the recycle system

3.4 Exercise 2.4

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

We stated the KKT matrix in exercise 2.2.

```
1 function [KKT, rhs] = kktMatrix(n,ubar,d0)
2 [H,g,A,b] = recycleSystem(n,ubar,d0);
3 KKT = [H -A; -A' zeros(n)];
4 rhs = [-g;-b];
5 end
```

Listing 3.2: The program constructs the KKT system of the recycle system

3.5 Exercise 2.5

Make a Matlab function that solves the recycle system using LU factorization

When we do LU-factorization of the KKT-system we get

$$\begin{aligned}\begin{bmatrix} I & -A \\ -A^T & 0 \end{bmatrix}\begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} -g \\ -b \end{bmatrix} \implies \\ (LU)\begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} -g \\ -b \end{bmatrix} \implies \\ L\left(U\begin{bmatrix} x \\ \lambda \end{bmatrix}\right) = \begin{bmatrix} -g \\ -b \end{bmatrix} \end{aligned} \tag{3.17}$$

We can now split the problem into two by introducing a dummy variable to explicitly see we have two linear systems.

$$U\begin{bmatrix} x \\ \lambda \end{bmatrix} = y \tag{3.18}$$

$$Ly = \begin{bmatrix} -g \\ -b \end{bmatrix} \tag{3.19}$$

We implement in matlab

```
function [x,lambda] = luSolver(KKT, rhs,n)
[L,U]  = lu(KKT);
sol = U\(L\rhs);
x = sol(1:n+1);
lambda = sol(n+2:2*n+1);
end
```

Listing 3.3: A LU-solver for an EQP

3.6 Exercise 2.6

Make a Matlab function that solves the recycle system using LDL factorization

As stated in exercise 1.3 and on page 455 in [NW06] the LDL-factorization is a numerical stable alternative to the LU factorization when ones matrix is symmetric but still indefinite. We implement by following page 455 in [NW06].

```
function [x,lambda]=ldlSolver(KKT, rhs,n)
[L, B, P] = ldl(KKT);
sol = L'\(B\(L\(P'*rhs)));
x = sol(1:n+1);
lambda = sol(n+2:2*n+1);
end
```

Listing 3.4: A LDL-solver for an EQP

3.7 Exercise 2.7

Make a Matlab function that solves the recycle system using the null space method.

From page 457 in [NW06] we know that the null space method works as long A is full rank and $Z^T HZ$ is positive definite, where z is the basis of the null space. The null space method is using the algorithmic idea constraint elimination presented on page 428-429 in [NW06], where we can transform our problem into an unconstrained problem.
From [Bro] we get that the null space method applies the constraint elimination idea the following way:

We QR-factorize A

$$A = QR = \begin{bmatrix} \hat{Q} & Q_{null} \end{bmatrix} \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}$$

meaning

$$\begin{aligned} A^T x = b \iff \\ R^T Q^T x = \hat{R}^T \hat{Q}^T x = b \end{aligned}$$

We can now write x as $x = \hat{Q} + Q_{null} v$ resulting in

$$\begin{aligned} \hat{R}^T \hat{Q}^T x &= \hat{R}^T \hat{Q}^T (\hat{Q} + Q_{null} v) \\ &= \hat{R}^T u = b \iff \\ u &= (\hat{R}^{-1})^T b \end{aligned}$$

We observe that u is fixed but v can vary freely

$$\begin{aligned} x &= \hat{Q} + Q_{null} v \\ &= \hat{Q} (\hat{R}^{-1})^T b + Q_{null} v \end{aligned}$$

Where we define $\hat{x} = \hat{Q}(\hat{R}^{-1})^T b$. We can now rewrite our objective function as

$$\begin{aligned} \phi(x) &= \frac{1}{2} x^T H x + g^T x \\ &= \frac{1}{2} (\hat{x} + Q_{null} v)^T H (\hat{x} + Q_{null} v) + g^T (\hat{x} + Q_{null} v) \\ &= \frac{1}{2} v^T (Q_{null}^T H Q_{null}) v + (\hat{x}^T H Q_{null} + g^T Q_{null}) v + \frac{1}{2} \hat{x}^T H \hat{x} + g^T \hat{x} \end{aligned}$$

This is an unconstrained problem in v so we can just solve for v.

$$v^* = -(Q_{null}^T H Q_{null})^{-1} (\hat{x} H Q_{null} + g^T Q_{null})$$

giving

$$x^* = \hat{x} + Q_{null} v^*$$

To obtain λ^* we can use equation 16.20 on page 457 in [NW06].

$$\begin{aligned} \lambda^* &= (\hat{Q}^T A)^{-1} \hat{Q}^T (g + H x^*) \\ &= \hat{R}^{-1} \hat{Q}^T (g + H x^*) \end{aligned}$$

We can now implement in matlab

```
function [x,lambda]=nullSpaceSolver(H,g,A,b)
[n,m] = size(A);
[Q,R] = qr(A);
Qhat = Q(:,1:m);
Qn = Q(:,m+1:n);
R = R(1:m,1:m);
Y = (R'\b);
Z = (Qn'*H*Qn)\(-Qn'*(H*Qhat*Y+g));
x = Qhat*Y+Qn*Z;
lambda = R\Qhat'*(g+H*x);
end
```

Listing 3.5: A null space solver for an EQP

3.8 Exercise 2.8

Make a Matlab function that solves the recycle system using the Range-Space method

The range space method, also called the Schur Complement method, assumes H is positive definite and is described on page 455-456 in [NW06].
We write the KKT system as two equations

$$Hx - A\lambda = -g \tag{3.20}$$

$$-A^T x = -b \implies$$

$$A^T x = b \tag{3.21}$$

We then multiply 3.20 with $A^T H^{-1}$

$$A^T x - A^T H^{-1} A\lambda = -A^T H^{-1} g \implies \tag{3.22}$$

$$A^T x = A^T H^{-1} A\lambda - A^T H^{-1} g \implies$$

$$x^* = H^{-1} A\lambda - H^{-1} g \tag{3.23}$$

We then subtract 3.22 form 3.21

$$A^T x - (A^T x - A^T H^{-1} A\lambda) = b + A^T H^{-1} g \implies$$

$$A^T H^{-1} A\lambda = b + A^T H^{-1} g \implies$$

$$\lambda^* = (A^T H^{-1} A)^{-1}(b + A^T H^{-1} g) \tag{3.24}$$

We can now implement the method in matlab.

```
1 function [x,lambda]=rangeSpaceSolver(H,g,A,b)
2 Hg = H\g;
3 HA = H\A;
4 lambda = (A'*HA)\(b+A'*Hg);
5 x = HA*lambda-Hg;
6 end
```

Listing 3.6: A range space solver for an EQP

CHAPTER 4

Inequality Constrained Quadratic Optimization

In example 16.4 on page 475 in [NW06] the following problem is stated

$$\begin{aligned} \min_x q(x) &= (x_1 - 1)^2 + (x_2 - 2.5)^2 \\ \text{subject to} \quad & x_1 - 2x_2 + 2 \geq 0, \\ & -x_1 - 2x_2 + 6 \geq 0, \\ & -x_1 + 2x_2 + 2 \geq 0, \\ & x_1 \geq 0, \\ & x_2 \geq 0. \end{aligned} \tag{4.1}$$

4.1 Exercise 3.1

Make a contour plot of the problem

We use matlab to plot 4.1 as a contour plot

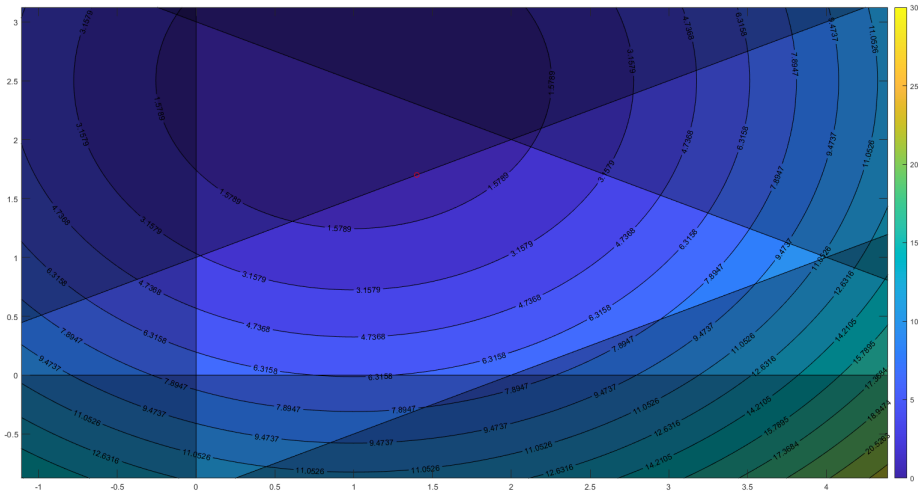


Figure 4.1: Contour of 4.1

4.2 Exercise 3.2

Write the KKT conditions up for 4.1

We need 4.1 in the form

$$\begin{array}{ll} \min_{x \in R^n} & \phi = \frac{1}{2} x^T H x + g^T x \\ \text{s.t.} & A^T x \leq b \end{array}$$

So we have

$$H = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \tag{4.2}$$

$$g = \begin{bmatrix} -2 \\ -5 \end{bmatrix} \tag{4.3}$$

$$A = \begin{bmatrix} 1 & -1 & -1 & 1 & 0 \\ -2 & -2 & 2 & 0 & 1 \end{bmatrix} \tag{4.4}$$

$$b = \begin{bmatrix} -2 \\ -6 \\ -2 \\ 0 \\ 0 \end{bmatrix} \tag{4.5}$$

We write up the Lagrangian

$$\mathcal{L}(x, \lambda) = \frac{1}{2} x^T H x + g^T x - \lambda^T (A^T x - b)$$

In all previous exercises we have only dealt with equality constrained problems and hence only needed (2.61a) and (2.61b) from propersition 2.10 [Jør21]. Now we have inequalities so we furhter need (2.61c) - (2.61e). (Here denoted (4.6) - (4.10))

$$\nabla_x \mathcal{L}(x, \lambda) = \nabla f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i \nabla c_i(x) = 0 \tag{4.6}$$

$$c_i(x) = 0, \quad i \in \mathcal{E} \tag{4.7}$$

$$c_i(x) \geq 0, \quad i \in \mathcal{I} \tag{4.8}$$

$$\lambda_i \geq 0, \quad i \in \mathcal{I} \tag{4.9}$$

$$c_i(x) = 0 \quad \vee \quad \lambda_i = 0, \quad i \in \mathcal{I} \tag{4.10}$$

One can now just input the relevant terms.

4.3 Exercise 3.3

Argue that the KKT conditions are both necessary and sufficient for 4.1

As in exercise 2.2 we know from proposition 2.10 in [Jør21] that the KKT conditions are necessary. Also like in exercise 2.2 we can use section 2.5 in [Jør21] to argue if H is positive semi definite the KKT conditions are also sufficient. We see in 4.2 that H is just the identity multiplied with 2 so trivially positive definite. We can therefore conclude that the KKT conditions for 4.1 are both necessary and sufficient.

4.4 Exercise 3.4

Make an EQP solver

4 different approaches have been explained in exercise 2.5 to 2.8 so we refer to these exercises.

4.5 Exercise 3.5

Apply a conceptual active set algorithm to 4.1. Use the iteration sequence in Figure 16.3 in [NW06]. Plot the iteration sequence in your contour plot. For each iteration you should list the working set, the solution, x and the lagrange multipliers, λ.

Our start point is $x_0 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ with initial working set $W_0 = \{3, 5\}$. Th primal active set method is stated on page 472 in [NW06] as algorithm 16.3.

Iteration 1 We find the step, p, to become optimal under the current working set $W_0 = \{3, 5\}$ and the corresponding λs by solving

$$\begin{pmatrix} H & -A \\ -A^T & 0 \end{pmatrix} \begin{pmatrix} p \\ \lambda \end{pmatrix} = - \begin{pmatrix} Hx_k + g \\ 0 \end{pmatrix} \quad A = [a_i]_{i \in W}$$

Where the above system of equations is a combination of (16.39) and (16.42) in [NW06].

We get $p_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ meaning that $x_0 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ is optimal under the current working set W_0. We inspect λ

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

This means that by dropping either c_3 or c_5 we can obtain an improvment of x with getting infeasible under all of our constraints. So we drop c_3 and proceed to iteration 2.

$$p_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \lambda_1 = \begin{bmatrix} 0 \\ 0 \\ -2 \\ 0 \\ -1 \end{bmatrix} \quad x_1 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \quad W_1 = \{5\}$$

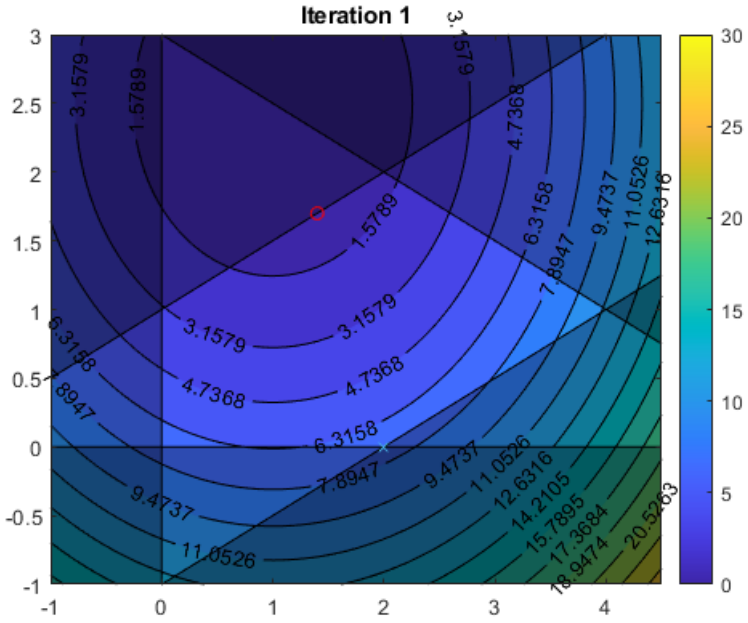


Figure 4.2: Iteration 1

Iteration 2 We again start by finding p and λ. We get $p_2 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$ meaning to get optimal under the current working set $W_1 = \{5\}$ we are to step p_2 from x_1. Before doing so we though have to check if we will remain feasible under all our constraints. To do so we will solve for the step length, α_k, under all constraints outside the current working set W_1.

$$\alpha_k \overset{\text{def}}{=} \min\left(1, \min_{i \notin \mathcal{W}_k, a_i^T p_k < 0} \frac{b_i - a_i^T x_k}{a_i^T p_k}\right)$$

We find $\alpha_2 = 1$ so $x_2 = x_1 + p_2$ is feasible under all our constraints. We therefore go to x_2 and continue to iteration 3.

$$p_2 = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \quad \lambda_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -5 \end{bmatrix} \quad x_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad W_2 = \{5\}$$

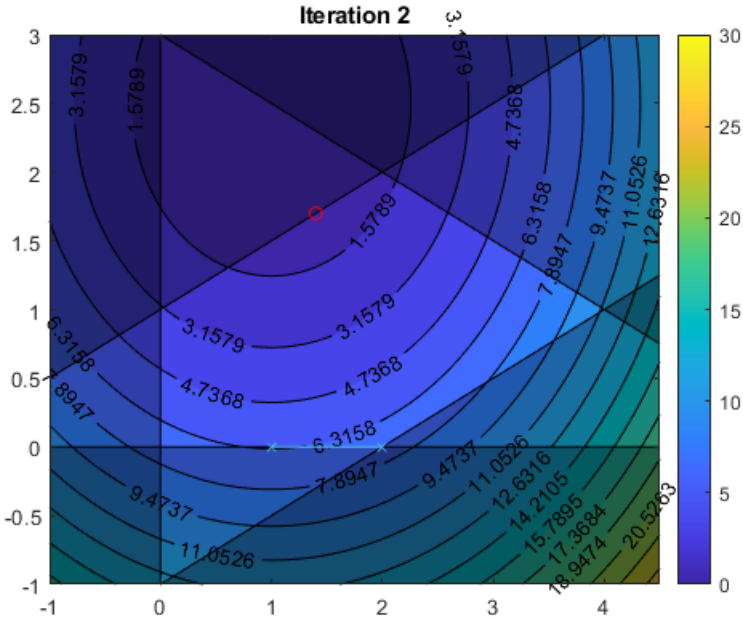


Figure 4.3: Iteration 2

Iteration 3 We calculate p_3 to be $\begin{bmatrix}0\\0\end{bmatrix}$ so x_2 is optimal under the current working set W_2. We inspect λ.

$$\lambda_3 = \begin{bmatrix}0\\0\\0\\0\\-5\end{bmatrix}$$

We drop c_5 because we observe from λ_3 that we can obtain an improvement in x without getting infeasible doing so.

$$p_3 = \begin{bmatrix}0\\0\end{bmatrix} \quad \lambda_3 = \begin{bmatrix}0\\0\\0\\0\\-5\end{bmatrix} \quad x_3 = \begin{bmatrix}1\\0\end{bmatrix} \quad W_3 = \{\}$$

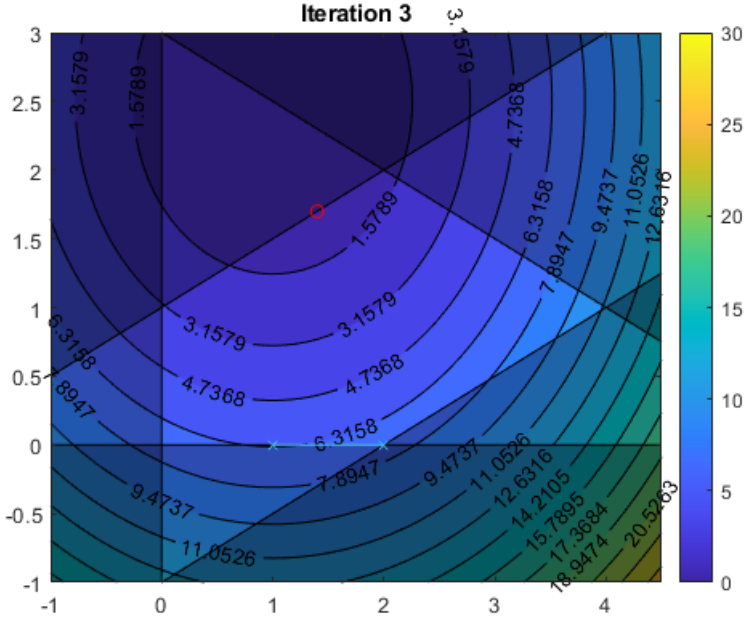


Figure 4.4: Iteration 3

Iteration 4 We get $p_4 = \begin{bmatrix} 0 \\ 2.5 \end{bmatrix}$ so we check the step length α. We get $\alpha_4 = 0.6$ meaning if we step further in the direction $x_3 + p_4$ we will get infeasible. By calculating

$$i_k = \underset{i \notin \mathcal{W}_k, a_i^T p_k < 0}{\arg\min} \frac{b_i - a_i^T x_k}{a_i^T p_k}$$

We get the constraint which is first violated in the direction $x_3 + p_4$ is c_1. We therefore add it to our working set, step to $x_4 = x_3 + \alpha_4 p_4$ and proceed to iteration 5.

$$p_4 = \begin{bmatrix} 0 \\ 2.5 \end{bmatrix} \quad \lambda_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_4 = \begin{bmatrix} 1 \\ 1.5 \end{bmatrix} \quad W_4 = \{1\}$$

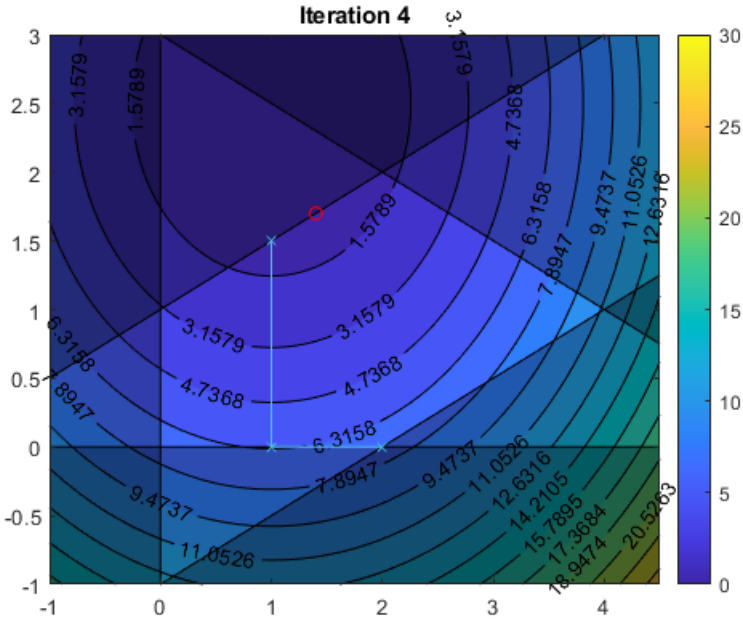


Figure 4.5: Iteration 4

Iteration 5 We get $p_5 = \begin{bmatrix} 0.4 \\ 0.2 \end{bmatrix}$ so we check the step length α. We get $\alpha_5 = 1$ so we step to $x_5 = x_4 + p_5$ and proceed.

$$p_5 = \begin{bmatrix} 0.4 \\ 0.2 \end{bmatrix} \quad \lambda_5 = \begin{bmatrix} 0.8 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_5 = \begin{bmatrix} 1.4 \\ 1.7 \end{bmatrix} \quad W_5 = \{1\}$$

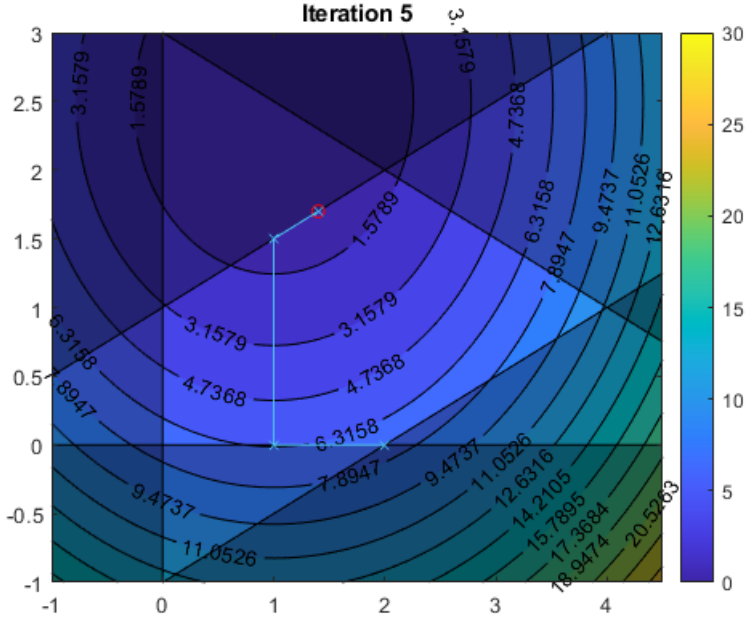


Figure 4.6: Iteration 5

Iteration 6 We calculate $p_6 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ so x_5 is optimal under the current working set W_5. We inspect λ.

$$\lambda_6 = \begin{bmatrix} 0.8 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

meaning we can not obtain any improvement in x by dropping a constraint without also getting infeasible. Therefore we are optimal under all our constraints in 4.1 and therefore return x_5 as the optimal solution and λ_6 as the Lagrange multipliers of the optimal solution.

$$p_6 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \lambda_6 = \begin{bmatrix} 0.8 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_6 = \begin{bmatrix} 1.4 \\ 1.7 \end{bmatrix} \quad W_6 = \{1\}$$

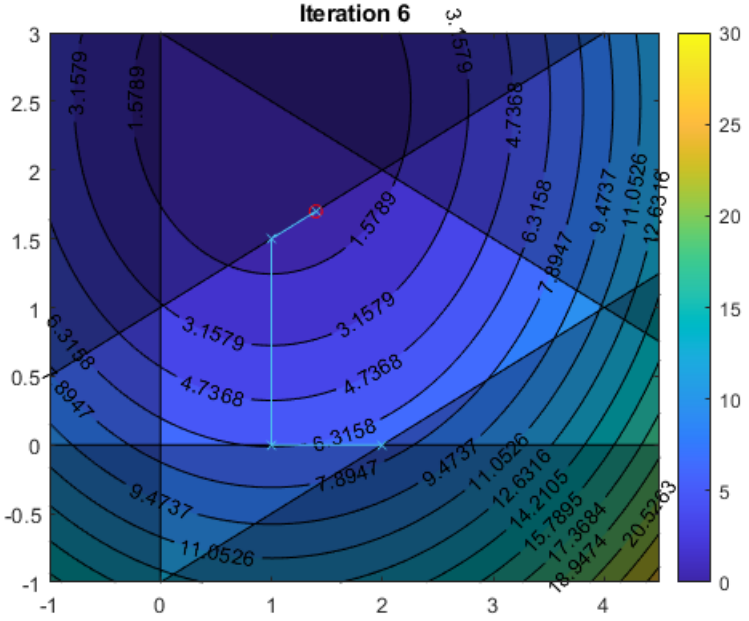


Figure 4.7: Iteration 6

4.6 Exercise 3.6

Comment on the Lagrange multipliers for each iteration.

Relevant comments regarding Lagrange multipliers for each iteration in exercise 3.5 can be found in exercise 3.5. Instead we will approach the Lagrange multipliers for the active set method a bit more generally here. Without loss of generality we consider the problem

$$\begin{aligned} \min_x f(x) &= x_1 + x_2 \\ \text{subject to} \quad -x_1 - x_2 &\geq 4, \\ x_1 &\geq 1, \\ x_2 &\geq 1. \end{aligned} \tag{4.11}$$

which can be illustrated as

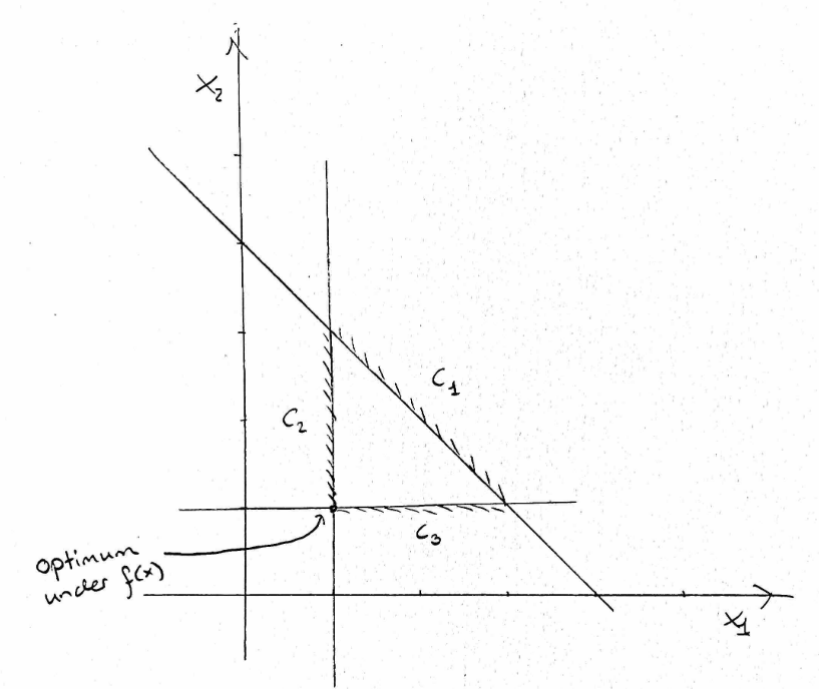


Figure 4.8: Illustration of 4.11

Furthermore we define our Lagrangian as

$$\mathcal{L}(x, \lambda) = f(x) - \sum_i \lambda_i c_i$$

We consider the optimum of $x = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ under the working set containing only c_1. We calculate the Lagrange multiplier

$$\nabla_x f(2,2) = \begin{bmatrix} 4 \\ 4 \end{bmatrix} \quad \nabla_x c_1(2,2) = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

giving

$$\begin{aligned} \nabla_x f(2,2) &= \nabla_x c_1(2,2) \Rightarrow \\ \begin{bmatrix} 4 \\ 4 \end{bmatrix} &= \lambda_1 \begin{bmatrix} -1 \\ -1 \end{bmatrix} \Rightarrow \\ \lambda_1 &= -4 \end{aligned}$$

Meaning we can drop c_1 and obtain an improvement in $f(x)$ without getting infeasible. If we try to redefine our Lagrangian to

$$\mathcal{L}(x, \lambda) = f(x) + \sum_i \lambda_i c_i$$

The sign of λ_1 would flip to positive. If we then flipped our inequality signs of our constraints

$$\begin{aligned} x_1 + x_2 &\leq -4, \\ -x_1 &\leq -1, \\ -x_2 &\leq -1. \end{aligned}$$

The sign would flip back to negative. So to determine if we are able to drop a constraint and obtain an improvement in $f(x)$ without getting infeasible depends on how we define our inequalities and Lagrangian. The relationship is summarized in the following table.

	$\leq$	$\geq$
$\mathcal{L} = f + \sum \lambda$	-	+
$\mathcal{L} = f - \sum \lambda$	+	-

Table 4.1: If we define our Lagrangian and inequalities as above we can drop a constraint and obtain an improvement in $f(x)$ without getting infeasible if the sign of our Lagrange multipliers are as above.

4.7 Exercise 3.7

Explain the active-set method for convex QPs listed on page 472 in [NW06]

The active-set method described on page 472 is defined with a negative Lagrange multiplier to check if we an drop a constraint to obtain an improvement. Therefore as explained in exercise 3.6 we need to have the following relationship

$$\begin{aligned} if \quad Ax \geq b \iff \mathcal{L}(x, \lambda) = f(x) - \lambda^T c \\ if \quad Ax \leq b \iff \mathcal{L}(x, \lambda) = f(x) + \lambda^T c \end{aligned}$$

If this is true we can apply the active-set method in the form on page 472 which goes as follows.

1. We start with a feasible point x_0 with a corresponding working set W_0

2. We calculate the EQP

$$\min_p \frac{1}{2} p^T H p + g_k^T p$$
$$\text{subject to } a_i^T p = 0, \quad i \in \mathcal{W}_k$$

which is the step p to go from x_{k-1} to the optimum under the current working set.

3. a) **If** $\|p\| = 0$ we calculate the Lagrange multipliers at x_{k-1} bu solving

$$\sum_{i \in W_{k-1}} a_i \lambda_i = H x_{k-1} + g$$

i. **If** all λ_i in the working set are positive we are optimal under all constraints and return x_{k-1} as the optimum

ii. **Else** we remove one of the constraints corresponding to a negative Lagrange multiplier to obtain an improvement in $f(x)$. **We return to 2**

b) **If** $\|p_k\| \neq 0$ we are not at a minimizer of our current working set. We therefore calculate the maximum step α, we can step without getting infeasible under all constraints. We calculate α by

$$\alpha_k = \min \left(1, \min_{i \notin \mathcal{W}_k, a_i^T p_k < 0} \frac{b_i - a_i^T x_k}{a_i^T p_k} \right)$$

i. **Else** then $x_k = x_{k-1} + \alpha_k p_k$ and we add the corresponding constrained to the working set. We calculate this by

$$i_k = \operatorname*{arg\,min}_{i \notin \mathcal{W}_k, a_i^T p_k < 0} \frac{b_i - a_i^T x_k}{a_i^T p_k}$$

We return to 2

ii. **Else** No constraints are blocking so $x_k = x_{k-1} + p_k$ and **return to 2**

4.8 Exercise 3.8

Use linprog to compute a feasible point to a QP. Apply and test this procedure to the problem in Example 16.4

When initializing the primal active-set method we need to solve a phase-1 problem as in the simplex method. This could be avoided by using the dual active-set method described in [HV07]. This works from the optimum of the unconstrained problem and towards the constrained optimum. This method also avoids cycling problems which the primal method is prone to. We solve the linear problem given by the constraints

```
A = [ 1 -2; -1 -2; -1 2; 1 0; 0 1/5];
b = [-2 -6 -2 0 0]';
linprog([0 0], A, b)

x =
    0.0000
    0.0000
```

We calculate

```
A*x-b

ans =
     2
     6
     2
     0
     0
```

So this corresponds to constraint 4 and 5 being in our working set. If we use this as initial point and working set we get the following solution path.

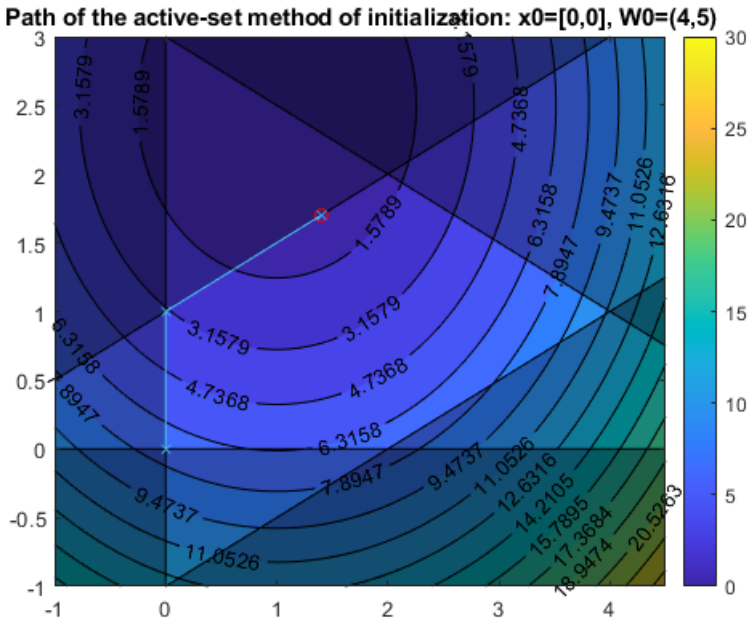


Figure 4.9: Solution path for $x_6 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $W_0 = \{4, 5\}$

4.9 Exercise 3.9

Implement the algorithm on page 472 and test it for the problem in Example 16.4. Print information for every iteartion of the algorithm and list it in the report.

We implement the algorithm

```
1  function ...
       [x, lambda, working_set_save, x_save, p_save, lambda_save, step_save] = ...
       activeSetMethod(H, g, A, b, x0, W0, max_iter, tol)
2  if nargin<7
3    max_iter = 100;
4  end
5  if nargin<8
6    tol = 0.00001;
7  end
8
```

```
addpath('C:/Users/anton/OneDrive - Københavns Universitet/Uni/Uni/8. ...
    semester/Constrained optimization/Exercises/Week5');

[n,m] = size(A);
working_set_save = zeros(m,max_iter+1);
x_save = zeros(n,max_iter+1);
lambda_save = zeros(m,max_iter+1);
p_save = zeros(n,max_iter+1);
step_save = zeros(1,max_iter+1);

lambdas = zeros(m,1);
working_set = zeros(m,1);
working_set(W0) = 1;
x = x0;
x_save(:,1) = x0;
working_set_save(:,1) = working_set;

for i=1:max_iter
   Wk = find(working_set == 1);
   %Solve direction
   [p,lambda] = rangeSpaceSolver(H,(H*x+g),A(:,Wk),zeros(length(Wk),1));

   p_save(:,i) = p;
   lambda_save(Wk,i) = lambda;
   if norm(p)<tol
       if ~any(lambda<0)
           lambdas(Wk) = lambda;
           lambda = lambdas;
           x_save(:,i+1) = x;
           working_set_save(:,i+1) = working_set;

           x_save = x_save(:,1:i+1);
           p_save = p_save(:,1:i);
           lambda_save = lambda_save(:,1:i);
           step_save = step_save(:,1:i);
           working_set_save = working_set_save(:,1:i+1);
           return
       else
           j = lambda==min(lambda(lambda<0));
           j = Wk(j);
           working_set(j) = 0;

           x_save(:,i+1) = x;
           working_set_save(:,i+1) = working_set;
       end
   else
       notWk = find(working_set == 0);
       denom = (A(:,notWk)'*p);
       dummy = ...
           (b(notWk(denom<-tol))-A(:,notWk(denom<-tol))'*x)./denom(denom<-tol);
       alpha = min(1,min(dummy));

       f = (find(min(dummy)==dummy));
       k=find((denom<0));
       j = k(f(1));
```

```
62
63          step_save(:,i) = alpha;
64
65          if alpha ≠ 1
66              x = x+alpha*p;
67              working_set(j) = 1;
68              x_save(:,i+1) = x;
69              working_set_save(:,i+1) = working_set;
70          else
71              x = x+p;
72              x_save(:,i+1) = x;
73              working_set_save(:,i+1) = working_set;
74          end
75      end
76  end
77  end
```

Listing 4.1: A matlab implementation of the algorithm on page 472 in [NW06]

We apply it to the initial conditions given in example 16.4 in [NW06].

```
1  >> x0 = [2;0];
2  >> W0 = [3  5];
3  >> [x,lambda,working_set_save,x_save] = activeSetMethod(H,g,A,b,x0,W0)
4
5  x =
6       1.4000
7       1.7000
8
9  lambda =
10      0.8000
11           0
12           0
13           0
14           0
15
16 working_set_save =
17      0      0      0      0      1      1      1
18      0      0      0      0      0      0      0
19      1      0      0      0      0      0      0
20      0      0      0      0      0      0      0
21      1      1      1      0      0      0      0
22
23 x_save =
24    2.0000    2.0000    1.0000    1.0000    1.0000    1.4000    1.4000
25         0         0         0         0    1.5000    1.7000    1.7000
```

We see our algorithm returns the correct numbers.

4.10 Exercise 3.10

Test your active-set algorithm for the problem in Example 16.4 using (16.47) and (16.48) in [NW06]. Do the same with quadprog.

CHAPTER 5

Markowitz Portfolio Optimization

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 with- out reducing return. Identify cation and construction of such portfolios is called hedging. The Markowitz Portfolio 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

5.1 Exercise 4.1

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

From example 16.1 in [NW06] we have

$$R = \sum_{i=1}^{n} x_i r_i, \quad \forall x_i \leq 0$$

$$\mathrm{E}[R] = \mathrm{E}[\sum_{i=1}^{n} x_i r_i] = \sum_{i=1}^{n} x_i \,\mathrm{E}[r_i] = x^T \mu$$

$$\mathrm{Var}[R] = \mathrm{E}[R - \mathrm{E}[R]^2] = x^T \Sigma x$$

where H is a $n \times n$ symmetric matrix defined by

$$\Sigma_{ij} = \rho_{ij}\sigma_i\sigma_j$$

i.e. the covariance matrix. Ideally we want to find a portfolio where $x^T\mu$ is large and $x^T\Sigma x$ is small, i.e. high return and low risk. We can combine these two into an optimization problem by

$$\max_x x^T\mu - \kappa x^T G x, \quad \text{subject to } \sum_{i=1}^{n} x_i = 1, x \geq 0 \tag{5.1}$$

where κ is a measure of risk tolerance. Or equivalently as a minimization problem.

$$\min_x \kappa x^T G x - x^T\mu, \quad \text{subject to } \sum_{i=1}^{n} x_i = 1, x \geq 0 \tag{5.2}$$

5.2 Exercise 4.2

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

5.3 Exercise 4.3

Use quadprog to find a portfolio with return, R=10 minimal risk. What is the optimal portfolio and what is the risk (variance)?

We set up the corresponding optimization problem

$$\min_x \; x^T G x \tag{5.3}$$

$$\text{s.t. } \sum_{i=1}^{5} x_i = 1, \tag{5.4}$$

$$x \geq 0, \tag{5.5}$$

$$\sum_{i=1}^{5} x_i\mu_i = 10 \tag{5.6}$$

giving

$$H = \Sigma \qquad g = []$$
$$A_{eq} = \begin{bmatrix} 1^{[1\times 5]} \\ \mu \end{bmatrix} \quad b_{eq} = \begin{bmatrix} 1 \\ 10 \end{bmatrix}$$

We know from the documentation of quadprog that the inequalities should be given as $Ax \geq b$. So we flip the inequality sign on 5.5 by multiplying with -1. This gives

$$A_{ineq} = \left[1^{[1\times 5]}\right] \quad b_{ineq} = \left[0\right]$$

We calculate the optimization problem in matlab

```
>> x = quadprog( H, f, Aineq, bineq, Aeq, beq)
 x =
      0.0000
      0.2816
      0.0000
      0.7184
      0.0000

>>port_risk = x'*covariance*x
 port_risk =
      2.0923
```

5.4 Exercise 4.4

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.

We can calculate the same problem as in exercise 4.3 with R taking values in the interval $[min(\mu), max(\mu)]$. We have chosen to equidistant steps of size 0.01 giving 866 optimization problems. The result can be plotted as the efficient frontier as seen below

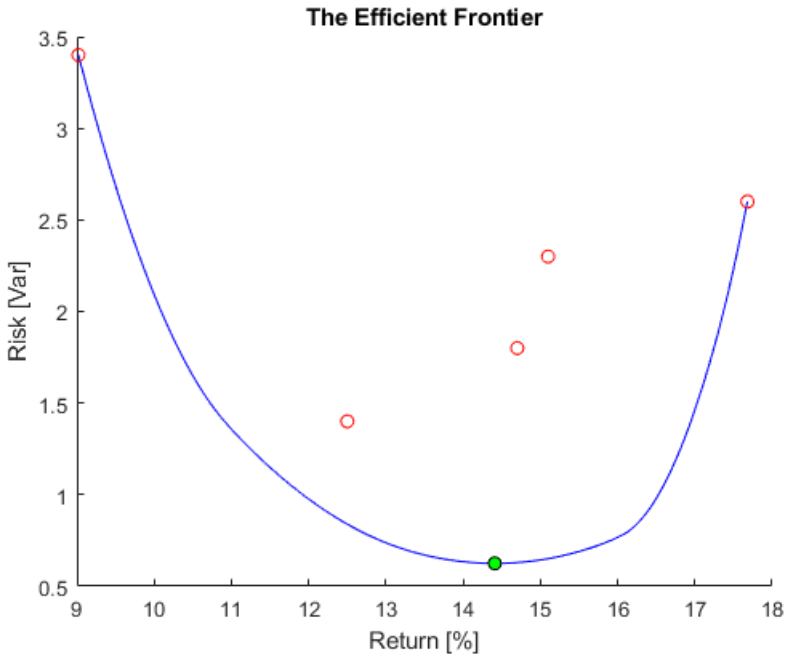


Figure 5.1: The efficient frontier

We clearly see the effect of hedging. When choosing assets so they correlate such that if one go down others go up we can obtain higher return with lower risk than more obvious combinations. We find the optimal portfolio to be

$$\begin{bmatrix} 0.0880 \\ 0.2512 \\ 0.2823 \\ 0.1040 \\ 0.2745 \end{bmatrix}$$

with a return of 14.41 and a risk(variance) of 0.6249

5.5 Exercise 4.5

In the following we add a risk free security to the financial market. It has return $r_f = 2.0$. What is the new covariance matrix and return vector?

Our assets are now

Security	Covariance						Return
1	2.30	0.93	0.62	0.74	-0.23	0	15.10
2	0.93	1.40	0.22	0.56	0.26	0	12.50
3	0.62	0.22	1.80	0.78	-0.27	0	14.70
4	0.74	0.56	0.78	3.40	-0.56	0	9.02
5	-0.23	0.26	-0.27	-0.56	2.60	0	17.68
6	0	0	0	0	0	0	2.0

5.6 Exercise 4.6

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

We apply the same technique as in exercise 4.4

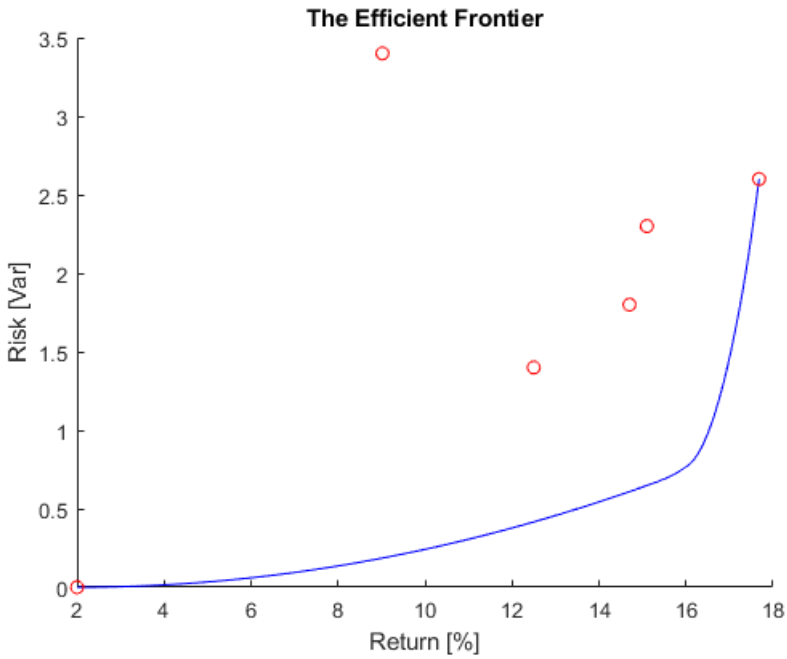


Figure 5.2: The efficient frontier with a risk free asset

We see when having a risk free asset and only optimize on risk we will of course only buy the risk free asset. If we though plot our old efficient frontier with the new

and then zoom in on our previous optimal solution.

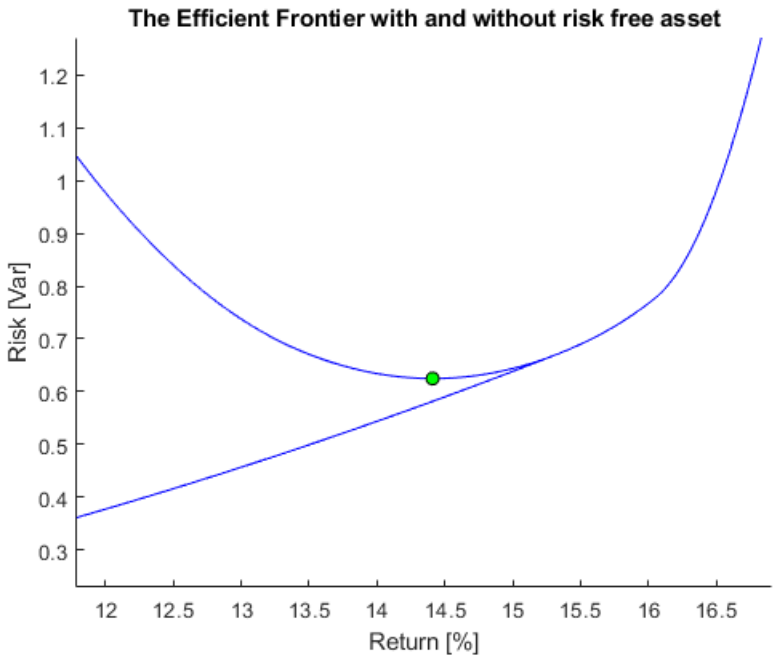


Figure 5.3: The efficient frontier with a risk free asset and with out

We see that we can the same return with a bit lower risk or with the same risk obtain a little higher return.

5.7 Exercise 4.7

What is the minimal risk and optimal portfolio giving a return of R=15. Plot this point in your optimal portfolio as function of return as well as on the efficient frontier diagram.

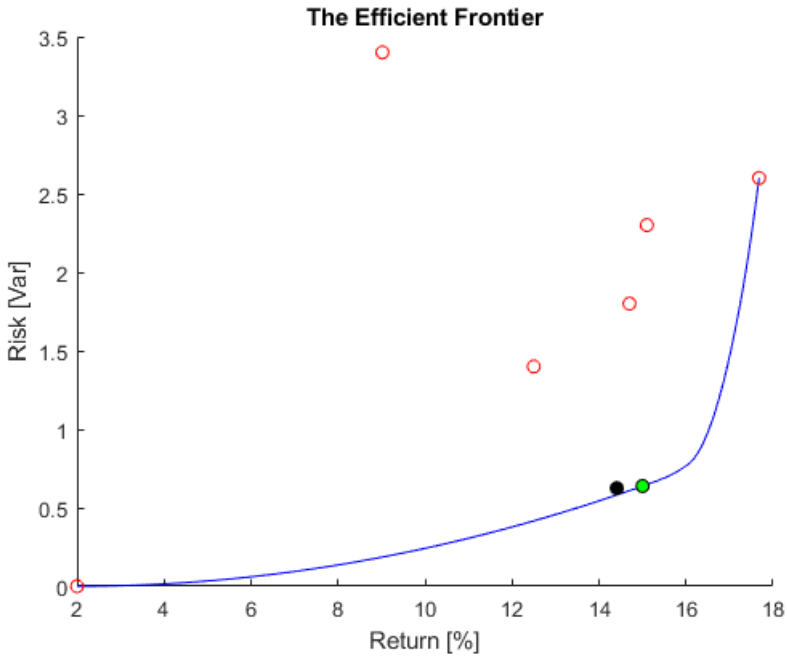


Figure 5.4: The efficient frontier with a risk free asset and with out

As pointed out in exercise we see that our new optima which is indicated with the green dot has almost the same risk as the previous(black dot) but a higher return. Our new optimal portfolio is

$$\begin{bmatrix} 0.1655 \\ 0.1365 \\ 0.3115 \\ 0.0266 \\ 0.3352 \\ 0.0247 \end{bmatrix}$$

with a risk(variance) of 0.6383. So we get an increase of 0.59 in return for a increase of 0.0134 in risk.

CHAPTER 6

Inequality Constrained Quadratic Programming

Consider the QP in Example 16.4 (p.475) in Nocedal and Wright. This is an example of a convex QP in the form

$$\begin{array}{ll} \min_{x \in R^n} & f(x) = \frac{1}{2} x^T H x + g^T x \\ \text{s.t.} & c_i(x) = a_i^T x + b_i \geq 0 \quad i \in \mathcal{I} \end{array} \tag{6.1}$$

6.1 Exercise 5.1

Make a contour plot of the problem.

From exercise 3.1 we copy the contour plot.

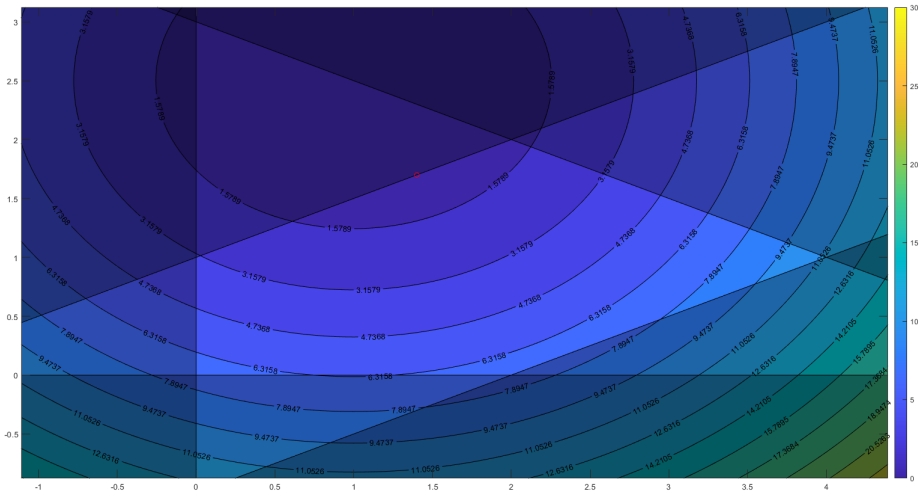


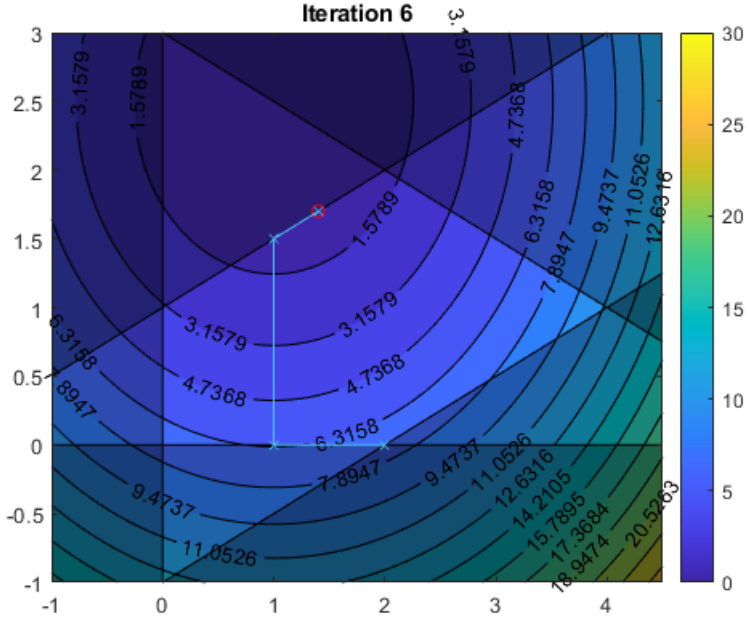
Figure 6.1: Contour of 6.1

6.2 Exercise 5.2

Implement a primal active-set QP algorithm and illustrate its iterations in the contour plot for a feasible starting point.

We have implemented the primal active-set method in exercise 3 so further explanation of the method can be read there. We will just plot the path of the algorithm starting at $x_0 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$ and with initial active set $W_0 = \{3, 5\}$.

$$p_6 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \lambda_6 = \begin{bmatrix} 0.8 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_6 = \begin{bmatrix} 1.4 \\ 1.7 \end{bmatrix} \quad W_6 = \{1\}$$



The method converge after 6 iterations and return

$$\lambda_6 = \begin{bmatrix} 0.8 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_6 = \begin{bmatrix} 1.4 \\ 1.7 \end{bmatrix} \quad W_6 = \{1\}$$

6.3 Exercise 5.3

Implement a dual active-set QP algorithm and illustrate its iterations in the contour plot (for any starting point)

Before just implementing the dual active-set method from a pseudo-code we will try to understand why it works. To do this we will derive the method from our primal problem formulation. The derivation is based on [HV07]. Our primal problem is stated in the form

$$\begin{array}{ll} \min_{x \in R^n} & f(x) = \frac{1}{2} x^T H x + g^T x \\ \text{s.t.} & c_i(x) = a_i^T x + b_i \geq 0 \quad i \in \mathcal{I} \end{array} \tag{6.2}$$

where H must be strictly convex for the dual method. The dual problem is then given as

$$\begin{aligned} \max_{x,\lambda} \quad & \mathcal{L}(x,\lambda) = \frac{1}{2}x^T Hx + g^T x - \lambda^T (A^T x - b) \\ \text{s.t.} \quad & \nabla_x \mathcal{L}(x,\lambda) = Hx + g - A\lambda = 0 \end{aligned}$$

As seen in exercise 1.9 we know from section 2.7 in [Jør21] that the KKT conditions are both necessary and sufficient for both the primal and the dual because we have restricted H to be strictly convex and all the c_is are affine. We now introduce a primal working set, W, and a dual working set, W_D. The primal working set comply with

$$\begin{aligned} Hx^{(k)} + g - \sum_{i \in W} a_i \lambda_i^{(k)} &= 0 \\ c_i(x^{(k)}) = a_i^T x^{(k)} - b_i &= 0, \quad i \in W \\ \lambda_i^{(k)} &\geq 0, \quad i \in W. \end{aligned}$$

And the dual set follows

$$\begin{aligned} Hx^{(k)} + g - \sum_{i \in W_D} a_i \lambda_i^{(k)} &= 0 \\ \lambda_i^{(k)} &= 0, \quad i \in W_D. \end{aligned}$$

We can now from 6.2, 6.3 and the KKT conditions infer that x is optimal when

$$c_i(x^{(k)}) \geq 0, \forall i \in W_D$$

We further observe that when ever there exists a constraint, c_r, not satisfying this in the dual set we are not optimal. This observation we will use to develop a iterative approach.

6.3.1 Improving direction and step length

We know that if some constraint c_r is negative we can increase the corresponding λ_r which will improve our dual problem 6.3 without causing it getting infeasible. This can formally be written as

$$\begin{aligned} \bar{x} &= x + s \\ \bar{\lambda}_i &= \lambda_i + u_i, \quad i \in W \\ \bar{\lambda}_r &= \lambda_r + t \\ \bar{\lambda}_i &= 0 \quad i \in W_D \backslash \{r\} \end{aligned} \tag{6.4}$$

Using the KKT conditions we can formulate this similarly to the KKT systems we have seen earlier.

$$\begin{bmatrix} H & -A \\ -A^T & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} + \begin{bmatrix} g \\ b \end{bmatrix} - \begin{bmatrix} a_r \\ 0 \end{bmatrix} \lambda_r + \\ \begin{bmatrix} H & -A \\ -A^T & 0 \end{bmatrix} \begin{bmatrix} s \\ u \end{bmatrix} - \begin{bmatrix} a_r \\ 0 \end{bmatrix} t = 0. \tag{6.5}$$

Using the KKT conditions again we also know that the first line of 6.5 must equal 0 so we can reduce it to

$$\begin{bmatrix} H & -A \\ -A^T & 0 \end{bmatrix} \begin{bmatrix} s \\ u \end{bmatrix} - \begin{bmatrix} a_r \\ 0 \end{bmatrix} t = 0. \tag{6.6}$$

We now 6.6 to

$$\begin{bmatrix} H & -A \\ -A^T & 0 \end{bmatrix} \begin{bmatrix} p \\ v \end{bmatrix} = \begin{bmatrix} a_r \\ 0 \end{bmatrix}, \quad \text{where} \quad \begin{bmatrix} s \\ u \end{bmatrix} = \begin{bmatrix} p \\ v \end{bmatrix} t$$

We have now the primal space direction, p, the dual space direction v and the step length t. We can solve the above system for the directions but the step length t we need to do a little more work to obtain a method which we can use to calculate it. We rewrite 6.4.

$$\begin{aligned} \bar{x} &= x + s = x + pt \\ \bar{\lambda}_i &= \lambda_i + u_i = \lambda_i + v_i t, \quad i \in W \\ \bar{\lambda}_r &= \lambda_r + t \\ \bar{\lambda}_i &= 0 \quad i \in W_D \backslash \{r\} \end{aligned} \tag{6.7}$$

We know all λ's must be non-negative for 6.3 to be feasible. Therefore we infer that it must hold for condition 2 in 6.7 that

$$t \in [0, t_{max}], \text{ where } \quad t_{max} = min(\infty, min_{i:v_i<0} \tfrac{\lambda_i}{v_i}),\ i \in W_D$$

We now have a method to derive a maximal step length in the dual space and we know from [HV07] that all step sizes $t > 0$ increases c_r so we do not have to worry about the primal problem. We will therefore now derive the optimal step length only using 6.3. We know from [HV07] that

$$\mathcal{L}(\bar{x}, \bar{\lambda}) = \mathcal{L}(x, \lambda) - \frac{1}{2} t^2 a_r^T p - t c_r(x) \Rightarrow$$
$$\mathcal{L}(\bar{x}, \bar{\lambda}) - \mathcal{L}(x, \lambda) = -\frac{1}{2} t^2 a_r^T p - t c_r(x)$$

We set the r.h.s. equal to zero and differentiate w.r.t. t to maximize the increase in $\mathcal{L}(\bar{x}, \bar{\lambda})$.

$$\frac{d\mathcal{L}}{dt} = -ta_r^T p - c_r(x) \Rightarrow t^* = \frac{-c_r(x)}{a_r^T p}$$

We now know that t must equal

$$t = min(t_{max}, t^*)$$

for both maximizing 6.3 and remaining feasible.

Lastly before implementing the algorithm we must make sure $A = [a_i]_{i \in W}$ atains full rank after adding c_r. From [HV07] we know that when

$$a_r^T p = 0$$

Then c_r and $A = [a_i]_{i \in W}$ are linearly dependent. To solve this we remove c_j from W where j equals

$$\arg\min_{j: v_j < 0} \frac{-\lambda_j}{v_j}$$

If we are not optimal and no $v_j < 0$ exists the problem is infeasible.

6.3.2 Implementation

The dual active-set method has two advantages over the primal formulation. First we do not have issues regarding cycling and secondly we do not need to solve a phase-1 problem to obtain an initial starting point and working set. We can just solve the unconstrained problem

$$\begin{aligned} x_0 &= -H^{-1}g \\ W_0 &= \{\} \\ W_D 0 &= \{c_i\}, \quad \forall i \in \mathcal{I} \end{aligned}$$

If one wants to see a pseudo-code formulation of the algorithm we recommend this pseudo-code formulation. We will just state the Matlab implementation in the appendix and show the contour plot here.

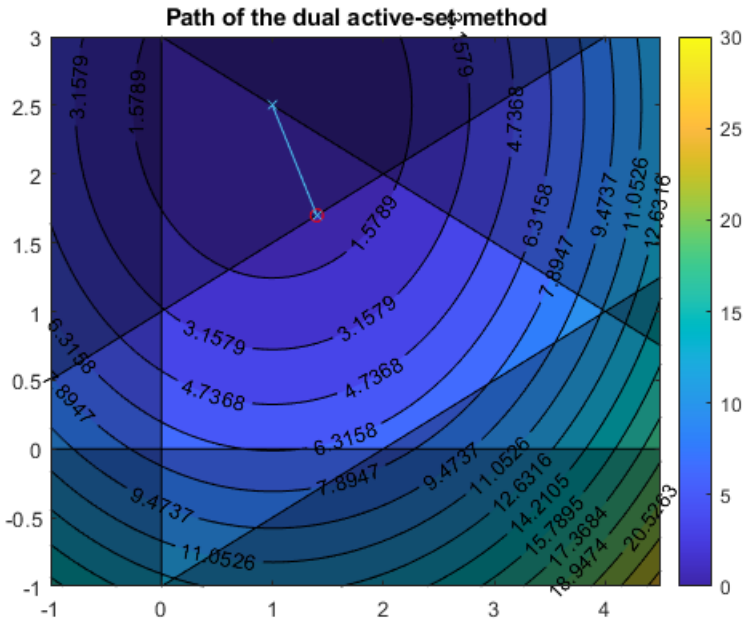


Figure 6.2: Contour of 6.1 solved by the dual active-set method

6.4 Exercise 5.4

Implement a primal-dual interior point QP algorithm and illustrate its iterations in the contour plot (for any starting point)

In the 6.1 we only have inequality constraints but we will derive the algorithm containing both equality and inequality. The problem is therefore of the form

$$\begin{array}{ll} \min_{x \in R^n} & \phi = \frac{1}{2} x^T H x + g^T x \\ \text{s.t.} & A^T x = b \\ & C^T x \geq d \end{array} \tag{6.8}$$

The following derivation is based on lecture 6 and especially the slide show "QuadraticOptimization". For the interior point method we do not want general form inequalities as $C^T x \geq d$ but only inequalities of the form $x \geq 0$. We therefore introduce slack variables to rewrite 6.8. We define

$$s := C^T x - d \geq 0$$

so

$$\begin{aligned} -C^T x + s + d &= 0 \\ s &\geq 0 \end{aligned}$$

We now have 6.8 in a new form

$$\begin{array}{ll} \min_{x \in R^n} & \phi = \frac{1}{2} x^T H x + g^T x \\ \text{s.t.} & A^T x = b \\ & C^T x + d + s = 0 \\ & s \geq 0 \end{array} \tag{6.9}$$

We write up the Lagrangian for 6.9

$$\mathcal{L}(x, \lambda, \mu) = \frac{1}{2} x^T H x + g^T x - \lambda^T (A^T x - b) - \mu^T (C^T x - d)$$

And the corresponding KKT conditions

$$\begin{aligned} \nabla_x \mathcal{L}(x, \lambda, \mu) &= Hx + g - A\lambda - C\mu = 0 \\ \nabla_\lambda \mathcal{L}(x, \lambda, \mu) &= -\left(A^T x - b\right) = 0 \\ \nabla_\mu \mathcal{L}(x, \lambda, \mu)) &= -\left(C^T x - d\right) \leq 0 \\ \mu &\geq 0 \\ \left(C^T x - d\right)_i \mu_i &= 0, \qquad i = 1, 2, \ldots, m_c \end{aligned}$$

We introduce a bit of notation to simplify further calculations down the road.

$$S = \begin{bmatrix} s_1 & & \\ & \ddots & \\ & & s_{m_c} \end{bmatrix} \quad M = \begin{bmatrix} \mu_1 & & \\ & \ddots & \\ & & \mu_{m_c} \end{bmatrix} \quad e = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

Which allows us to rewrite

$$\begin{aligned} s_i \mu_i = 0, \quad i = 1 \ldots m_c \Rightarrow \\ SMe = 0 \end{aligned}$$

We can then write the KKT conditions as

$$\begin{aligned} F(x, \lambda, \mu, s) = \begin{bmatrix} Hx + g - A\lambda - C\mu \\ -A^T x + b \\ -C^T x + s + d \\ SMe \end{bmatrix} = \begin{bmatrix} r_L \\ r_A \\ r_C \\ r_{SM} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ s \geq 0 \\ \mu \geq 0 \end{aligned}$$

We can now apply Newtons method for which we need the Jacobian of F.

$$J_F = \begin{bmatrix} H & -A & C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & M \end{bmatrix}$$

Which gives the Newton step

$$J_F \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta \mu \\ \Delta s \end{bmatrix} = - \begin{bmatrix} r_L \\ r_A \\ r_C \\ r_{SM} \end{bmatrix}$$

When having calculated the direction $\begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta \mu \\ \Delta s \end{bmatrix}$ we need to make sure μ and s stays positive. To ensure this we introduce a step length α.

$$\begin{bmatrix} x \\ \lambda \\ \mu \\ s \end{bmatrix}_{k+1} = \begin{bmatrix} x \\ \lambda \\ \mu \\ s \end{bmatrix}_k + \alpha \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta \mu \\ \Delta s \end{bmatrix}, \quad s.t. \;\; (\mu_{k+1}, s_{k+1}) \geq 0$$

When solving this problem one thing to have in mind is staying inside the interior of our feasible region. If we get out to the boundary of the region we would take very small steps and hence converge very slowly. To formalize this idea we introduce a concept called the central path.

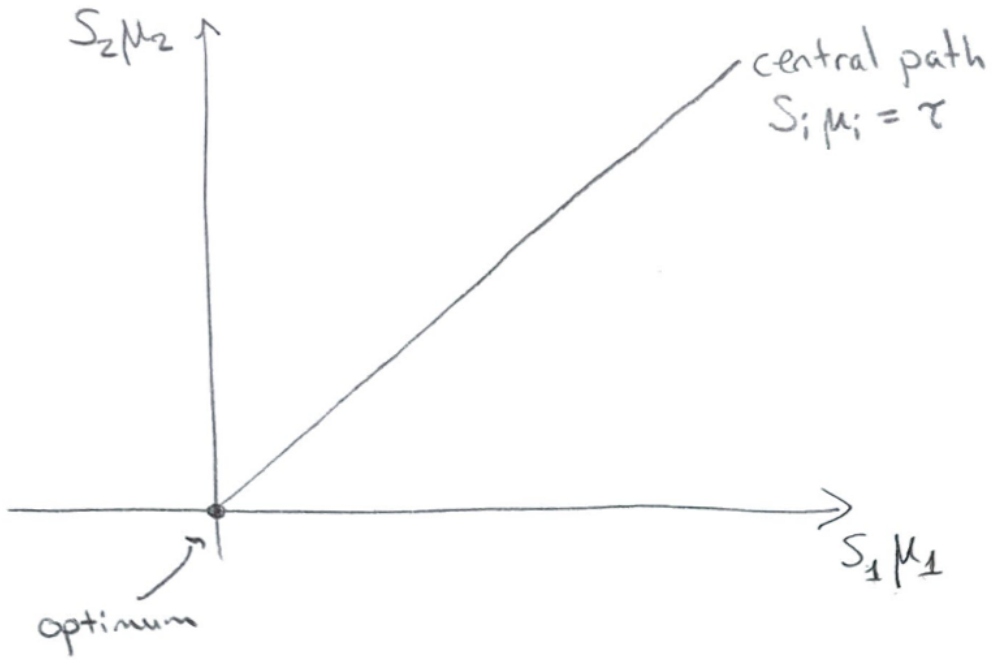


Figure 6.3: Illustration of the central path

We parameterize the central path as

$$C = \{(x_\tau, \lambda_\tau, \mu_\tau, s_\tau) : \tau > 0\}$$

which gives rise to a new parameterized newton step.

$$\begin{bmatrix} H & -A & C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & M \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\mu \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_L \\ -r_A \\ -r_C \\ -r_{SM} + \tau e \end{bmatrix}$$

To calculate τ we introduce a measure of the duality gap for our current solution and the optimum.

$$\delta = \frac{s^T \mu}{m_c}$$

So when $\delta = 0$ it means that $\left(C^T x - d\right)_i \mu_i = s_i \mu_i = 0$ for all our constraints as the KKT conditions says it should be at the optimum. We calculate τ as

$$\tau = \sigma\delta, \quad \sigma \in [0, 1]$$

Where σ is a centering parameter. If $\sigma = 0$ we call it an affine step and if $\sigma = 1$ we call is a centering step.

To choose σ in an adaptive manner we introduce Mehrotra's modifications:

1. We do an affine step, i.e. a step with $\sigma = 0$
2. We then calculate the duality gap of our affine step as

$$\delta^{aff} = \frac{(\mu + \alpha^{aff}\Delta\mu^{aff})^T(s + \alpha^{aff}\Delta s^{aff})}{m_c}$$

   We can then calculate σ for a centering step as

$$\sigma = \left(\frac{\delta^{aff}}{\delta}\right)^3$$

   Where δ is the duality gap of the previous iteration. We see that the centering step will be very different from the affine step if the affine step did not produce a large reduction to the duality gap.

3. If we were to take a full step in the affine direction our complementary condition induced by our central path would not be satisfied as it can be seen here

$$\mu_i\Delta s_i + s_i\Delta\mu_i = \sigma\delta - \mu s_i - \Delta\mu_i\Delta s_i$$

   where the last term is not present in the current formulation. So we add it as a corrector step.

These three steps can be solved in two systems. First we calculate the affine step.

$$\begin{bmatrix} H & -A & C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & M \end{bmatrix} \begin{bmatrix} \Delta x^{aff} \\ \Delta\lambda^{aff} \\ \Delta\mu^{aff} \\ \Delta s^{aff} \end{bmatrix} = \begin{bmatrix} -r_L \\ -r_A \\ -r_C \\ -r_{SM} \end{bmatrix}$$

When factorizing the jacobian matrix J_F we save it and use it for the second step where we combine the corrector and centering step to.

$$\begin{bmatrix} H & -A & C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & M \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\mu \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_L \\ -r_A \\ -r_C \\ -r_{SM} - \Delta\mu_i\Delta s_i + \sigma\delta e \end{bmatrix}$$

The complexity of a matrix factorization grows cubicly with the size of the matrix while the back substitution grown quadratically so if we could reduce the size of our jacobian matrix we would gain a lot computational wise. On slide 24 and 25 from

the slide show "QuadraticOptimization" the reductions of the jacobian to get the augmented equations are stated. We will not go through them here but just state the reduced system.

$$\begin{bmatrix} H + C(S^{-1}M)C^T & -A \\ -A^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -r_L + C(S^{-1}M)(r_C - M^{-1}\bar{R}_{SM}) \\ -r_A \end{bmatrix}$$

where $\bar{r}_{SM} = -S\Delta\mu - M\Delta s$. We can the compute $\Delta\mu$ and Δs by

$$\begin{aligned} \Delta\mu &= -\left(S^{-1}M\right)C^T\Delta x + \left(S^{-1}M\right)\left(r_C - M^{-1}r_{SM}\right) \\ \Delta s &= -M^{-1}r_{SM} - M^{-1}S\Delta\mu \end{aligned}$$

We can now compute the step size α where we regularize it with a constant η. So our final newton step becomes

$$\begin{bmatrix} x \\ \lambda \\ \mu \\ s \end{bmatrix}_{k+1} = \begin{bmatrix} x \\ \lambda \\ \mu \\ s \end{bmatrix}_{k} + \eta\alpha \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\mu \\ \Delta s \end{bmatrix}, \quad s.t. \ \ (\mu_{k+1}, s_{k+1}) \geq 0$$

Lastly different stopping criteria are discussed on slide 20-22 in the slide show "QuadraticOptimization". We will not elaborate further on this here but just use

$$\delta \geq \varepsilon 10^{-2}\delta^0$$

as stopping criteria.

6.4.1 Implementation

We will now summarize the above derivation.

1. Compute
$$\bar{H} = H + C\left(S^{-1}M\right)C^T$$
2. Compute
$$\bar{r}_L = r_L - C\left(S^{-1}M\right)\left(r_C - M^{-1}r_{SM}\right)$$
3. Solve
$$\begin{bmatrix} \bar{H} & -A \\ -A^T & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} \bar{r}_L \\ \bar{r}_A \end{bmatrix}$$
4. Compute
$$\Delta\mu = -\left(S^{-1}M\right)C^T\Delta x + \left(S^{-1}M\right)\left(r_C - M^{-1}r_{SM}\right)$$

5. Compute

$$\Delta s = -M^{-1} r_{SM} - M^{-1} S \Delta \mu$$

6. Update solution

$$\begin{bmatrix} x \\ \lambda \\ \mu \\ s \end{bmatrix}_{k+1} = \begin{bmatrix} x \\ \lambda \\ \mu \\ s \end{bmatrix}_{k} + \eta\alpha \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta \mu \\ \Delta s \end{bmatrix}, \quad s.t. \quad (\mu_{k+1}, s_{k+1}) \geq 0$$

7. Check convergence and return to 1 if not satisfied.

We will not state the matlab implementation here but refer to appendix. One extra thing the matlab implementation does which is not mentioned in the above is some initial point heuristics. It is mentioned on slide 29 of the slide show "QuadraticOptimization".

Below we have plotted the solution path for 6.1 where we used the interior algorithm with $x0 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$ where the implementation corrects for a initial point which would create a slack variable equal to 0.

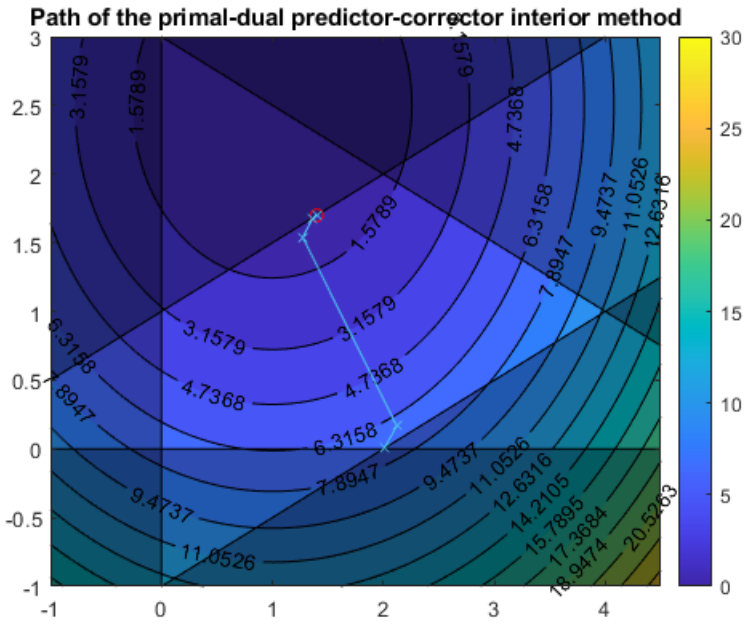


Figure 6.4: Contour of 6.1 solved by the primal-dual predictor-corrector interior method

CHAPTER 7

Linear Programming

In the following we will go through two methods for solving a linear program(LP). The first is the revised simplex which is an active set method and the second is Mehrota's predictor-corrector primal-dual method which is a interior point method. We will start deriving the revised simplex method.

7.1 Revised Simplex

The following derivation is based on lecture 11 and 13 in the course [Bro]. The derivation will start with a standard form LP.

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

Now assume that A is full rank and the rows of A are linearly independent.

$$A \in R^{m \times n} \quad rank(A) = m \quad m \leq n$$

Assume further that the variables have been written in such an order that the last m columns of A are linearly independent. We can then partition A and x as follows

$$\begin{aligned} A &= \begin{bmatrix} A_N & A_B \end{bmatrix}, \quad A_B \in R^{m \times m}, \ A_N \in R^{m \times (n-m)} \\ x &= \begin{bmatrix} x_N \\ x_B \end{bmatrix}, \quad x_B \in R^m, \ x_N \in R^{n-m} \end{aligned} \tag{7.1}$$

The constraints $Ax = b$ have now become

$$\begin{aligned} \begin{bmatrix} A_N & A_B \end{bmatrix} \begin{bmatrix} x_N \\ x_B \end{bmatrix} &= b \Rightarrow \\ A_N x_N + A_B x_B &= b \end{aligned} \tag{7.2}$$

If we set $x_N = 0$ then 7.2 becomes

$$\begin{aligned} A_B x_B &= b \Rightarrow \\ x_B &= A_B^{-1} b \end{aligned} \tag{7.3}$$

The resulting $x = \begin{bmatrix} x_N \\ x_B \end{bmatrix}$ has at least $n - m$ zero elements and is called a basic solution.
x_N are the non-basic variables and x_B are the basic variables.

From the Fundamental theorem of linear programming we know that if A is assumed full rank then

1. If a feasible solution exists then a basic feasible solution(BFS) exists
2. If an optimal feasible solution exists then a optimal BFS exists

So this theorem says we only need to consider basic feasible solutions when searching for an optimum, i.e. we are looking for an optimal basis B.
The simplex method then says: Once we have a basic feasible solution we try to improve it by replacing one basic variable with a non-basic variable. We continue until no improvement to $c^T x$ is possible.
Assume we have a BFS $x^{(k)}$ with sets B and N of basic and non-basic variables. We compute

$$x_B^{(k)} = A_B^{-1} b$$

and the cost is

$$\begin{aligned} c^T x^{(k)} &= c_N^T x_N^{(k)} + c_B x_B^{(k)} \\ &= c_B^T x_B^{(k)} \end{aligned}$$

We want to see what happens to the cost if we start changing the non-basic variables from zero. If we change x_N, we need to recompute x_B from 7.2

$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N$$

This is possible because A_B is invertible by design. The cost can then be expressed in terms of x_N as

$$\begin{aligned} c^T x^{(k)} &= c_B^T x_B^{(k)} + c_N^T x_N^{(k)} \\ &= c_B^T A_B^{-1} b + (c_N^T x_N^{(k)} - c_B^T A_B^{-1} A_N) x_N^{(k)} \\ &= c^T x^{(k)} + r^T x_N^{(k)} \end{aligned}$$

where $r = c_N - A_N^T A_B^{-T} c_B$. This vector r is known as the vector of reduced cost or the Lagrange multipliers. r quantifies how much each non-basic variable will affect the cost if increased, i.e. if entry 1 of r is positive $x_N[1]$ will increase the cost and vice versa.
Therefore if $r \succeq 0$ then no improvement is possible which means we are optimal.

Otherwise we pick the most negative entry of r and increase the corresponding non-basic variable, x_e, called the entering variable.
We now need to find which variable in the basic set which have to leave. To find this we will start with

$$x_B = A_B^{-1}b - A_B^{-1}A_N x_N$$

Which becomes

$$\begin{aligned} x_B &= A_B^{-1}b - A_B^{-1}a_e x_e \\ &= \hat{b} + dx_e \end{aligned}$$

where a_e is the column of A corresponding to x_e.
The variable x_i, $i \in B$, becomes zero when

$$x_e = \frac{\hat{b}_i}{-d_i}$$

We are interested in the first basic variable, x_l, to reach zero

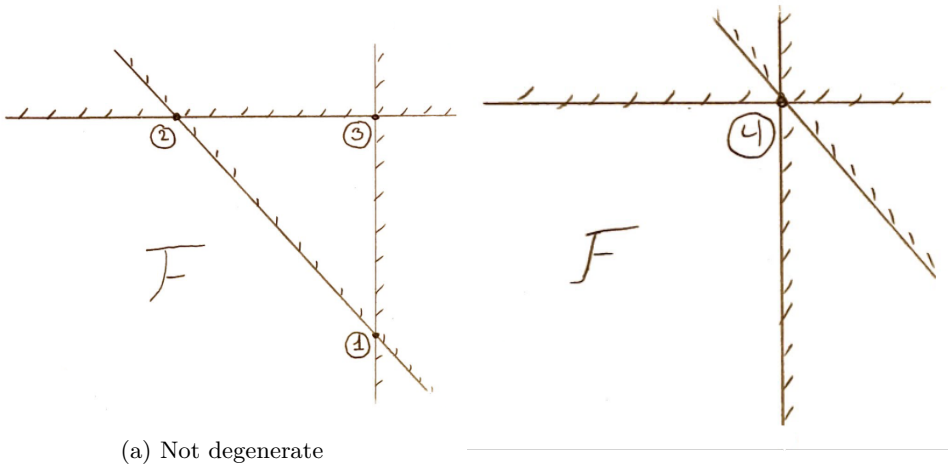
$$\frac{\hat{b}_l}{-d_l} = \min_{i \in B : d_i < 0} \frac{\hat{b}_i}{-d_i}$$

If no $d_i < 0$ then we know x_e can be increased indefinitely which means the LP is unbounded. We now switch l and e in the sets N and B. We then update A_N and A_B and proceed to the next iteration. It can be shown that our new A_B is of full rank so invertibility is not a problem.

It may be that one $\hat{b}_i = 0$ and $d_i < 0$. In this case x_e can not be increased, so the objective is not decreased. We can still switch the entering and the leaving variable to get a new basis, but our objective remain the same. In this case it is possible to return to the same basis set and begin cycling. This phenomenon is called degeneracy which active set-methods some times are vulnerable to as we also saw for the primal active-set method for IQPs.

7.1.1 Degeneracy

A BFS is called degenerate if $x_i = 0$ for at least on i in the basis. Geometrically degeneracy is when two basic solutions overlap.



(a) Not degenerate

(b) Degenerate

Figure 7.1: We see that the vertical constraint has been shifted to the left from figure (a) to figure (b) which has made (b) degenerate.

We see that the three basic solutions in 7.1a have merged into one in 7.1b. This phenomenon can as mentioned cause cycling which is when a sequence of bases $B_1 \dots B_k$ occurs repeatedly. This requires that

1. Each basis $B_1 \dots B_k$ is degenerate as the entering variable is 0
2. No x_e is altered so all BFSs are the same point as in 7.1b

An example of a LP which will cause the plain vanilla simplex to cycle is Beal's example given in section 4 of this paper.

A solution to the cycling problem is Bland's anti-cycling rules

1. Of all non-basic variables for which the corresponding entry of r is negative, choose the one with the smallest index to be the entering variable.
2. If there is a tie for the leaving variable, pick the one with the smallest index.

These rules only needs to be invoked if no decrease in $c^T x$ is obtained.

7.1.2 Implementation

The starting assumption of the revised simplex algorithm is that we have a BFS. This is called the phase-1 problem and is almost as much work as solving the LP it self. We will not dive further into the topic here but only recommend this source where

three different methods are explained.

A pseudo-code for the revised simplex algorithm is stated below.

Revised Simplex

1. Initialization: We have sets B_0, N_0 and corresponding A_B and $x_B = A_B^{-1}b$
2. Iterate until $r \succeq 0$
   a) Compute reduced costs: $r = c_N - A_N^T A_B^{-T} c_B$
   b) We find the index e of r which is the most negative. This x_e from the set N is our entering variable.
   c) We now compute the leaving variable

$$\hat{b} = A_B^{-1}b, \quad d = -A_B^{-1}a_e$$

$$l = \underset{i \in B: d_i < 0}{\arg\min} \frac{\hat{b}_i}{-d_i}$$

   where x_l from the set B is our leaving variable.
   d) If any of the ratios $\frac{\hat{b}_i}{-d_i}$ are 0 we restart the iterate with Bland's rules invoked.
   e) We update N and B

We see that we in the above algorithm is calculating A_B^{-1} so when implementing the algorithm we should calculate the factorization every iterate and then reuse it for all three inverse. An even faster implementation would make use of something as a rank-1 update of the inverse every iteration because we are only changing one row per iteration. We will not implement this in our implementation but if one wants to know more about this topic we recommend this source.

7.2 Mehrota's predictor-corrector method

We will now turn to Mehrota's predictor-corrector method. The method is a primal-dual interior point method proposed by Sinjay Mehrota in 1989.

Instead of having a static central path neighbourhood the method adjust the centering step based on how much the predictor step reduces a late to be defined duality measure.

The following derivation is based on chapter 14 in [NW06].

7.2.1 Derivation

We have a linear program in standard form

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{array} \tag{7.4}$$

where $c \in R^n$, $A \in R^{m \times n}$, $b \in R^m$ and $x \in R^n$. The KKT-conditions for 7.4 are as follows.

$$\begin{aligned} c - A^T\lambda - s &= 0 \\ Ax - b &= 0 \\ XSe &= 0 \\ x, s &\geq 0 \end{aligned}$$

where $X = diag(x)$, $S = diag(s)$ and $e = \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix} \in R^n$.
We can rewrite the KKT-conditons as

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

We now want to take a newton step towards the optimum using our KKT-conditions. This full newton step is called the predictor step or the affine scaling direction.

$$J_F(x, \lambda, s) \begin{bmatrix} \Delta x^{aff} \\ \Delta\lambda^{aff} \\ \Delta s^{aff} \end{bmatrix} = -F(x, \lambda, s) \tag{7.5}$$

Where J_F is the Jacobian of F given by

$$J_F = \begin{bmatrix} 0 & -A^T & -I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix}$$

Nest we define the centering step which is exactly the same as 7.6 exept the last index of F. Here we include a duality measure

$$\delta = \frac{x^T s}{n}$$

which is weighted by a centering parameter $\sigma \in [0, 1]$

$$J_F(x, \lambda, s) \begin{bmatrix} \Delta x^{cen} \\ \Delta \lambda^{cen} \\ \Delta s^{cen} \end{bmatrix} = - \begin{bmatrix} c - A^T \lambda - s \\ Ax - b \\ XSe - \sigma \delta e \end{bmatrix} \tag{7.6}$$

If $\sigma = 1$ we will not improve our objective but redirect back to the central path. If $\sigma = 0$ we will take a full predictor step improving our objective as much as possible in that iteration.
Lastly we have the corrector step. To understand this step we will write up the last equation of 7.6.

$$s_{i_i}^{aff} + x_i \Delta s_i^{aff} = -x_i s_i \tag{7.7}$$

The complementary slackness should be

$$(x_i + \Delta x_i^{aff})(s_i + \Delta s_i^{aff}) = 0 \tag{7.8}$$

We now subtrract 7.7 from 7.8 and get

$$\Delta x_i^{aff} \Delta s_i^{aff} = 0$$

So we add the term $\Delta x_i^{aff} \Delta s_i^{aff}$ giving the aggregated system

$$J_F(x, \lambda, s) \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta s \end{bmatrix} = - \begin{bmatrix} c - A^T \lambda - s \\ Ax - b \\ XSe - \sigma \delta e + \Delta X^{aff} \Delta S^{aff} e \end{bmatrix} \tag{7.9}$$

We now only need to define how to select the centering parameter σ adaptively. Mehrota came up with the formula

$$\sigma = \left(\frac{\delta^{aff}}{\delta} \right)^3$$

where

$$\begin{aligned} \delta^{aff} &= (x + \alpha_{aff}^{Pri} \Delta^{aff})^T (s + {}_{aff}^{Dual} \Delta s^{aff}) \\ \alpha_{aff}^{Pri} &= \min(1, \min_{i:\Delta x_i^{aff} < 0} - \frac{x_i}{\Delta x_i^{aff}}) \\ \alpha_{aff}^{Dual} &= \min(1, \min_{i:\Delta s_i^{aff} < 0} - \frac{s_i}{\Delta s_i^{aff}}) \end{aligned}$$

So if the predictor step reduces the duality gap a lot compared to the previous duality gap, the centering parameter will be close to 0 and vice versa.

Our final step is then

$$x^{k+1} = x^k + \alpha_k^{Pri} \Delta x^k$$
$$\lambda^{k+1} = \lambda^k + \alpha_k^{Dual} \Delta \lambda^k$$
$$s^{k+1} = s^k + \alpha_k^{Dual} \Delta s^k$$

where

$$\alpha_k^{Pri} = \min \left(1, \eta(\min_{i:\Delta x_i^k < 0} -\frac{x_i}{\Delta x_i^k}) \right)$$
$$\alpha_k^{Dual} = \min \left(1, \eta(\min_{i:\Delta s_i^k < 0} -\frac{s_i}{\Delta s_i^k}) \right)$$

and η we often choose to 0.95.

7.2.2 Implementation

When implementing Mehrota's predictor-corrector method we are interested in factorizing the smallest possible matrix because computations grow cubicly with the size of the matrix. On slide 21 from week 7 the normal equations are presented.

!!! BESKRIV HVORDAN NORMAL EQUATIONS REDUCERER PROGRAMMET TIL EN STØRRELSE SÅ VI KAN BENYTTE CHOLESKY FAC OG SLIPPER FOR KÆMPE MATRICER

APPENDIX A

An Appendix

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bibliography

[3bl] 3blue1brown. *Implicit differentiation, what's going on here? | Essence of calculus, chapter 6*. https://www.youtube.com/watch?v=qb40J4N1fa4.

[Bro] Dr Richard Brown. *UC math 352, University of Canterbury*. https://www.youtube.com/playlist?list=PLh464gFUoJWOmBYla3zbZbc4nv2AXez6X.

[HV07] Esben Lundsager Hansen and Carsten Völcker. "Numerical Algorithms for Sequential Quadratic Optimization." Technical University of Denmark, 2007.

[Jør21] John Bagterp Jørgensen. *Numerical Methods for Constrained Optimization*. first. Springer, 2021.

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. second. New York, NY, USA: Springer, 2006.

{2.7182818284}ϑφερτυθιοπσδφγηξκλ