Chapter 4. Optimization

4.1 Introduction

Optimization problems are ubiquitous in science and engineering, and even in our daily life, thinking about how we optimize our way to go to work, the choice of line we stand in at the supermarket, or deciding for the education for our children.

We begin this chapter with several simple examples, which show the breadth of problems that fall into the category of optimization problems.

4.1.1 How much daily exercise is optimal ?

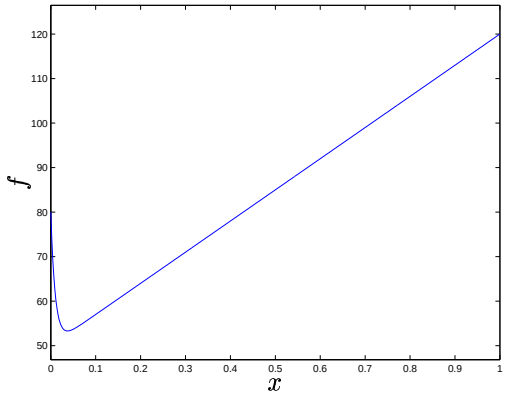
In his John von Neumann lecture at the annual SIAM meeting, Joe Keller asked this question and proposed a very simple model to answer it. Suppose at birth, every human being is given a fixed number of heart beats, and once these heartbeats are used up, life ends. How should one optimally use these heart beats to have as long a life as possible ? A first immediate idea is to stay in bed and rest, so the heart beat stays low, and one uses the heart beats as economically as possible. Another idea comes however from the fact that a well trained heart is beating much more slowly when the person is at rest, than the heart of an untrained person. So exercise could be beneficial to increase the lifespan. Unfortunately during exercise, the heart beats fast, so that one uses up heart beats faster, in the hope to gain them back during rest. So is there an optimum ?

Suppose that the untrained heart beats 80 times a minute when a person is at rest, and that during exercise, it beats 120 times per minute. If a person exercises the fraction x of its time, then this person uses on average

$$f(x) := 120x + g(x)(1-x)$$

heartbeats per minute, where the unknown function g should be close to 80 for x small, which means the person does hardly do any exercise, and probably around 50 for x approaching 1, which means that the person is extremely well trained. Since it is known that a little exercise every day decreases the heart beat at rest already considerably, a simple model for g would be exponential decay, i.e.

$$g(x) := 50 + 30e^{-100x},$$



where the choice -100 is quite arbitrary here, and should be much more carefully researched with the help of a medical doctor. Figure 4.1.1 shows the function f for this example, and there is clearly a minimum, which means there is an optimum choice of x, which minimizes the average use of heartbeats, and thus leads to the longest life possible. From calculus, we know that we need to set the derivative to zero to find the minimum, which with Maple is easily achieved by

```
f:=120*x+(50+30*exp(-100*x))*(1-x);
fp:=diff(f,x);
solve(fp,x);
```

$$-\frac{1}{100}\,\mathrm{LambertW}(\frac{7}{3}\,e^{101})+\frac{101}{100}$$

It is interesting to see that the closed form solution of this problem involves the LambertW function we have already encountered in Chapter **??** on the numerical solution of non-linear equations. To obtain a numerical value in Maple, we type

```
evalf(%);
```

and Maple returns 0.0373019079, which means that one should exercise a bit over 50 minutes every day. If there had been no closed form solution in Maple, one could have used any of the nonlinear equation solvers from Chapter **??** to find the solution, or the maple command `fsolve`.

Is it also possible to find the minimum directly without knowing the derivative ? For this one dimensional problem, an algorithm like bisection from Section **??** would be nice, but it is not possible to tell from the midpoint of an initial interval $[a, b]$ if the minimum lies in the left or right half of the interval. If one however computes the function value for two distinct points x_1 and x_2 in $[a, b]$, $x_1 < x_2$, and if $f(x_1)$ is smaller than $f(x_2)$, as in the example in Figure 4.1, then a minimum must lie in the interval $[a, x_2]$. On the other hand, if $f(x_1)$ is bigger than $f(x_2)$, then a minimum must lie in the interval $[x_1, b]$, and hence

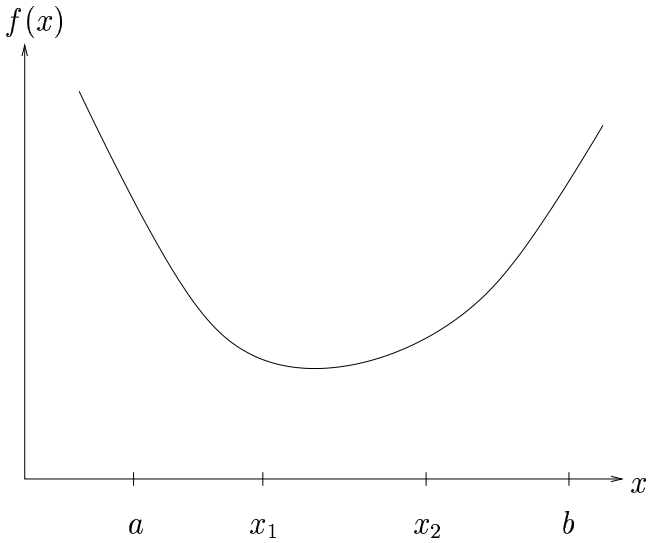


FIGURE 4.1.
Example of a minimum search algorithm similar to bisection, using two points x_1 and x_2 in the search interval $[a, b]$.

one can continue to search for the minimum in a smaller interval, as in bisection. A simple choice for x_1 and x_2 is to choose one third and two thirds of the interval $[a, b]$, but this choice has the big disadvantage that one needs to evaluate f twice in each step of the algorithm, since in the new interval, neither x_1 in the former, nor x_2 in the latter case lies at one third or two thirds of the new interval. A better choice would be to choose

$$x_1 = \lambda a + (1 - \lambda)b, \quad x_2 = (1 - \lambda)a + \lambda b, \tag{4.1}$$

and to determine λ such that already evaluated function values can be reused. This would mean that in the former case, the old x_1 would need to become the new x_2 in the interval $[a, x_2]$, i.e.

$$x_1 = (1 - \lambda)a + \lambda x_2.$$

Substituting the values from equation (4.1) and solving for λ yields

$$\lambda = \frac{-1 + \sqrt{5}}{2} \approx 0.6180, \tag{4.2}$$

where we have chosen the positive of the two roots. In the latter case, the reuse would imply

$$x_2 = \lambda x_1 + (1 - \lambda)b,$$

which leads after substitution from equation (4.1) to the same value of λ. This choice of λ corresponds to the golden section, and leads to the following simple algorithm in Matlab:

```
function x=Minimize(f,a,b)
```

```
% MINIMIZE finds a minimum of a scalar function
%   x=Minimize(f,a,b) searches for a minimum of the scalar function f
%   in the interval [a,b]

fa=feval(f,a); fb=feval(f,b);
l=(-1+sqrt(5))/2;
x1=l*a+(1-l)*b; x2=(1-l)*a+l*b;
fx1=feval(f,x1); fx2=feval(f,x2);
while a<x1 & x1<x2 & x2<b
  if fx1>fx2
    a=x1;
    x1=x2; fx1=fx2;
    x2=(1-l)*a+l*b; fx2=feval(f,x2);
  else
    b=x2;
    x2=x1; fx2=fx1;
    x1=l*a+(1-l)*b; fx1=feval(f,x1);
  end;
end;
x=x1;
```

4.1.2 Portable Phone Networks

Portable phone systems can be used almost everywhere, because there is a dense net of base stations, to which the weak portable phone can connect. Each base station covers its neighborhood and when you move with your portable phone from the neighborhood of one station into the neighborhood of a different one while calling, your call is automatically transferred from the old station to the new one. The dense covering of areas with base stations causes problems: if two base stations use the same frequency for a concurrent connection with corresponding portable phones, the signals may interfere and the quality of the communication decreases. This is the case in particular if the two base stations are physically close to each other. Therefore one would like to assign different frequencies to different base stations. However, in practice, there are less frequencies available than base stations and thus frequencies have to be reused. Naturally the problem arises how to assign frequencies to base stations so that the interference among simultaneous calls is minimized. This problem is known as the frequency assignment problem.

We derive a mathematical formulation of the frequency assignment problem we consider n base stations S_i, $i = 1, 2, \ldots, n$. We assume first that each station S_i sends with the same frequency at a power level p_i to describe how the power level affects interference. We look at pairs of base stations S_i and S_j as given in Figure 4.2. We define the entries of the link gain matrix $\tilde{G} = [\tilde{g}_{ij}]$ as follows:

- $\tilde{g}_{ii}p_i$ is the minimum signal power the phone connected with station S_i (sending with power p_i) receives anywhere in the area of S_i.

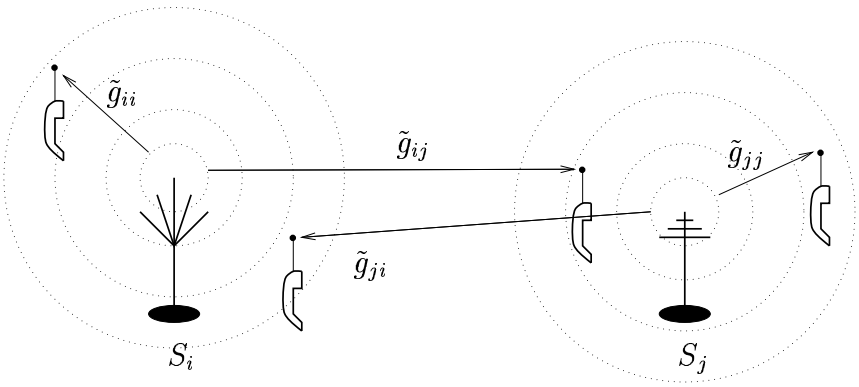


FIGURE 4.2.
Two base stations and positions of worst reception and interference

- $\tilde{g}_{ij}p_j$ is the maximum interference power the phone connected with station S_i receives anywhere in the area of S_i from the sender S_j (sending with power p_j).

In the following we neglect natural noise from other sources. The worst possible signal to noise ratio q_i of a phone connected with station S_i can now be defined as the ratio between the minimum signal power from the station S_i and the sum of the interference powers $\tilde{g}_{ij}p_j$ of the other stations S_j, $j \neq i$,

$$q_i := \frac{\tilde{g}_{ii}p_i}{\sum_{j\neq i}\tilde{g}_{ij}p_j} = \frac{p_i}{\sum_{j\neq i}\frac{\tilde{g}_{ij}}{\tilde{g}_{ii}}p_j} = \frac{p_i}{\left(\sum_j \frac{\tilde{g}_{ij}}{\tilde{g}_{ii}}p_j\right) - p_i}. \tag{4.3}$$

Defining the normalized link gain matrix $G = [g_{ij}]$ by

$$g_{ij} := \begin{cases} \frac{\tilde{g}_{ij}}{\tilde{g}_{ii}} & i \neq j, \\ 0 & \text{otherwise}, \end{cases} \tag{4.4}$$

the worst signal to noise ratio from (4.3) becomes

$$q_i = \frac{p_i}{\sum_j g_{ij}p_j}. \tag{4.5}$$

In the simple example with one frequency we have only the sending power p_i at each station S_i to maximize the smallest signal to noise ratio q_i over all stations. The optimum is obtained when the signal to noise ratio is the same for all stations, as one can see as follows: starting with all the q_i equal and decreasing the sending power p_j of station S_j we necessarily decrease the signal to noise ratio q_j since $g_{jj} = 0$. On the other hand increasing the power p_j of station S_j would necessarily decrease the signal to noise ratio of coupled neighboring stations, because $g_{jk} \geq 0$.

Since the matrix G is non negative, we have by the Perron Frobenius Theorem 3.3 that the largest eigenvalue λ_{max} of G is real and the corresponding

eigenvector $\boldsymbol{\phi}_{max}$ is non negative. Using this eigenvector as our power distribution, $\boldsymbol{p} := \boldsymbol{\phi}_{max}$, we obtain for the signal to noise ratio

$$q_i = \frac{p_i}{\sum_j g_{ij} p_j} = \frac{p_i}{\lambda_{max} p_i} = \frac{1}{\lambda_{max}}, \tag{4.6}$$

and thus all the stations can guarantee at least a signal to noise ratio $\frac{1}{\lambda_{\max}}$, the best possible solution with one frequency. If one has several frequencies, the goal is to collect stations in groups so that for each group $\frac{1}{\lambda_{\max}}$ is maximized, or in other words, the largest eigenvalue of the link gain matrix G for each group is minimized.

Suppose we are given a normalized link gain matrix $G \in \mathbb{R}^{n \times n}$ which has zeros on the diagonal and non-negative entries otherwise,

$$G = \begin{bmatrix} 0 & g_{12} & g_{13} & \dots & g_{1n} \\ g_{21} & 0 & g_{23} & \dots & g_{2n} \\ \vdots & & \ddots & & \vdots \\ g_{n1} & g_{n2} & \dots & \dots & 0 \end{bmatrix}, \quad g_{ij} \geq 0, 1 \leq i, j \leq n,$$

and we have two frequencies to be assigned. To do so, we need to find a decomposition of a permutation Π of the matrix G into four sub blocks,

$$\Pi^T G \Pi = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad \begin{array}{ll} A \in \mathbb{R}^{p \times p}, & B \in \mathbb{R}^{p \times (n-p)}, \\ C \in \mathbb{R}^{(n-p) \times p}, & B \in \mathbb{R}^{(n-p) \times (n-p)}, \end{array}$$

such that the maximum of the largest eigenvalues of the sub matrices on the diagonal is minimized,

$$\min_{\Pi, p} (\max(\lambda_{max}(A), \lambda_{max}(D))). \tag{4.7}$$

An exhaustive search to solve this minimax problem is performed by the Matlab code

```
function s=BruteForcePartition(G)
% BRUTEFORCEPARTITION find best two frequency partition
%   s=BruteForcePartition(G); finds for a given link gain matrix G
%   the best partition into four submatrices, such that the maximum
%   of the largest eigenvalues of the two diagonal blocks is minimized.
%   The result is given by the binary index s for one of the submatrices.

n=size(G,1);
for k=1:2^(n-1)-1
  s=logical(double(dec2bin(k,n))-'0');
  r(k)=max([max(eig(G(s,s))) max(eig(G(~s,~s)))]);
end
[rmin,k]=min(r);
s=logical(double(dec2bin(k,n))-'0');
```

To test this code, we need various configurations of base stations. Since base stations are usually intelligently distributed, we use for testing purposes base stations on a rectangular grid, which has been randomly perturbed, as in the Matlab code

```
function [A,x,y]=GenerateProblem(n,m,de);
% GENERATEPROBLEM generates organized random set of base stations
%   [A,x,y]=GenerateProblem(n,m,de); generates n x m base station
%   locations on a rectangular unitary grid, each at a random location
%   in a square of side 2*de centered at the corresponding gridpoint,
%   and computes the corresponding link gain matrix using the fact
%   that the signal decays like 1/r^3.  The output is the link gain
%   matrix A and the coordinates of the base stations (x,y).

[X,Y]=meshgrid(1:n,1:m);
X=X+de*(rand(size(X))-1/2);
Y=Y+de*(rand(size(Y))-1/2);
x=X(:);
y=Y(:);
for i=1:n*m-1,
  for j=i+1:n*m,
    A(i,j)=1/(sqrt((x(i)-x(j))^2+(y(i)-y(j))^2)^3);
    A(j,i)=A(i,j);
  end;
end;
plot(x,y,'o','MarkerSize',12,'LineWidth',2)
```

We show in Figure 4.3 on the left the result of the commands

```
[A,x,y]=GenerateProblem(4,3,0.3);
s=PartitionBruteForce(A);
plot(x(s),y(s),'*',x(~s),y(~s),'o');
```

The minimum of the largest eigenvalue was found to be 1.2026, and one can see how frequencies are assigned intuitively to separate base station regions of influence. On the right in Figure 4.3, we show the results when the randomness parameter has been increased to 0.7. In this case, the minimum of the largest eigenvalue was found to be 2.9017, and it is already less evident why this partition should be optimal.

The exhaustive search is currently feasible for matrices of size up to 20×20, for which one has to try 2^{20} possibilities. Since the matrices arising in the radio communication problems are of order 100×100, a feasible strategy to compute approximations to the best solution is needed.

To do so, we first relate (4.7) to a problem involving norms: suppose the normalized link gain matrix G is symmetric, which would hold in an environment without obstacles. Then G is diagonalizable, $G = Q^T \Lambda Q$ where Q is orthogonal

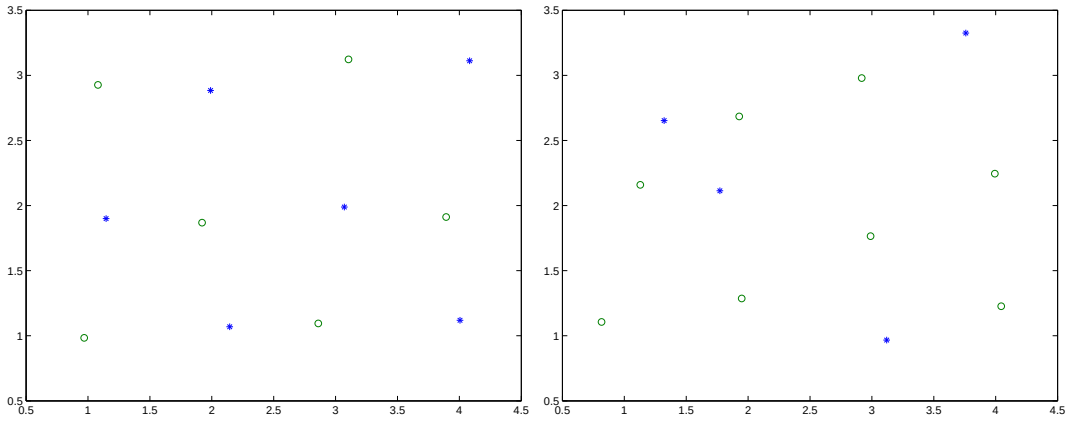


FIGURE 4.3.
Two configurations of base stations, and the best two frequency assignment found by exhaustive search.

and Λ is diagonal with the eigenvalues of G on the diagonal. Thus

$$||G||_2 = ||Q^T \Lambda Q||_2 = ||\Lambda||_2 = \lambda_{max}(G)$$

and the minimax problem in (4.7) can be written equivalently using the spectral norm,

$$\min_{\Pi,p}(\max(||A||_2, ||D||_2)). \tag{4.8}$$

Often the spectral norm is ideal for minimization problems, but in this case different norms are easier to handle. One could for example approximately minimize the one or infinity norm,

$$||A||_1 = \max_j \sum_i |a_{ij}|, \qquad ||A||_\infty = \max_i \sum_j |a_{ij}|,$$

or the Frobenius norm,

$$||A||_F = \sqrt{\sum_{ij} |a_{ij}|^2}.$$

The physical intuition behind the use of different norms is the following: If we manage to keep only weak links (small link gain) in the sub-matrices A and D and put strong links (large link gain) into the off diagonal blocks B and C, then the strong links lose their importance, because they link stations with different frequencies. So physically minimizing norms of A and D is still meaningful, even in the non symmetric case.

Suppose we want to minimize the Frobenius norm of the two sub blocks A and D,

$$\min_{\Pi,p}(\max(||A||_F^2, ||D||_F^2)). \tag{4.9}$$

Let $\boldsymbol{p} \in \mathbb{R}^n$ be a partitioning vector, $p_i \in \{-1, 1\}$. Then the sum of the Frobenius norms of A and D can be written as

$$||A||_F^2 + ||D||_F^2 \;=\; \frac{1}{4} \sum_{ij} g_{ij}^2 (p_i + p_j)^2$$

$$= \frac{1}{4}\sum_{ij} g_{ij}^2(p_i^2 + p_j^2) + \frac{1}{2}\sum_{ij} g_{ij}^2 p_i p_j$$
$$= \frac{1}{2}(||G||_F^2 + \boldsymbol{p}^T G^2 \boldsymbol{p})$$

and hence minimizing the sum of the two Frobenius norms $||A||_F^2 + ||D||_F^2$ is equivalent to minimizing

$$\min_{p_i \in \{-1,1\}} \boldsymbol{p}^T G^2 \boldsymbol{p} \tag{4.10}$$

where the square on G is understood to be element-wise. This formulation corresponds to the spectral bisection problem in graph partitioning, and it is shown to be NP-hard as well in the literature.

To find an approximation to the solution of (4.10) one can relax the constraint on $\boldsymbol{p}$ to contain only elements $+1$ or -1 and allows arbitrary $\boldsymbol{p}$. Thus the minimum is obtained for $\boldsymbol{p}$ being the eigenvector of G^2 associated with the smallest eigenvalue $\lambda_{min}(G^2)$. Such a search is performed by the Matlab code

```
function s=SpectralPartition(G)
% SPECTRALPARTITION find approximate two frequency partition
%   s=SpectralPartition(G); finds for a given link gain matrix
%   G an approximation to the best partition into four submatrices,
%   such that the maximum of the largest eigenvalues of the two
%   diagonal blocks is minimized.  The result is given by the binary
%   index s for one of the submatrices.

n=size(G,1);
G2=G.^2;
[V,E]=eig(G2);
[dummy,k]=min(diag(E));
[v,id]=sort(V(:,k));
for p=1:n-1
  r(p)=max([max(eig(G2(id(1:p),id(1:p)))) max(eig(G2(id(p+1:n),id(p+1:n))))]);
end
[rmin,p]=min(r);
s=logical(dec2bin(sum(2.^(id(1:p)-1)),n)-'0');
```

There are several ways one can then decide which station uses which frequency: one could simply use the sign of the entries in the eigenvector, or one could search the optimal partitioning point, as we did in the algorithm above.

Using the same configuration as in Figure 4.3, we obtain with the spectral approximate optimization the results in Figure 4.4. On the left, with the randomness parameter equal to 0.3, the approximate minimum of the largest eigenvalue was found to be 1.2026, and is identical to the result of the exhaustive search. On the right in Figure 4.4, we show the results when the randomness parameter has been increased to 0.7, and now the approximate minimum of the

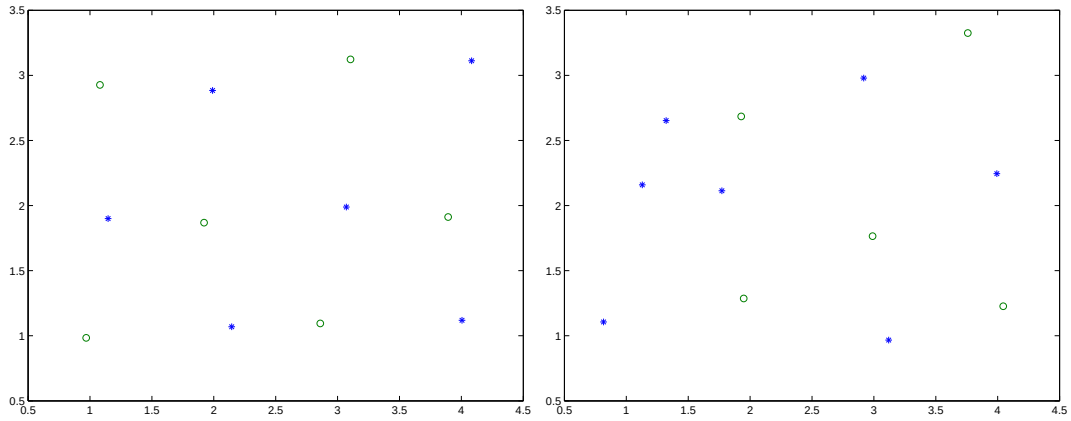


FIGURE 4.4.
Two configurations of base stations, and the best two frequency assignment found by approximate spectral search.

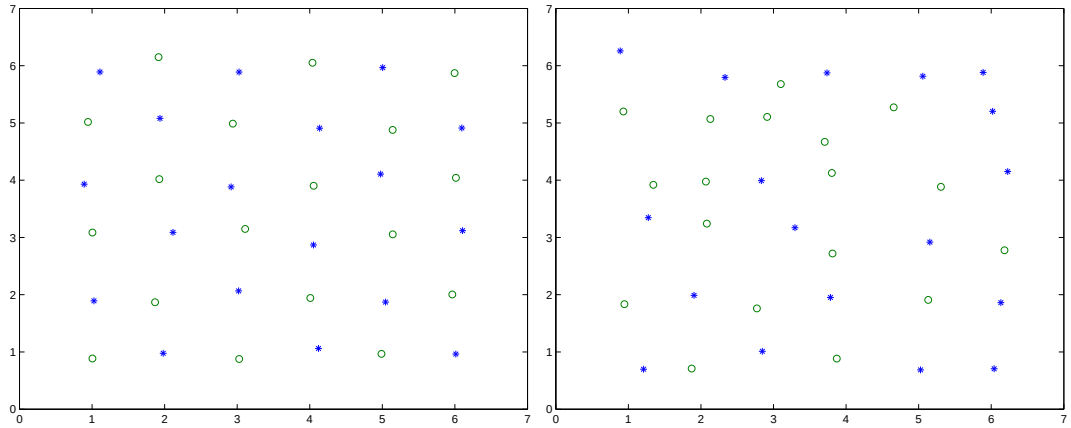


FIGURE 4.5.
Two larger configurations of base stations, and the best two frequency assignment found by approximate spectral optimization.

largest eigenvalue was found to be 3.0173, which is not the optimal partition found by the exhaustive search in Figure 4.3 on the right.

With the approximate optimization, we can however solve approximately much larger problems. The commands

```
[A,x,y]=GenerateProblem(6,6,0.3);
s=PartitionSpectral(A);
plot(x(s),y(s),'*',x(~s),y(~s),'o');
```

lead to the result in Figure 4.5 on the left, and on the right with the increased randomness parameter 0.7.

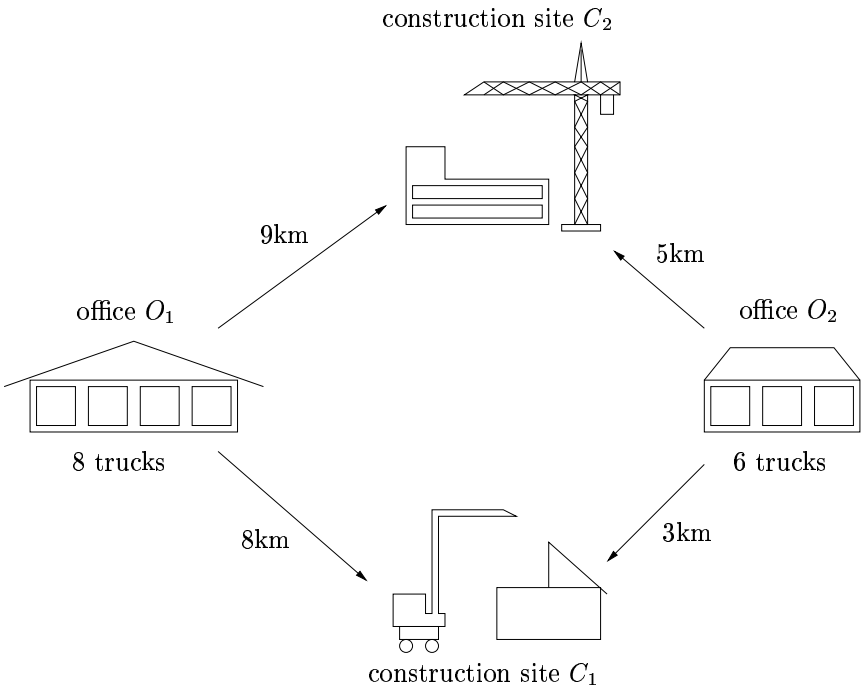


FIGURE 4.6.
A construction company specialized in transport wants to optimize the use of their trucks.

4.1.3 A Problem from Operations Research

A construction company specialized in transport has two offices at different locations, O_1 and O_2, with 8 trucks available at O_1 and 6 trucks available at O_2. The company is currently servicing two construction sites C_1 and C_2. The site C_1 needs 4 trucks to be operational, and the site C_2 needs 7 trucks. The distances from office O_1 to construction site C_1 is 8 kilometers, and to construction site C_2 it is 9 kilometers. The distances from office O_2 to construction site C_1 is 3 kilometers, and to construction site C_2 it is 5 kilometers, as illustrated in Figure 4.6. The construction company would like to minimize fuel cost and hence send trucks to the construction sites such that the overall distance the trucks have to drive is minimized. If we denote by x_1 the number of trucks sent from O_1 to C_1, by x_2 the number of trucks sent from O_1 to C_2, by x_3 the number of trucks sent from O_2 to C_1, by x_4 the number of trucks sent from O_2 to C_2, the company needs to make a choice satisfying the following conditions:

$$\begin{array}{rcll} x_1 + x_2 & \leq & 8 & \text{no more than 8 trucks at } O_1, \\ x_3 + x_4 & \leq & 6 & \text{no more than 6 trucks at } O_2, \\ x_1 + x_3 & = & 4 & \text{need 4 trucks at } C_1, \\ x_2 + x_4 & = & 7 & \text{need 7 trucks at } C_2, \\ x_i & \geq & 0 & \text{number of trucks need to be non-negative.} \end{array} \tag{4.11}$$

In addition, the choice of x_i, $i = 1, 2, 3, 4$ needs to be such that the fuel use is minimized, i.e. the objective function

$$8x_1 + 9x_2 + 3x_3 + 5x_4 \longrightarrow \min.$$

To simplify this problem, one can first eliminate the variables x_3 and x_4 using the equalities in (4.11),

$$x_3 = 4 - x_1, \qquad x_4 = 7 - x_2,$$

which leads to the simpler optimization problem

$$\begin{array}{rcl} 5x_1 + 4x_2 + 47 & \longrightarrow & \min \\ x_1 + x_2 & \leq & 8 \\ -x_1 - x_2 & \leq & -5 \\ x_1 & \leq & 4 \\ x_2 & \leq & 7 \\ x_1 & \geq & 0 \\ x_2 & \geq & 0. \end{array} \tag{4.12}$$

Since we have now only two variables left, one can solve this problem graphically, as shown in Figure 4.7. The inequalities in (4.12) determine the region shown in Figure 4.7, where all admissible solutions of the operations research problem of the construction company are. The line with slope $-\frac{5}{4}$ represents the objective function, which should be as low as possible. Moving this line as low as possible, without leaving the region of all admissible solutions gives the solution $x_1 = 0$ and $x_2 = 5$, which implies $x_3 = 4$ and $x_4 = 2$. This solution leads to 4 trucks at construction site C_1 and 7 trucks at construction site C_2, as required, and the trucks drive the overall distance

$$8x_1 + 9x_2 + 3x_3 + 5x_4 = 67\text{km},$$

which is the smallest distance possible. The solution is interesting, since 5 trucks drive the longest distance in the network at the optimal solution which minimizes the overall distance, whereas a less carefully, intuitively chosen solution could have sent only one truck for the longest distance, which would have led to a worse overall solution of 71 km.

4.1.4 Classification of Optimization Problems

In general an optimization problem at the mathematical level is given by an objective function $f : K \longrightarrow \mathbb{R}$, and one searches for $x^* \in K$, such that $f(x^*) \leq f(x)$ for all $x \in K$. Often, K is a subset of $\mathbb{R}^n$ defined by constraints, and the optimization problem is formulated as

$$\begin{array}{rcll} f(x) & \longrightarrow & \min & x \in \mathbb{R}^n, \\ c_j(x) & \geq & 0 & j \in I, \\ c_j(x) & = & 0 & j \in E, \end{array} \tag{4.13}$$

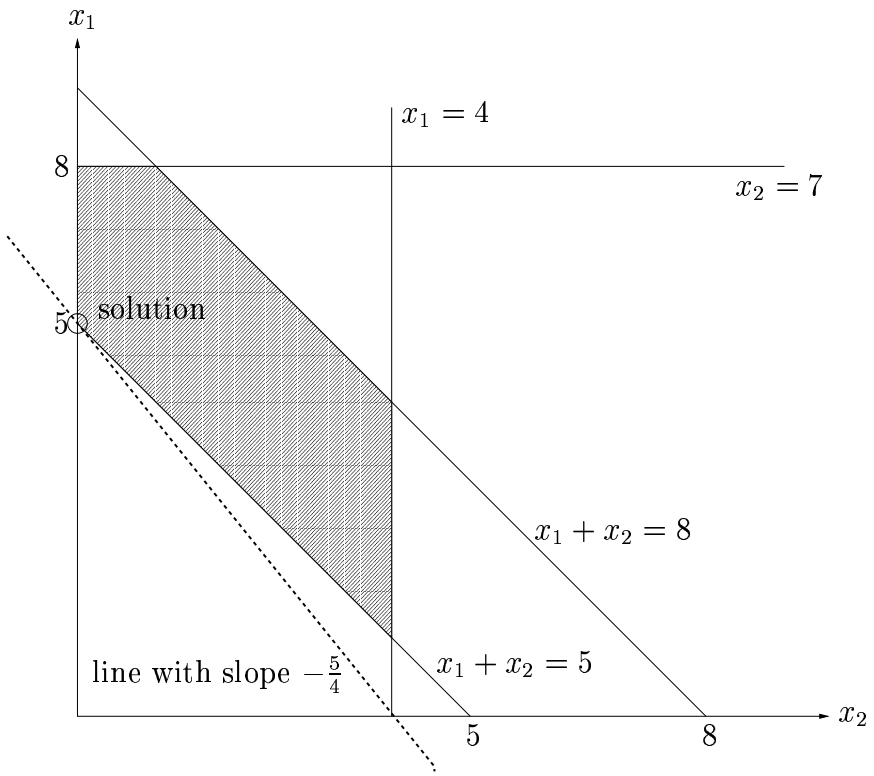


FIGURE 4.7.
Graphical solution of the operations research problem of the construction company.

where I is an index set denoting the inequality constraints, and E is an index set denoting the equality constraints of the problem, whose sizes we denote by $m_I = |I|$ and $m_E = |E|$, and $c_j : \mathbb{R}^n \longrightarrow \mathbb{R}$ are the constraints.

One can roughly divide optimization problems into the following categories:

1. Optimization problems without constraints: $m_I = m_E = 0$
   (a) Quadratic optimization problems: the objective function f is quadratic, i.e. $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$, where $A \in \mathbb{R}^{n\times n}$ is a symmetric matrix. Such problems require the solution of a linear system, as shown in Chapter 3, and the conjugate gradient algorithm can be conveniently used for its solution.
   (b) General nonlinear optimization problems without constraints: f is neither quadratic nor linear.
2. Optimization problems with linear constraints: the functions c_j are affine.
   (a) Problems with equality constraints only, $m_I = 0$.
   (b) Problems with inequality constraints:
      i. Linear programming, if f is linear.
      ii. Quadratic-linear optimization problems, if f is quadratic.
      iii. Nonlinear optimization problems with linear constraints, if f is neither linear nor quadratic.
   (c) Nonlinear optimization problems:
      i. Nonlinear optimization problems with equality constraints.
      ii. General nonlinear optimization problems.
3. Optimal control problems, where x is a function of one or several parameters.
4. Combinatorial optimization problems, where the set $K \subset \mathbb{R}^n$ is discrete, or even finite.

We will not address the combinatorial optimization problems further than in the example in section 4.1.2 on portable phone networks. The techniques used for such problems are substantially different from the techniques used for all the other cases. Optimal control problems however could be handled by the techniques of this chapter after discretization.

4.2 Mathematical Optimization

A function $f : K \longrightarrow \mathbb{R}$ with $K \subset \mathbb{R}^n$ has a local minimum at the point $\boldsymbol{x}^* \in K$ if

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in K \cap U, \tag{4.14}$$

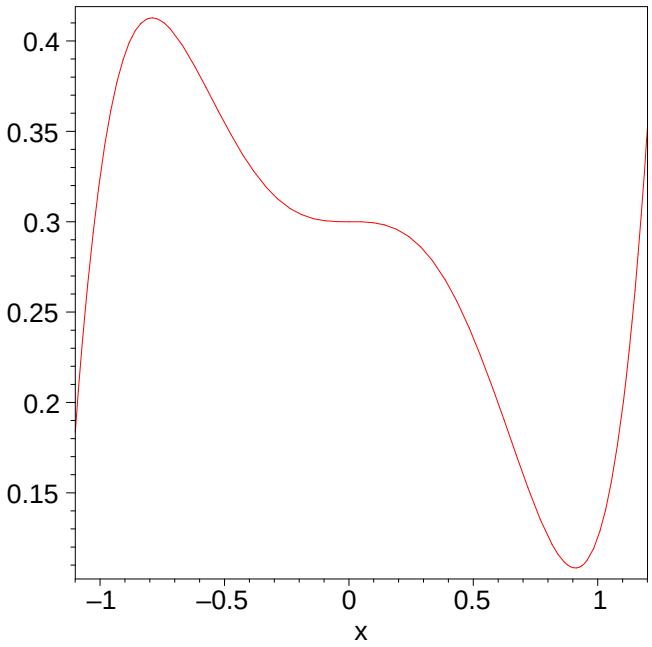


FIGURE 4.8. *The function* $f(x) = \frac{1}{2}x^5 - \frac{3}{40}x^4 - \frac{3}{5}x^3 + \frac{3}{10}$.

where U is a neighborhood of $\boldsymbol{x}^*$. The function f has a global minimum at the point $\boldsymbol{x}^* \in K$ if

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in K. \tag{4.15}$$

We say that the minimum is strict, if we have $f(\boldsymbol{x}^*) < f(\boldsymbol{x})$ for all $\boldsymbol{x} \neq \boldsymbol{x}^*$ in (4.14) or (4.15). Note that to obtain results for maxima, it suffices to simply change the sign and replace f by $-f$.

Definitions (4.14) and (4.15) are not very useful in practice, and the topic of this section is to establish other, more easily verifiable conditions which are necessary or sufficient for at least local minima. The cases are rare where one can make a statement about global minima.

4.2.1 Local Minima

We start with a simple one dimensional example: let $f : \mathbb{R} \longrightarrow \mathbb{R}$ be given by $f(x) = \frac{1}{2}x^5 - \frac{3}{40}x^4 - \frac{3}{5}x^3 + \frac{3}{10}$, a function shown in Figure 4.8, obtained by the maple commands

```
f:=1/2*x^5-3/40*x^4-3/5*x^3+3/10;
plot(f,x=-1.1..1.2,axes=boxed);
```

Clearly the function has a maximum on the left, and a minimum on the right. We know from calculus that if f is twice continuously differentiable, then

$$\left\{ \begin{array}{rcl} f'(x^*) & = & 0 \\ f''(x^*) & > & 0 \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} f \text{ has a local} \\ \text{minimum at } x^* \end{array} \right\} \Longrightarrow \left\{ \begin{array}{rcl} f'(x^*) & = & 0 \\ f''(x^*) & \geq & 0 \end{array} \right\}. \tag{4.16}$$

The conditions on the left are sufficient conditions for f to have minimum, and the conditions on the right are necessary conditions for f to have a minimum:

if they are not satisfied, there can not be a minimum. For the example, we can verify with the Maple commands

```
fp:=diff(f,x);
fpp:=diff(f,x,x);
sols:=solve(fp,x);
```

$$sols := 0,\ 0,\ \frac{3}{50} + \frac{3\sqrt{201}}{50},\ \frac{3}{50} - \frac{3\sqrt{201}}{50}$$

```
x:=sols[1];
evalf([f,fp,fpp]);
```

$$[.3000000000, 0., 0.]$$

```
x:=sols[3];
evalf([f,fp,fpp]);
```

$$[.1084415149, 0., 3.527111813]$$

```
x:=sols[4];
evalf([f,fp,fpp]);
```

$$[.4127585747, 0., -2.658791812]$$

The example also shows that the vanishing derivative at $x = 0$ does not imply an extrema, since the second derivative vanishes as well there. The fact that a strict minimum does not imply a strictly positive second derivative is easily seen by looking at the function $f(x) = x^4$ for example.

To generalize the result (4.16) to n dimensions, we use the Taylor series with remainder: for $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ twice continuously differentiable, we have

$$f(\boldsymbol{x}^* + \boldsymbol{x}) = f(\boldsymbol{x}^*) + f'(\boldsymbol{x}^*)\boldsymbol{x} + \frac{1}{2}f''(\boldsymbol{x}^*)(\boldsymbol{x}, \boldsymbol{x}) + r(\boldsymbol{x})||\boldsymbol{x}||^2, \tag{4.17}$$

where $r(\boldsymbol{x}) \to 0$ as $\boldsymbol{x} \to \boldsymbol{0}$. In (4.17), f' is the transpose of the gradient of f

$$(f'(\boldsymbol{x}^*))^T = \nabla f(\boldsymbol{x}^*) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(\boldsymbol{x}^*) \\ \frac{\partial f}{\partial x_2}(\boldsymbol{x}^*) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\boldsymbol{x}^*) \end{pmatrix},$$

and f'' is the bilinear form defined by the Hessian of f,

$$f''(\boldsymbol{x}^*)(\boldsymbol{x}, \boldsymbol{x}) = \boldsymbol{x}^T \nabla^2 f(\boldsymbol{x}^*)\boldsymbol{x} = \boldsymbol{x}^T H(\boldsymbol{x}^*)\boldsymbol{x},$$

where the symmetric Hessian matrix is given by

$$H(\boldsymbol{x}^*) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(\boldsymbol{x}^*) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(\boldsymbol{x}^*) & \cdots \frac{\partial^2 f}{\partial x_1 \partial x_n}(\boldsymbol{x}^*) \\ \frac{\partial^2 f}{\partial x_1 \partial x_2}(\boldsymbol{x}^*) & \frac{\partial^2 f}{\partial x_2^2}(\boldsymbol{x}^*) & \cdots \frac{\partial^2 f}{\partial x_2 \partial x_n}(\boldsymbol{x}^*) \\ \vdots \ddots & \vdots & \\ \frac{\partial^2 f}{\partial x_1 \partial x_n}(\boldsymbol{x}^*) & \frac{\partial^2 f}{\partial x_2 \partial x_n}(\boldsymbol{x}^*) & \cdots \frac{\partial^2 f}{\partial x_n^2}(\boldsymbol{x}^*) \end{bmatrix}.$$

THEOREM 4.1 (UNCONSTRAINED SUFFICIENT AND NECESSARY OPTIMALITY CONDITIONS). *Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be twice continuously differentiable, and let $\boldsymbol{x}^* \in \mathbb{R}^n$. Then*

$$\left\{ \begin{array}{c} \nabla f(\boldsymbol{x}^*) = 0 \\ \boldsymbol{x}^T H(\boldsymbol{x}^*)\boldsymbol{x} > 0 \\ \textit{for all } \boldsymbol{x} \in \mathbb{R}^n \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} f \textit{ has a} \\ \textit{local minimum} \\ \textit{at } \boldsymbol{x}^* \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} \nabla f(\boldsymbol{x}^*) = 0 \\ \boldsymbol{x}^T H(\boldsymbol{x}^*)\boldsymbol{x} \geq 0 \\ \textit{for all } \boldsymbol{x} \in \mathbb{R}^n \end{array} \right\}. \tag{4.18}$$

PROOF. To show the sufficient condition, let $\lambda_{\min}$ be the smallest eigenvalue of $H(\boldsymbol{x}^*)$. Then

$$\boldsymbol{x}^T H(\boldsymbol{x}^*)\boldsymbol{x} \geq \lambda_{\min} \boldsymbol{x}^T \boldsymbol{x} \quad \forall \boldsymbol{x} \in \mathbb{R}^n.$$

The Taylor series (4.17) with $\nabla f(\boldsymbol{x}^*) = 0$ then implies

$$f(\boldsymbol{x}^* + \boldsymbol{x}) - f(\boldsymbol{x}^*) \geq \left(\frac{1}{2}\lambda_{\min} + r(\boldsymbol{x}) \right) ||\boldsymbol{x}||^2.$$

Now if $\boldsymbol{x}^T H(\boldsymbol{x}^*)\boldsymbol{x} > 0$, $\lambda_{\min} > 0$, and hence $f(\boldsymbol{x}^* + \boldsymbol{x}) > f(\boldsymbol{x}^*)$ for sufficiently small but non-zero $\boldsymbol{x}$, which means $\boldsymbol{x}^*$ is a strict local minimum.

To show the necessary condition, we apply the one dimensional result (4.16) to the function $g(t) = f(\boldsymbol{x}^* + t\boldsymbol{x})$. If f has a local minimum at $\boldsymbol{x}^*$, then g also has a local minimum at $t = 0$, and hence (4.16) implies on the one hand

$$g'(0) = 0 \Longrightarrow (f(\boldsymbol{x}^*))^T \boldsymbol{x} = 0 \ \forall \boldsymbol{x} \in \mathbb{R}^n \Longrightarrow \nabla f(\boldsymbol{x}^*) = 0,$$

and on the other hand

$$g''(0) \geq 0 \Longrightarrow \boldsymbol{x}^T H(\boldsymbol{x}^*)\boldsymbol{x} \geq 0 \ \forall \boldsymbol{x} \in \mathbb{R}^n,$$

which concludes the proof. □

4.2.2 Constrained minima and Lagrange multipliers

We start again by an example: suppose we want to minimize $f = x_1 x_2$ on the manifold $\mathcal{M} = \{(x_1, x_2) | g(x_1, x_2) = x_1^2 + x_2^2 - 1 = 0\}$. The manifold is a circle, and it is instructive to look at the contours of f together with the circle, as shown in Figure 4.9, which was obtained with the MAPLE commands

```
f:=x1*x2;
```

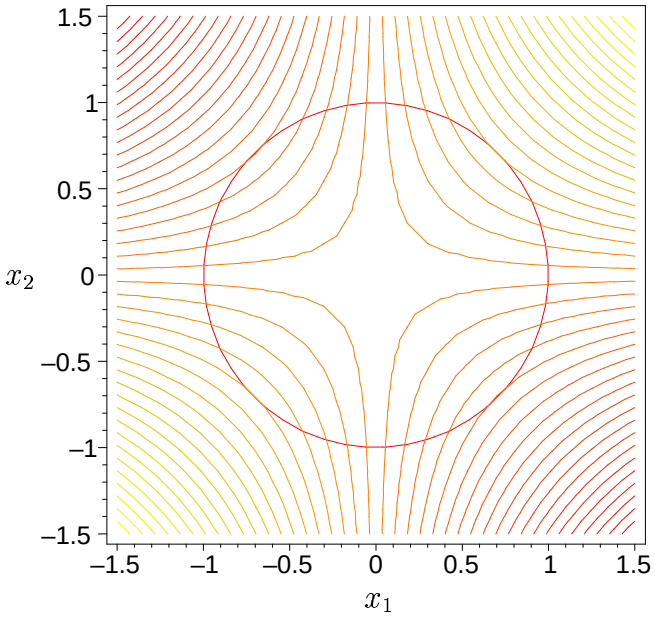


FIGURE 4.9. *Minimization of $f = x_1 x_2$ on the unit circle.*

```
g:=x1^2+x2^2-1;
with(plots):
P1:=contourplot(f,x1=-1.5..1.5,x2=-1.5..1.5,contours=40,axes=boxed):
P2:=contourplot(g,x1=-1.5..1.5,x2=-1.5..1.5,contours=[0],axes=boxed):
display([P1,P2]);
```

Since f is positive and increasing in the first and third quadrant, and negative and decreasing in the second and fourth, we can see from the plot that the minima on the circle are attained at $(x_1, x_2) = (-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ and $(x_1, x_2) = (\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$.

We now present a geometric and an algebraic argument to find these points. Geometrically, we see from Figure 4.9 that the extremal points of the constraint optimization problem are characterized by tangent level set curves of f to the constraint curve defined by g. To understand this, we note that the gradient of a function is always orthogonal to its level sets, as one can see for example from the Taylor expansion,

$$f(\boldsymbol{x}_0 + \boldsymbol{x}) - f(\boldsymbol{x}_0) = (\nabla f(\boldsymbol{x}_0))^T \boldsymbol{x} + O(||\boldsymbol{x}||^2).$$

Now the minimum can not be at an intersection of the level sets of f and g, since one could then immediately decrease f by continuing a bit on the zero level set of g. Hence the level sets must be tangential, which implies that the gradient of f must be parallel to the gradient of g, or in other words, at the solutions of this problem, there must be a parameter $\lambda \in \mathbb{R}$ such that

$$\nabla f(\boldsymbol{x}^*) = \lambda \nabla g(\boldsymbol{x}^*). \tag{4.19}$$

In addition, we must be on the manifold, i.e. $g(\boldsymbol{x}^*) = 0$, which together with (4.19) forms a system of equations for the unknowns $\boldsymbol{x}^*$ and λ. In our example,

we obtain with the MAPLE commands

```
with(linalg):
fp:=grad(f,[x1,x2]);
gp:=grad(g,[x1,x2]);
sols:=solve({seq(fp[i]-lambda*gp[i],i=1..2),g},{x1,x2,lambda});
allvalues(sols[1]);
allvalues(sols[2]);
```

the solutions

$$\{x1 = \frac{\sqrt{2}}{2},\ x2 = \frac{\sqrt{2}}{2},\ \lambda = \frac{1}{2}\},\ \{x1 = -\frac{\sqrt{2}}{2},\ x2 = -\frac{\sqrt{2}}{2},\ \lambda = \frac{1}{2}\}$$

$$\{x2 = \frac{\sqrt{2}}{2},\ x1 = -\frac{\sqrt{2}}{2},\ \lambda = \frac{-1}{2}\},\ \{x1 = \frac{\sqrt{2}}{2},\ x2 = -\frac{\sqrt{2}}{2},\ \lambda = \frac{-1}{2}\}$$

as expected. Note that from these first order conditions, we can not infer which of the solutions represent actual minima.

To find an algebraic argument, it is easiest to start with a parametric representation of the manifold, in our example the circle, which can be represented by the function

$$\boldsymbol{\varphi}(t) = \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}.$$

Now the optimization problem becomes an unconstrained optimization problem by substituting the parametrization into the function and then minimizing, i.e. minimizing the function

$$F(t) := f(\boldsymbol{\varphi}(t)) = \sin(t)\cos(t) = \frac{1}{2}\sin(2t).$$

Computing the first and second derivative, we find

$$F'(t) = \cos(2t), \quad F''(t) = -2\sin(2t),$$

and hence the minima are at $\frac{3\pi}{4}$ and $\frac{7\pi}{4}$, as desired.

For a general function $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$ and a one dimensional manifold $\mathcal{M}$ given by the parametrization $\boldsymbol{\varphi} : \mathbb{R} \longrightarrow \mathbb{R}^2$, we obtain at the extrema

$$F'(t) = (f(\boldsymbol{\varphi}(t)))' = (\nabla f(\boldsymbol{\varphi}(t)))^T \boldsymbol{\varphi}'(t) = 0.$$

At a minimum $\boldsymbol{x}^*$, we therefore have

$$\nabla f(\boldsymbol{x}^*) \perp \boldsymbol{\varphi}'(t^*),$$

where $\boldsymbol{\varphi}'(t^*)$ is the tangent to the manifold $\mathcal{M}$ at $\boldsymbol{x}^*$.

THEOREM 4.2 (CONSTRAINED SUFFICIENT AND NECESSARY OPTIMALITY CONDITIONS 1). *Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be twice continuously differentiable, and let*

$\boldsymbol{\varphi} : \mathbb{R}^k \longrightarrow \mathbb{R}^n$ be a local parametrization of the manifold $\mathcal{M}$ in a neighborhood of $\boldsymbol{x}^* \in \mathbb{R}^n$, $\boldsymbol{\varphi}(\boldsymbol{z}^*) = \boldsymbol{x}^*$. Then

$$\left\{ \begin{array}{c} (\nabla f(\boldsymbol{x}^*))^T \boldsymbol{\varphi}(\boldsymbol{z}^*) = 0 \\ \boldsymbol{z}^T H(\boldsymbol{z}^*) \boldsymbol{z} > 0 \\ \text{for all } \boldsymbol{z} \in \mathbb{R}^k \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} f|_{\mathcal{M}} \text{ has a} \\ \text{local minimum} \\ \text{at } \boldsymbol{x}^* \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} (\nabla f(\boldsymbol{x}^*))^T \boldsymbol{\varphi}(\boldsymbol{z}^*) = 0 \\ \boldsymbol{z}^T H(\boldsymbol{z}^*) \boldsymbol{z} \geq 0 \\ \text{for all } \boldsymbol{z} \in \mathbb{R}^k \end{array} \right\}, \tag{4.20}$$

where $H(\boldsymbol{z})$ is the Hessian matrix of $\boldsymbol{z} \mapsto f(\boldsymbol{\varphi}(\boldsymbol{z}))$.

PROOF. It suffices to apply Theorem 4.1 for the unconstrained case to the function $F(\boldsymbol{z}) := f(\boldsymbol{\varphi}(\boldsymbol{z}))$. □

The condition $(\nabla f(\boldsymbol{x}^*))^T \boldsymbol{\varphi}'(\boldsymbol{z}^*) = 0$ means that $\nabla f(\boldsymbol{x}^*)$ is orthogonal to the tangent space of the manifold $\mathcal{M}$ at $\boldsymbol{x}^*$, $T_{\boldsymbol{x}^*}\mathcal{M}$. If the manifold is given by a set of equations $\boldsymbol{g}(\boldsymbol{x}) = 0$, the tangent space $T_{\boldsymbol{x}^*}\mathcal{M}$ is $\ker(\boldsymbol{g}'(\boldsymbol{x}^*))$, where $\boldsymbol{g}'$ denotes the Jacobi matrix of $\boldsymbol{g}$. Now

$$\boldsymbol{x} \in \ker A \Longleftrightarrow A\boldsymbol{x} = 0 \Longleftrightarrow \boldsymbol{y}^T A \boldsymbol{x} = 0, \ \forall \boldsymbol{y} \Longleftrightarrow \boldsymbol{x} \perp A^T \boldsymbol{y} \Longleftrightarrow \boldsymbol{x} \perp \Im(A^T),$$

and hence

$$\nabla f(\boldsymbol{x}^*) \perp \ker(\boldsymbol{g}'(\boldsymbol{x}^*)) \Longleftrightarrow \nabla f(\boldsymbol{x}^*) \in \Im((\boldsymbol{g}'(\boldsymbol{x}^*))^T),$$

or in other words, there exists a vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)^T$ such that

$$\nabla f(\boldsymbol{x}^*) = (\boldsymbol{g}'(\boldsymbol{x}^*))^T \boldsymbol{\lambda}. \tag{4.21}$$

The λ_j, $j = 1, \ldots, m$ are called Lagrange multipliers and defining the Lagrangian by

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) := f(\boldsymbol{x}) - (\boldsymbol{g}(\boldsymbol{x}))^T \boldsymbol{\lambda}, \tag{4.22}$$

equation (4.21) is equivalent to $\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = 0$, while $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$ is equivalent to $\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = 0$. We therefore have the necessary condition $\nabla \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = 0$ for a minimum of f on the manifold defined by $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$.

THEOREM 4.3 (CONSTRAINED SUFFICIENT AND NECESSARY OPTIMALITY CONDITIONS 2). *Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be twice continuously differentiable, and let $\mathcal{M} = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{g}(\boldsymbol{x}) = 0\}$ be a manifold, $\boldsymbol{x}^* \in \mathcal{M}$ and $\boldsymbol{\lambda} \in \mathbb{R}^m$. Then, with the Lagrangian (4.22), we have*

$$\left\{ \begin{array}{c} \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}) = 0 \\ \boldsymbol{w}^T \nabla^2_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}) \boldsymbol{w} > 0 \\ \forall \boldsymbol{w} \in T_{\boldsymbol{x}^*}\mathcal{M}, \ \boldsymbol{w} \neq \boldsymbol{0} \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} f|_{\mathcal{M}} \text{ has a} \\ \text{local minimum} \\ \text{at } \boldsymbol{x}^* \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}) = 0 \\ \boldsymbol{w}^T \nabla^2_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}) \boldsymbol{w} \geq 0 \\ \forall \boldsymbol{w} \in T_{\boldsymbol{x}^*}\mathcal{M} \end{array} \right\}. \tag{4.23}$$

PROOF. See the problem section. □

4.2.3 Equality and Inequality Constraints

For the continuously differentiable functions $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, $\boldsymbol{c}_E : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ and $\boldsymbol{c}_I : \mathbb{R}^n \longrightarrow \mathbb{R}^k$, we look for local minima of the problem

$$\begin{array}{rcl} f(\boldsymbol{x}) & \longrightarrow & \min, \\ \boldsymbol{c}_E(\boldsymbol{x}) & = & \boldsymbol{0}, \\ \boldsymbol{c}_I(\boldsymbol{x}) & \geq & \boldsymbol{0}. \end{array} \tag{4.24}$$

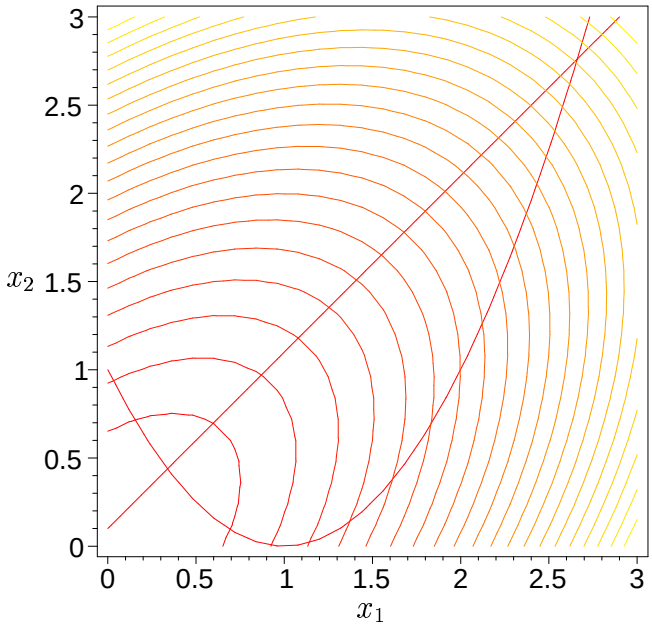


FIGURE 4.10.
Level sets of the function to minimize, and zero level sets of the two inequality constraints of an example.

While such a problem can not be treated analytically in general, we show in the following example that in principle it could be treated analytically.

EXAMPLE 4.2.1. *Let* $f(\boldsymbol{x}) = x_1^2 + x_2^2 + x_3^2$, $c_E(\boldsymbol{x}) = x_1 - x_2 - x_3$ *and* $\boldsymbol{c}_I(\boldsymbol{x}) = (x_1 - x_2 + \frac{1}{10}, x_2 - (x_3 + x_2 - 1)^2)^T$. *We first use the equality constraint to eliminate the variable* $x_3 = x_1 - x_2$, *which leads to the simplified problem with* $f(\boldsymbol{x}) = 2(x_1^2 + x_1 x_2 + x_2^2)$ *and* $\boldsymbol{c}_I(\boldsymbol{x}) = (x_1 - x_2 + \frac{1}{10}, x_2 - (x_1 - 1)^2)^T$. *Using the* MAPLE *commands*

```
f:=x1^2+x2^2+x3^2;
cE:=x1-x2-x3;
cI1:=x1-x2+1/10;
cI2:=x2-(x3+x2-1)^2;
x3:=solve(cE,x3);
with(plots):
P1:=contourplot(f,x1=0..3,x2=0..3,contours=20,axes=boxed):
P2:=contourplot(cI1,x1=0..3,x2=0..3,contours=[0],axes=boxed):
P3:=contourplot(cI2,x1=0..3,x2=0..3,contours=[0],axes=boxed):
display([P1,P2,P3]);
```

we obtain Figure 4.10. From this figure, we can see that in the interior of the domain bounded by the inequality constraints, f does not have any extrema, as one can also readily test with the MAPLE commands

```
with(linalg):
```

```
fp:=grad(f,[x1,x2]);
solve({fp[1],fp[2]},{x1,x2});
```

which gives only the zero solution. We therefore need to search for extrema on the boundary of the compact set. Using Lagrange multipliers, we obtain with MAPLE on the straight line constraint

```
L1:=f-lambda*cI1;
L1p:=grad(L1,[x1,x2,lambda]);
solve({seq(L1p[i],i=1..3)},{x1,x2,lambda});
```

which gives the only solution $(x_1, x_2) = (-\frac{1}{20}, \frac{1}{20})$, outside of the compact set, which is also easily identified looking at Figure 4.10, slightly outside on the left. On the second inequality constraint, we obtain

```
L2:=f-lambda*cI2;
L2p:=grad(L2,[x1,x2,lambda]);
solve({seq(L2p[i],i=1..3)},{x1,x2,lambda});
sols:=allvalues(%);
evalf(sols[1]);
assign(%);
f;
```

and hence there is an extrema at $(x_1, x_2) \approx (0.3929927044, 0.368457857)$, which is indeed a minimum, on the boundary of the compact set, and the value of f at this point is .2908064168. The other two solutions are complex, as one readily checks using the commands `evalf(sols[2]); evalf(sols[3]);`. We finally need to check the corners of the boundary of the compact set,

```
x1:='x1';x2:='x2';
x2c1:=solve(cI1,x2);
x2c2:=solve(cI2,x2);
x1sols:=solve(x2c2=x2c1,x1);
x1:=x1sols[1];x2:=x2c1;
evalf(f);
x1:=x1sols[2];x2:=x2c1;
evalf(f);
```

which gives for the function values at the corners 14.72374902 and 0.3162509758, and hence the local minimum found earlier on the boundary is indeed the global minimum, and hence solution of the optimization problem.

As one can see from this simple example, an analytical treatment of such optimization problems is hopeless for realistic high dimensional problems. We will therefore look in the sequel at numerical techniques to find local minima.

As we have seen in subsection 4.2.2, equality constraints can be dealt with Lagrange multipliers, and Theorem 4.2 gave sufficient and necessary conditions for optimality in that case. It is natural that for a problem with inequality

constraints, only the constraints active at a local optimum play a role in the optimality conditions, i.e the constraints which satisfy at the optimum $\boldsymbol{x}^*$ the relation $(\boldsymbol{c}_I)_i(\boldsymbol{x}^*) = 0$. We call the set $A = \{i|(\boldsymbol{c}_I)_i(\boldsymbol{x}^*) = 0\}$ the active set for the local optimum $\boldsymbol{x}^*$.

THEOREM 4.4. *Suppose that $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ and $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ are twice continuously differentiable and that $\boldsymbol{x}^* \in \mathbb{R}^n$ is a local minimum of*

$$\begin{aligned} f(\boldsymbol{x}) &\longrightarrow \min \\ \boldsymbol{g}(\boldsymbol{x}) &\geq 0. \end{aligned}$$

Then there exists $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that

$$\begin{aligned} \boldsymbol{g}(\boldsymbol{x}^*) &\geq 0, \\ \nabla f(\boldsymbol{x}^*) - (\nabla \boldsymbol{g}(\boldsymbol{x}^*))^T \boldsymbol{\lambda} &= 0, \quad \textit{with } \boldsymbol{\lambda} \geq 0 \\ g_i(\boldsymbol{x}^*)\lambda_i &= 0. \end{aligned}$$

PROOF. We consider a perturbation around $\boldsymbol{x}^*$ which satisfies the constraints. Since the constraints which are not active at $\boldsymbol{x}^*$, i.e. $g_i(\boldsymbol{x}^*) > 0$, remain inactive in a neighborhood of $\boldsymbol{x}^*$, it suffices to consider the active constraints only, $g_i(\boldsymbol{x}^*) = 0$ for $i \in A$.

Let $\boldsymbol{x}(\alpha)$, $\alpha \geq 0$ be a twice continuously differentiable path with $\boldsymbol{x}(0) = \boldsymbol{x}^*$, i.e.

$$\boldsymbol{x}(\alpha) = \boldsymbol{x}^* + \alpha \boldsymbol{s} + \frac{1}{2}\alpha^2 \boldsymbol{p} + O(\alpha^3),$$

along which the active constraints remain satisfied, i.e. $g_i(\boldsymbol{x}(\alpha)) \geq 0$ for $i \in A$. This implies that

$$\begin{aligned} 0 &\leq g_i(\boldsymbol{x}(\alpha)) \\ &= g_i(\boldsymbol{x}^* + \alpha \boldsymbol{s} + \frac{1}{2}\alpha^2 \boldsymbol{p} + O(\alpha^3)) \\ &= g_i(\boldsymbol{x}^*) + (\alpha \boldsymbol{s} + \frac{1}{2}\alpha^2 \boldsymbol{p})^T \nabla g_i(\boldsymbol{x}^*) + \frac{1}{2}\alpha^2 \boldsymbol{s}^T H_i(\boldsymbol{x}^*)\boldsymbol{s} + O(\alpha^3)) \\ &= \alpha \boldsymbol{s}^T \nabla g_i(\boldsymbol{x}^*) + + \frac{1}{2}\alpha^2 \left(\boldsymbol{p}^T \nabla g_i(\boldsymbol{x}^*) + \boldsymbol{s}^T H_i(\boldsymbol{x}^*)\boldsymbol{s}\right) + O(\alpha^3)), \end{aligned}$$

where $H_i(\boldsymbol{x})$ denotes the Hessian of $g_i(\boldsymbol{x})$. Hence for α small, we must either have

$$\boldsymbol{s}^T \nabla g_i(\boldsymbol{x}^*) \geq 0,$$

or, if $\boldsymbol{s}^T \nabla g_i(\boldsymbol{x}^*) = 0$, we must have

$$\boldsymbol{p}^T \nabla g_i(\boldsymbol{x}^*) + \boldsymbol{s}^T H_i(\boldsymbol{x}^*)\boldsymbol{s} \geq 0$$

for all $i \in A$. If we expand the function $f(\boldsymbol{x}(\alpha))$, we obtain

$$f(\boldsymbol{x}(\alpha)) = f(\boldsymbol{x}^* + \alpha \boldsymbol{s} + \frac{1}{2}\alpha^2 \boldsymbol{p} + O(\alpha^3))$$

$$\begin{aligned} &= f(\boldsymbol{x}^*) + (\alpha \boldsymbol{s} + \frac{1}{2}\alpha^2 \boldsymbol{p})^T \nabla f(\boldsymbol{x}^*) + \frac{1}{2}\alpha^2 \boldsymbol{s}^T H(\boldsymbol{x}^*)\boldsymbol{s} + O(\alpha^3)) \\ &= \alpha \boldsymbol{s}^T \nabla f(\boldsymbol{x}^*) + + \frac{1}{2}\alpha^2 \left(\boldsymbol{p}^T \nabla f(\boldsymbol{x}^*) + \boldsymbol{s}^T H(\boldsymbol{x}^*)\boldsymbol{s}\right) + O(\alpha^3)), \end{aligned}$$

where $H(\boldsymbol{x})$ denotes the Hessian of $f(\boldsymbol{x})$. Therefore $\boldsymbol{x}^*$ can only be a local minimum, if the set

$$S := \{\boldsymbol{s} | \boldsymbol{s}^T \nabla f(\boldsymbol{x}^*) < 0 \text{ and } \boldsymbol{s}^T \nabla g_i(\boldsymbol{x}^*) \geq 0 \text{ for } i \in A\}$$

is empty. Using now the Farkas Lemma below, the result follows. □

LEMMA 4.2.1 (FARKAS). *Let $\boldsymbol{u} \in \mathbb{R}^n$, $v_i \in \mathbb{R}^n$ for $i \in A$. Then the set*

$$S := \{\boldsymbol{s} | \boldsymbol{s}^T \boldsymbol{u} < 0 \text{ and } \boldsymbol{s}^T \boldsymbol{v}_i \geq 0 \text{ for } i \in A\}$$

is empty if and only if

$$\boldsymbol{u} = \sum_{i \in A} \lambda_i \boldsymbol{v}_i$$

for some $\lambda_i \geq 0$, for $i \in A$.

PROOF. See exercises. □

4.3 Unconstrained Optimization

For a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, we want to find a local minimum,

$$f(\boldsymbol{x}) \longrightarrow \min .$$

We have seen that mathematically, one simple searches for zeros of the gradient, and verifies if the Hessian is positive definite, in order to find local minima. However already finding a zero of a nonlinear system of equations is a highly non-trivial task, as we have seen in Chapter **??**, and the numerical methods can have severe convergence problems. In optimization, one tries immediately to use the fact that one searches for a minimum of a function, and one does not simply try to solve a linear system of equations given by $\nabla f(\boldsymbol{x}) = 0$. In general, these methods only converge to a local minimum of f, except if f satisfies certain stringent requirements.

4.3.1 Line Search Methods

Line search methods construct a sequence of iterates $\boldsymbol{x}_k$ by the formula

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k,$$

where $\boldsymbol{p}_k$ is a search direction, and one requires $\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) < 0$ to guarantee that it is a descent direction, and the scalar parameter $\alpha_k > 0$ is a step length.

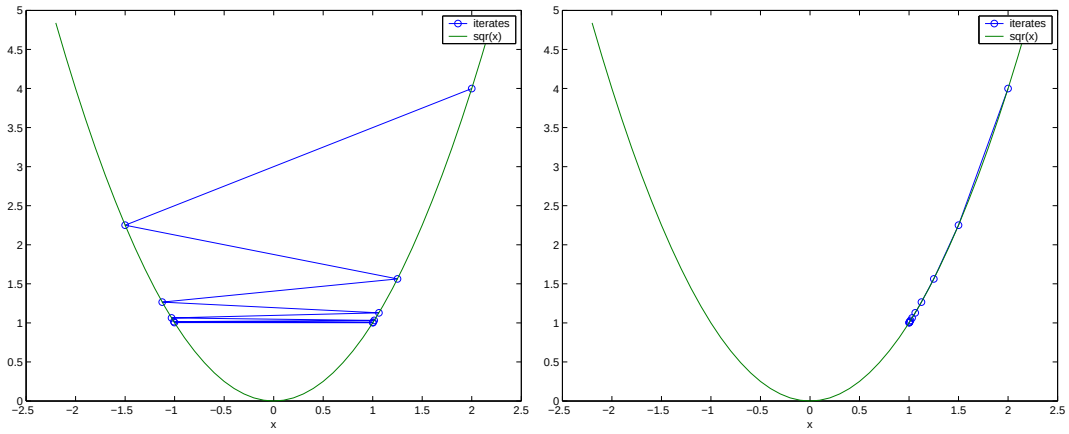


FIGURE 4.11.
On the left a case where the step length in the line search is too long, and on the right one where the step length is too short. In both cases, the line search methods do not converge to a minimum.

A first idea to determine the step length α_k is to simply minimize f in the search direction $\boldsymbol{p}_k$, which is a one dimensional minimization problem, i.e. we compute a zero of the derivative with respect to α,

$$\frac{d}{d\alpha} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) = \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) = 0.$$

We have seen this approach already in Chapter **??**, when we introduced the method of steepest descent to solve a symmetric positive definite system of linear equations. Unfortunately this method was not very good, and we introduced the method of conjugate gradients, a Krylov method, with much better performance. Conjugate gradients can be considered as an optimization algorithm for unconstrained problems of the form $f(\boldsymbol{x}) = \boldsymbol{x}^T A \boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x}$.

Minimizing with respect to α is called exact line search, and is in general far to expensive to perform. Today, there are many inexact line search methods, which try to determine an α_k at each step which does neither go too far, nor not far enough. This is critical, as one can see with the following two examples.

EXAMPLE 4.3.1. *Suppose we need to minimize $f(x) = x^2$ using a line search algorithm starting at $x_0 = 2$. If we choose the descent direction to be $p_k = (-1)^{k+1}$, and the step length $\alpha_k = 2 + \frac{3}{2^{k+1}}$, the result is shown in Figure 4.11 on the left, which was obtained with the Matlab commands*

```
x(1)=2;
for k=1:10
  x(k+1)=x(k)+(2+3/(2^k))*(-1)^k;
end
xx=-2.2:0.1:2.2;
plot(x,x.^2,'-o',xx,xx.^2,'-');
xlabel('x');
```

```
legend('iterates','sqr(x)');
```

where we needed to pay attention to the fact that array indices in Matlab always start at 1. Clearly the method fails, the step length is too big, and the method stagnates in a two cycle. The cycle can easily be computed using the MAPLE *recurrence relation solver,*

```
rsolve({x(k+1)=x(k)+(2+3/2^(k+1))*(-1)^(k+1),x(0)=2},{x});
```

$$\{\mathrm{x}(k) = (-1)^k + (\frac{-1}{2})^k\}$$

and thus shows convergence to an oscillation between 1 *and* -1.

EXAMPLE 4.3.2. *Suppose we need to minimize the same function* $f(x) = x^2$, *but now we choose the descent direction* $p_k = -1$, *and the step length* $\alpha_k = \frac{1}{2^{k+1}}$. *The result is shown in Figure 4.11 on the right, obtained with the Matlab commands*

```
x(1)=2;
for k=1:10
  x(k+1)=x(k)-1/(2^k);
end
xx=-2.2:0.1:2.2;
plot(x,x.^2,'-o',xx,xx.^2,'-');
xlabel('x');
legend('iterates','sqr(x)');
```

Again the method fails, the step length is too short in this case, and the method stagnates before reaching the minimum. Again the recurrence relation can be solved with MAPLE,

```
rsolve({x(k+1)=x(k)-1/2^(k+1),x(0)=2},{x});
```

$$\{\mathrm{x}(k) = 1 + (\frac{1}{2})^k\}$$

and thus shows convergence to 1. These two examples show clearly that it is very important to choose the line search parameter neither too large, nor too small.

Armijo Backtracking Line Search

Among the many line search techniques, one of the most successful ones is the Armijo backtracking line search, which starts with a reasonably big line

search parameter α_{init}, and then shortens it, until the function value at the new position is sufficiently smaller than the value at the old position:

ALGORITHM 4.1. *Generic line search method*

for $\boldsymbol{x}$ and $\boldsymbol{p}$ given;
$\alpha = \alpha_{\text{init}}$;
while $f(\boldsymbol{x} + \alpha\boldsymbol{p}) > f(\boldsymbol{x}) + \alpha\beta\boldsymbol{p}^T\nabla f(\boldsymbol{x})$
  $\alpha = \tau\alpha$;
end;

Here α_{init} is an initial guess for the line search parameter, for example $\alpha_{\text{init}} = 1$, β is a parameter to control the required degree of decrease of the function (note that $\boldsymbol{p}^T\nabla f(\boldsymbol{x}) < 0$ for a descent direction), a number between zero and one, and τ is the factor by which α is diminished each time, for example one half. The Armijo backtracking line search avoids too small steps, since the first step that gives the required decrease in the function value is accepted. Too big steps are also avoided, since one requires a sufficient decay in the function value. Naturally one might wonder if the Armijo backtracking strategy always finds a suitable value for the line search parameter α. The following theorem and its corollary give an affirmative answer.

THEOREM 4.5. *Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be continuously differentiable, and let ∇f be locally Lipschitz at $\boldsymbol{x}_k$ with Lipschitz constant $L(\boldsymbol{x}_k)$,*

$$||\nabla f(\boldsymbol{x}_k + \boldsymbol{x}) - \nabla f(\boldsymbol{x}_l)||_2 \leq L(\boldsymbol{x}_k)||\boldsymbol{x}||_2. \tag{4.25}$$

If $\beta \in (0,1)$ and $\boldsymbol{p}_k$ is a descent direction of f at $\boldsymbol{x}_k$, then the Armijo condition

$$f(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k) \leq f(\boldsymbol{x}_k) + \alpha\beta\boldsymbol{p}_k^T\nabla f(\boldsymbol{x}_k) \tag{4.26}$$

is satisfied for all $\alpha \in [0, \alpha_{\max}(\boldsymbol{x}_k, \boldsymbol{p}_k)]$, where

$$\alpha_{\max}(\boldsymbol{x}_k, \boldsymbol{p}_k) = \frac{2(\beta - 1)\boldsymbol{p}_k^T\nabla f(\boldsymbol{x}_k)}{L(\boldsymbol{x}_k)||\boldsymbol{p}_k||_2^2}. \tag{4.27}$$

PROOF. Using integration, we find

$$\begin{aligned}
f(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k) - f(\boldsymbol{x}_k) - \alpha\boldsymbol{p}_k^T\nabla f(\boldsymbol{x}_k) &= \int_0^1 \alpha\boldsymbol{p}_k^T\nabla f(\boldsymbol{x}_k + \tau\alpha\boldsymbol{p}_k)d\tau - \int_0^1 \alpha\boldsymbol{p}_k^T\nabla f(\boldsymbol{x}_k)d\tau \\
&= \alpha\int_0^1 \boldsymbol{p}_k^T(\nabla f(\boldsymbol{x}_k + \tau\alpha\boldsymbol{p}_k) - \nabla f(\boldsymbol{x}_k))d\tau \\
&\leq \alpha||\boldsymbol{p}_k||_2\int_0^1 ||\nabla f(\boldsymbol{x}_k + \tau\alpha\boldsymbol{p}_k) - \nabla f(\boldsymbol{x}_k))||_2 d\tau \\
&\leq \alpha^2||\boldsymbol{p}_k||_2^2 L(\boldsymbol{x}_k)\int_0^1 \tau d\tau \\
&= \frac{\alpha^2}{2}L(\boldsymbol{x}_k)||\boldsymbol{p}_k||_2^2,
\end{aligned}$$

where we used the Cauchy-Schwarz inequality for vectors in the first inequality, and the Lipschitz condition (4.25) in the second one. Now using that $\alpha \leq \alpha_{\max}$, we can replace one factor α in the α^2 on the right hand side to obtain

$$\begin{aligned} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) &\leq f(\boldsymbol{x}_k) + \alpha \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) + \alpha(\beta - 1)\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) \\ &= f(\boldsymbol{x}_k) + \alpha\beta \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k), \end{aligned}$$

which concludes the proof. □

COROLLARY 4.3.1. *Under the conditions of the previous theorem, the backtracking Armijo line search terminates with*

$$\alpha_k \geq \min(\alpha_{init}, \frac{2\tau(\beta - 1)\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)}{L(\boldsymbol{x}_k)||\boldsymbol{p}_k||_2^2}). \tag{4.28}$$

PROOF. Either α_{init} already satisfies the Armijo condition, or there is a second to last step in the Armijo backtracking algorithm which does not yet satisfy the Armijo condition, and hence the next step will multiply this second to last one by τ, which then satisfies the Armijo condition, and the algorithm stops with α_k satisfying (4.28). □

In general, it would be very difficult to get an estimate of the local Lipschitz constant $L(\boldsymbol{x}_k)$, and the backtracking Armijo search is precisely a tool to find a suitable line search parameter without knowing this quantity.

Generic Line Search Method with Armijo Backtracking Line Search

A generic minimization procedure with the Armijo backtracking line search is

ALGORITHM 4.2.
Generic minimization with Armijo line search

```
x = x_0;
k = 0;
while not converged
   find a suitable descent direction p;
   α = α_init;
   while f(x + αp) > f(x) + αβp^T ∇f(x)
      α = τα;
   end;
   x = x + αp;
   k = k + 1;
end;
```

THEOREM 4.6. *Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be continuously differentiable, and let ∇f be locally Lipschitz at $\boldsymbol{x}_k$ with Lipschitz constant $L(\boldsymbol{x}_k)$. Then the generic minimization algorithm with backtracking Armijo line search leads to one of the following results:*

1. *$\nabla f(\boldsymbol{x}_k) = 0$ for some $k \geq 0$.*

2. $\lim_{k\to\infty} f(\boldsymbol{x}_k) = -\infty$.

3. $\lim_{k\to\infty} \min(|\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)|, \frac{|\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)|}{||\boldsymbol{p}_k||_2}) = 0$.

PROOF. Case 1 is a lucky hit of a stationary point, and case 2 can happen when the function is not bounded from below. Assuming that neither is the case, i.e. $\nabla f(\boldsymbol{x}_k) \neq 0$ for all $k \geq 0$, and $\lim_{k\to\infty} f(\boldsymbol{x}_k) > -\infty$, the Armijo condition gives for all $k \geq 0$

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k) \leq \alpha_k \beta \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k),$$

and summing over k, we obtain

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_0) \leq \sum_{j=0}^{k} \alpha_j \beta \boldsymbol{p}_j^T \nabla f(\boldsymbol{x}_j),$$

which shows that the sum remains bounded from below, since $f(\boldsymbol{x}_{k+1}) > -\infty$. Now all the terms in the sum are negative, since the $\boldsymbol{p}_j$ are descent directions, and therefore the terms must converge to zero,

$$\lim_{k\to\infty} \alpha_k \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) = 0.$$

We now define the two sets

$$K_1 := \{k | \alpha_{\text{init}} > \frac{2\tau(\beta-1)\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)}{L(\boldsymbol{x}_k)||\boldsymbol{p}_k||_2^2}, \quad K_2 := \{k | \alpha_{\text{init}} \leq \frac{2\tau(\beta-1)\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)}{L(\boldsymbol{x}_k)||\boldsymbol{p}_k||_2^2}.$$

If $k \in K_1$, we have from the corollary

$$\alpha_k \geq \frac{2\tau(\beta-1)\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)}{L(\boldsymbol{x}_k)||\boldsymbol{p}_k||_2^2}$$

and therefore

$$\alpha_k \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) \leq \frac{2\tau(\beta-1)}{L(\boldsymbol{x}_k)} \left(\frac{\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)}{||\boldsymbol{p}_k||_2}\right)^2 < 0,$$

which implies

$$\lim_{k\in K_1\to\infty} \frac{|\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)|}{||\boldsymbol{p}_k||_2} = 0.$$

For $k \in K_2$, we have from the corollary

$$\alpha_k \geq \alpha_{\text{init}},$$

which implies

$$\lim_{k\in K_2\to\infty} |\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)| = 0,$$

and hence concludes the proof. □

This first convergence result does unfortunately not imply that the algorithm converges to a stationary point where the gradient vanishes, since in case 3, the gradient can simply become more and more orthogonal to the search direction. For a successful algorithm, one therefore needs to demand more of the search direction than simply that it is a descent direction.

Steepest Descent

The classical choice is to go into the direction of steepest descent, i.e. $\boldsymbol{p}_k := -\nabla f(\boldsymbol{x}_k)$, which leads to a convergent algorithm:

COROLLARY 4.3.2. *Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be continuously differentiable, and let ∇f be locally Lipschitz at $\boldsymbol{x}_k$ with Lipschitz constant $L(\boldsymbol{x}_k)$. Then the generic minimization algorithm with backtracking Armijo line search and steepest descent search direction leads to one of the following results:*

1. $\nabla f(\boldsymbol{x}_k) = 0$ *for some* $k \geq 0$.
2. $\lim_{k\to\infty} f(\boldsymbol{x}_k) = -\infty$.
3. $\lim_{k\to\infty} \nabla f(\boldsymbol{x}_k) = \mathbf{0}$.

PROOF. The two special cases 1 and 2 are as in Theorem 4.6. For case 3, we obtain from Theorem 4.6 that

$$\lim_{k\to\infty} \min(|\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)|, \frac{|\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)|}{||\boldsymbol{p}_k||_2}) = \lim_{k\to\infty} \min(||\nabla f(\boldsymbol{x}_k)||_2^2, ||\nabla f(\boldsymbol{x}_k)|_2) = 0,$$

which concludes the proof. □

Here is a Matlab implementation of the optimization algorithm with Armijo line search and steepest descent direction:

```
function [x,xk]=SteepestDescent(f,fp,x0,tol,maxiter,tau,be,alinit)
% STEEPESTDESCENT steepest descent minimum search with Armijo line search
%   [x,xk]=SteepestDescent(f,fp,x0,tol,maxiter,tau,be,alinit) finds an
%   approximate minimum of the function f with gradient fp, starting
%   at the initial guess x0. The remaining parameters are optional and
%   default values are used if they are omited. xk contains all the
%   iterates of the method.

if nargin<8, alinit=1; end;
if nargin<7, be=0.1; end;
if nargin<6, tau=0.5; end;
if nargin<5, maxiter=100; end;
if nargin<4, tol=1e-6; end;

x=x0;
xk=x0;
p=-feval(fp,x);
```

```
k=0;
while norm(p)>tol & k<maxiter
  al=alinit;
  while feval(f,x+al*p)>feval(f,x)-al*be*p'*p
    al=tau*al;
  end;
  x=x+al*p
  p=-feval(fp,x);
  k=k+1;
  xk(:,k+1)=x;
end;
```

As we have seen in the Chapter **??** on linear equations, steepest descent is not necessarily the best choice, and the same holds true for optimization, as the example in Figure 4.12 shows, which was generated using the Matlab commands

```
f=inline('10*(x(2)-x(1)^2)^2+(x(1)-1)^2','x');
fp=inline('[-40*(x(2)-x(1)^2)*x(1)+2*(x(1)-1); 20*(x(2)-x(1)^2)]','x');
F=inline('10*(y-x.^2).^2+(x-1).^2','x','y');   % for plotting purposes
Fp1=inline('-40*(y-x.^2).*x+2*(x-1)','x','y');
Fp2=inline('20*(y-x.^2)','x','y');
[x,xk]=SteepestDescent(f,fp,[-1.2;1])
[X,Y]=meshgrid(-1.5:0.1:1,-0.5:0.1:1.5);
contour(X,Y,F(X,Y),50)
hold on
quiver(X,Y,Fp1(X,Y),Fp2(X,Y),5)
plot(xk(1,:),xk(2,:),'-o')
hold off
```

However, even though there are better choices, as we will see now, many modern methods resort to the steepest descent choice if the method detects convergence problems.

Newton Search Direction

To obtain a better search direction, we first notice that for any symmetric positive definite matrix B_k, the search direction

$$\boldsymbol{p}_k := -B_k^{-1}\nabla f(\boldsymbol{x}_k)$$

is a descent direction, since

$$\boldsymbol{p}_k^T\nabla f(\boldsymbol{x}_k) = -(\nabla f(\boldsymbol{x}_k))^T B_k^{-T}\nabla f(\boldsymbol{x}_k) > 0,$$

because B_k positive definite implies B_k^{-T} to be positive definite as well. In addition this particular search direction leads directly to a stationary point of the quadratic problem

$$g(\boldsymbol{x}_k + \boldsymbol{p}) := f(\boldsymbol{x}_k) + \boldsymbol{p}^T\nabla f(\boldsymbol{x}_k) + \boldsymbol{p}^T B_k\boldsymbol{p},$$

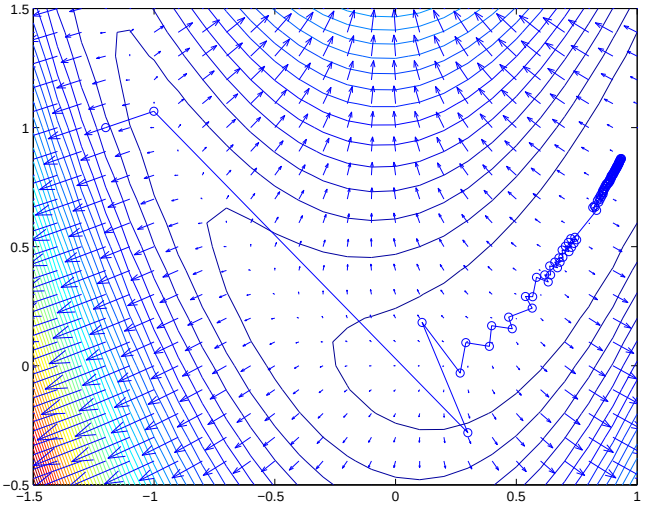


FIGURE 4.12.
Steepest Descent method for a model problem showing convergence problems of the method.

which can be considered as a quadratic approximation of the function f in the neighborhood of $\boldsymbol{x}_k$. In particular, if $B_k = H(\boldsymbol{x}_k)$, the Hessian of f at $\boldsymbol{x}_k$, then g is nothing else than a second order Taylor approximation of f about $\boldsymbol{x}_k$. In this case, one calls the search direction $\boldsymbol{p}_k = -H(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k)$ the Newton direction, because this is the step the classical Newton method described in Chapter **??** would take trying to find a root of the equation $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$. In optimization, this direction is however only useful, if the Hessian $H(\boldsymbol{x}_k)$ is positive definite, since otherwise, the Newton direction could well be an ascent direction.

THEOREM 4.7. *If $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is continuously differentiable, and ∇f is Lipschitz in $\mathbb{R}^n$, the generic minimization algorithm with the backtracking Armijo line search and descent direction $\boldsymbol{p}_k = -B_k\nabla f(\boldsymbol{x}_k)$, where B_k are symmetric positive definite matrices with $\lambda_{\max}(B_k) \leq \lambda_{\max} < \infty$ and $\lambda_{\min}(B_k) \geq \lambda_{\min} > 0$ for all k leads to one of the following results:*

1. $\nabla f(\boldsymbol{x}_k) = 0$ *for some* $k \geq 0$.
2. $\lim_{k\to\infty} f(\boldsymbol{x}_k) = -\infty$.
3. $\lim_{k\to\infty} \nabla f(\boldsymbol{x}_k) = 0$.

PROOF. The points 1 and 2 are as in Theorem 4.6. For 3, we have for all $\boldsymbol{x} \neq 0$ that

$$\lambda_{\min} \leq \lambda_{\min}(B_k) \leq \frac{\boldsymbol{x}^T B_k \boldsymbol{x}}{||\boldsymbol{x}||_2} \leq \lambda_{\max}(B_k) \leq \lambda_{\max},$$

and therefore the two estimates

$$|\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)| = |\nabla f(\boldsymbol{x}_k)^T B_k^{-T} \nabla f(\boldsymbol{x}_k)| \geq \lambda_{\min}(B_k^{-1})||\nabla f(\boldsymbol{x}_k)||_2^2 \geq \frac{1}{\lambda_{\max}}||\nabla f(\boldsymbol{x}_k)||_2^2$$

and

$$\begin{aligned} ||\boldsymbol{p}_k||_2^2 &= \nabla f(\boldsymbol{x}_k)^T B_k^{-T} B_k^{-1} \nabla f(\boldsymbol{x}_k) \\ &= \nabla f(\boldsymbol{x}_k)^T B_k^{-2} \nabla f(\boldsymbol{x}_k) \\ &\leq \lambda_{\max}(B_k^{-2}) ||\nabla f(\boldsymbol{x}_k)||_2^2 \\ &\leq \frac{1}{\lambda_{\min}^2} ||\nabla f(\boldsymbol{x}_k)||_2^2 \end{aligned}$$

hold. Using them together leads to

$$\frac{|\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)|}{||\boldsymbol{p}_k||_2} \geq \frac{\lambda_{\min}}{\lambda_{\max}} ||\nabla f(\boldsymbol{x}_k)||_2.$$

Therefore, the important quantity converging to zero in Theorem 4.6 is an upper bound,

$$\min(|\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)|, \frac{|\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)|}{||\boldsymbol{p}_k||_2}) \geq \frac{1}{\lambda_{\max}} ||\nabla f(\boldsymbol{x}_k)||_2 \min(||\nabla f(\boldsymbol{x}_k)||_2, \lambda_{\min}),$$

and since $\lambda_{\min} > 0$ and $\lambda_{\max}$ is finite, $||\nabla f(\boldsymbol{x}_k)||_2$ must converge to zero as k goes to infinity, which concludes the proof. □

Here is a Matlab implementation of the Armijo backtracking line search using the Newton search direction

```
function [x,xk]=Newton(f,fp,fpp,x0,tol,maxiter,tau,be,alinit)
% NEWTON Minimization with Newton descent and Armijo line search
%   [x,xk]=Newton(f,fp,fpp,x0,tol,maxiter,tau,be,alinit) finds an
%   approximate minimum of the function f with gradient fp and Hessian
%   fpp, starting at the initial guess x0. The remaining parameters are
%   optional and default values are used if they are omited. xk
%   contains all the iterates of the method.

if nargin<9, alinit=1; end;
if nargin<8, be=0.1; end;
if nargin<7, tau=0.5; end;
if nargin<6, maxiter=100; end;
if nargin<5, tol=1e-6; end;

x=x0;
xk=x0;
p=-feval(fpp,x)\feval(fp,x);
k=0;
while norm(feval(fp,x))>tol & k<maxiter
  al=alinit;
%  while feval(f,x+al*p)>feval(f,x)-al*be*p'*p
%    al=tau*al;
```

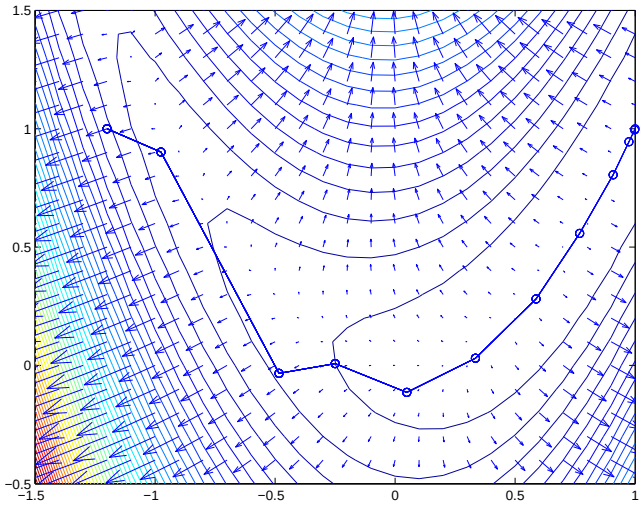


FIGURE 4.13.
Backtracking Armijo line search with Newton search direction for a model problem showing much better convergence behavior than the steepest descent direction from Figure 4.12.

```
%  end;
  x=x+al*p
  p=-feval(fpp,x)\feval(fp,x);
  k=k+1;
  xk(:,k+1)=x;
end;
```

For the same example as in Figure 4.12, we show in Figure 4.13 how much better the Newton step performs on this problem. The following theorem gives a convergence estimate for the backtracking Armijo line search with Newton direction.

THEOREM 4.8. *Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be twice continuously differentiable, and its Hessian matrix $H : \mathbb{R}^n \longrightarrow \mathbb{R}^{n\times n}$ be Lipschitz in $\mathbb{R}^n$ with Lipschitz constant L. If the sequence $\{\boldsymbol{x}_k\}$, $k = 1, 2, \ldots$ generated by the generic minimization algorithm with backtracking Armijo linesearch and $\alpha_{init} = 1$, $\beta \in (0, \frac{1}{2})$ and*

$$\boldsymbol{p}_k = \begin{cases} -H(\boldsymbol{x}_k)^{-1}\nabla f(\boldsymbol{x}_k) & \text{if } H(\boldsymbol{x}_k) \text{ is positive definite} \\ -B_k^{-1}\nabla f(\boldsymbol{x}_k) & B_k \text{ as in Theorem 4.7,} \end{cases}$$

has a limit point $\boldsymbol{x}^*$ with $H(\boldsymbol{x}^*)$ positive definite, then

1. *$\alpha_k = 1$ for k large,*
2. *the entire sequence $\{\boldsymbol{x}_k\}$ converges to $\boldsymbol{x}^*$, and*
3. *the convergence rate is quadratic, i.e.*

$$\lim_{k\to\infty} \frac{||\boldsymbol{x}_{k+1} - \boldsymbol{x}^*||_2}{||\boldsymbol{x}_{k+1} - \boldsymbol{x}^*||_2^2} \leq C, \quad C > 0 \text{ a constant.}$$

PROOF. We start by considering a subsequence $\{\boldsymbol{x}_{k'}\}$ which converges to $\boldsymbol{x}^*$. By continuity, $H(\boldsymbol{x}_{k'})$ is positive definite for k' large enough, and it exists a k'_0 such that for all $k' \geq k'_0$ we have

$$\boldsymbol{p}_{k'}^T H(\boldsymbol{x}_{k'})\boldsymbol{p}_{k'} \geq \frac{1}{2}\lambda_{\min}(H(\boldsymbol{x}^*))||\boldsymbol{p}_{k'}||_2^2. \tag{4.29}$$

This implies with the Newton step $H(\boldsymbol{x}_{k'})\boldsymbol{p}_{k'} = -\nabla f(\boldsymbol{x}_{k'})$

$$|\boldsymbol{p}_{k'}^T \nabla f(\boldsymbol{x}_{k'})| = -\boldsymbol{p}_{k'}^T \nabla f(\boldsymbol{x}_{k'}) = \boldsymbol{p}_{k'}^T H(\boldsymbol{x}_{k'})\boldsymbol{p}_{k'} \geq \frac{1}{2}\lambda_{\min}(H(\boldsymbol{x}^*))||\boldsymbol{p}_{k'}||_2^2, \tag{4.30}$$

and $\lim_{k'\to\infty} \boldsymbol{p}_{k'} = 0$ by Theorem 4.8. Now from Taylor's Theorem with integral remainder term, we obtain

$$\begin{aligned} f(\boldsymbol{x}_k + \boldsymbol{p}_k) &= f(\boldsymbol{x}_k) + \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) + \int_0^1 (1-t)\boldsymbol{p}_k^T H(\boldsymbol{x}_k + t\boldsymbol{p}_k)\boldsymbol{p}_k dt \\ &= f(\boldsymbol{x}_k) + \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) + \boldsymbol{p}_k^T H(\boldsymbol{x}_k + \tau\boldsymbol{p}_k)\boldsymbol{p}_k \int_0^1 (1-t)dt \quad 0 \leq \tau \leq 1 \\ &= f(\boldsymbol{x}_k) + \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2}\boldsymbol{p}_k^T H(\boldsymbol{x}_k + \tau\boldsymbol{p}_k)\boldsymbol{p}_k \end{aligned}$$

and we can therefore estimate

$$\begin{aligned} f(\boldsymbol{x}_k + \boldsymbol{p}_k) - f(\boldsymbol{x}_k) - \frac{1}{2}\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) &= \frac{1}{2}\left(\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) + \boldsymbol{p}_k^T H(\boldsymbol{x}_k + \tau\boldsymbol{p}_k)\boldsymbol{p}_k\right) \\ &= \frac{1}{2}\left(\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) + \boldsymbol{p}_k^T H(\boldsymbol{x}_k)\boldsymbol{p}_k\right) \\ &+ \frac{1}{2}\left(\boldsymbol{p}_k^T (H(\boldsymbol{x}_k + \tau\boldsymbol{p}_k) - H(\boldsymbol{x}_k))\boldsymbol{p}_k\right), \end{aligned}$$

and since $H(\boldsymbol{x}_k)\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k)$, the first term on the right hand side is identically zero. Using the Lipschitz condition $H(\boldsymbol{x}_k)$ satisfies, and that $0 \leq \tau \leq 1$, we therefore obtain

$$f(\boldsymbol{x}_k + \boldsymbol{p}_k) - f(\boldsymbol{x}_k) - \frac{1}{2}\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) \leq \frac{1}{2}L||\boldsymbol{p}_k||_2^3. \tag{4.31}$$

Hence choosing k' large enough, such that

$$L||\boldsymbol{p}_{k'}||_2 \leq \frac{1}{2}\lambda_{\min}(H(\boldsymbol{x}^*))(1-2\beta),$$

we obtain first using (4.31) and then (4.29) that

$$\begin{aligned} f(\boldsymbol{x}_{k'} + \boldsymbol{p}_{k'}) - f(\boldsymbol{x}_{k'}) &\leq \frac{1}{2}\boldsymbol{p}_{k'}^T \nabla f(\boldsymbol{x}_{k'}) + \frac{1}{4}\lambda_{\min}(H(\boldsymbol{x}^*))(1-2\beta)||\boldsymbol{p}_{k'}||_2^2 \\ &\leq \frac{1}{2}(1-(1-2\beta))\boldsymbol{p}_{k'}^T \nabla f(\boldsymbol{x}_{k'}) = \beta \boldsymbol{p}_{k'}^T \nabla f(\boldsymbol{x}_{k'}), \end{aligned}$$

which shows that for k' large enough, the Armijo condition is satisfied with $\alpha = 1$. In addition, we have for k' large enough $||H(\boldsymbol{x}_k)^{-1}||_2 \leq \frac{2}{\lambda_{\min}(H(\boldsymbol{x}^*))}$, and

$$\begin{aligned}
\boldsymbol{x}_{k'+1} - \boldsymbol{x}^* &= \boldsymbol{x}_{k'} - H(\boldsymbol{x}_{k'})^{-1}\nabla f(\boldsymbol{x}_{k'}) - \boldsymbol{x}^* \\
&= \boldsymbol{x}_{k'} - H(\boldsymbol{x}_{k'})^{-1}(\nabla f(\boldsymbol{x}_{k'}) - \nabla f(\boldsymbol{x}^*)) - \boldsymbol{x}^* \quad (\text{note } \nabla f(\boldsymbol{x}^*) = 0) \\
&= H(\boldsymbol{x}_{k'})^{-1}(\nabla f(\boldsymbol{x}_{k'}) - \nabla f(\boldsymbol{x}^*) - H(\boldsymbol{x}_{k'})(\boldsymbol{x}^* - \boldsymbol{x}_{k'})),
\end{aligned}$$

which allows us to estimate the difference in norm,

$$\begin{aligned}
||\boldsymbol{x}_{k'+1} - \boldsymbol{x}^*||_2 &\leq ||H(\boldsymbol{x}_{k'})^{-1}||_2||(\nabla f(\boldsymbol{x}_{k'}) - \nabla||f(\boldsymbol{x}^*) - H(\boldsymbol{x}_{k'})(\boldsymbol{x}^* - \boldsymbol{x}_{k'}))||_2 \\
&= ||H(\boldsymbol{x}_{k'})^{-1}||_2||\int_0^1 H(\boldsymbol{x}_{k'} + \tau(\boldsymbol{x}^* - \boldsymbol{x}_{k'}))(\boldsymbol{x}^* - \boldsymbol{x}_{k'})d\tau - H(\boldsymbol{x}_{k'})(\boldsymbol{x}^* - \boldsymbol{x}_{k'})||_2 \\
&\leq ||H(\boldsymbol{x}_{k'})^{-1}||_2\int_0^1 ||H(\boldsymbol{x}_{k'} + \tau(\boldsymbol{x}^* - \boldsymbol{x}_{k'})) - H(\boldsymbol{x}_{k'})||_2||(\boldsymbol{x}^* - \boldsymbol{x}_{k'})||_2 d\tau \\
&\leq ||H(\boldsymbol{x}_{k'})^{-1}||_2\frac{1}{2}L||(\boldsymbol{x}^* - \boldsymbol{x}_{k'})||_2^2,
\end{aligned}$$

and hence for k' large enough, we have

$$\frac{||\boldsymbol{x}_{k'+1} - \boldsymbol{x}^*||_2}{||(\boldsymbol{x}^* - \boldsymbol{x}_{k'})||_2^2} \leq \frac{L}{2}||H(\boldsymbol{x}_{k'})^{-1}||_2 \leq \frac{L}{\lambda_{\min}(H(\boldsymbol{x}^*))}$$

which proves item 3 for $k = k'$ with $C = \frac{L}{\lambda_{\min}(H(\boldsymbol{x}^*))}$. In addition, this implies that for k' large enough, $k' + 1$ is also an index in the convergent subsequence, which proves item 2, and thus item 1 and 3 for all k large enough. □

Modified Newton Methods

In modified Newton methods, one replaces the Newton search direction determined by

$$H(\boldsymbol{x}_k)\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k)$$

by a modified search direction determined by

$$(H(\boldsymbol{x}_k) + M_k)\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k),$$

where the matrix M_k is chosen to guarantee that the matrix $H(\boldsymbol{x}_k) + M_k$ is sufficiently positive definite. There are several possibilities to obtain such a matrix M_k:

1. If one has an eigendecomposition of the Hessian, $H(\boldsymbol{x}_k) = Q_k^T\Lambda_k Q_k$, one can choose

   $$H(\boldsymbol{x}_k) + M_k = Q_k^T \max(\varepsilon I, \Lambda_k)Q_k,$$

   i.e. one shifts the eigenvalues which are too small or negative to $\varepsilon > 0$. The computation of the eigendecomposition can however be very expensive at each step.

2. If one has an estimate of the smallest eigenvalue of the Hessian, $\lambda_{\min}(H(\boldsymbol{x}_k))$, one can choose

$$M_k = \max(0, \varepsilon - \lambda_{\min}(H(\boldsymbol{x}_k))) \cdot I,$$

which shifts all the eigenvalues, if small or negative eigenvalues are present. This has the disadvantage that one large negative eigenvalue can distort the entire problem.

3. One can try to compute the Cholesky decomposition of the Hessian, which is needed anyway to solve the system defining the search direction $\boldsymbol{p}_k$, see Subsection **??**. If the Cholesky decomposition fails due to small or negative eigenvalues, there are techniques to adapt it to obtain a decomposition of the form

$$H(\boldsymbol{x}_k) + M_k = L_k L_k^T.$$

Quasi Newton Methods

Quasi Newton methods were developed in order to reduce the cost of computing the Hessian, which is nowadays less an issue. There are two main techniques:

1. Approximation of the Hessian by finite differences, i.e. for the i-th unit vector $\boldsymbol{e}_i$, one computes

$$H(\boldsymbol{x}_k)\boldsymbol{e}_i \approx \frac{\nabla f(\boldsymbol{x}_k + h\boldsymbol{e}_i) - \nabla f(\boldsymbol{x}_k)}{h}$$

and h is chosen approximately to be the square-root of `macheps` to balance roundoff and approximation errors.

2. Using a secant approximation: one searches for a matrix B_k which satisfies the so called secant condition,

$$B_{k+1}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k),$$

which is motivated by the fact that the real Hessian matrix satisfies the secant condition approximately by the Taylor theorem. There is a lot of freedom to determine a matrix B_k satisfying the secant condition. Two of the most successful ones are

Symmetric rank one method: Starting with a given matrix B_0, one computes recursively the rank one updates

$$B_{k+1} = B_k + \frac{(\boldsymbol{y}_k - B_k\boldsymbol{s}_k)(\boldsymbol{y}_k - B_k\boldsymbol{s}_k)^T}{\boldsymbol{s}_k^T(\boldsymbol{y}_k - B_k\boldsymbol{s}_k)},$$

where $\boldsymbol{y}_k := \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$ and $\boldsymbol{s}_k := \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$. This approach can unfortunately lead to indefinite matrices B_k and even fail with a division by zero.

Broyden-Fletcher-Goldfarb-Shanno (BFGS) method: One computes recursively

$$B_{k+1} = B_k + \frac{\boldsymbol{y}\boldsymbol{y}^T}{\boldsymbol{s}_k\boldsymbol{y}_k} - \frac{B_k\boldsymbol{s}_k\boldsymbol{s}_k^T B_k}{\boldsymbol{s}_k^T B_k \boldsymbol{s}_k}.$$

It is easy to show (see the exercises) that if one starts with a matrix B_0 which is symmetric and positive definite, then all the B_k are symmetric and positive definite, provided the $\boldsymbol{s}_k^T\boldsymbol{y}_k$ are positive. This latter condition can be guaranteed by a special line search strategy due to Goldstein.

In general, the finite difference approximations of the Hessian in 1 are more expensive than the secant condition updates in 2.

Truncated Newton Methods

Truncated Newton methods are especially useful if the problem is very high dimensional. In that case, solving for the Newton direction using the linear system

$$H(\boldsymbol{x}_k)\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k)$$

can be prohibitively expensive. The idea of truncated Newton methods is to solve the linear system approximately using the conjugate gradient method, see Section 3.13. If one starts the conjugate gradient method with the zero vector, the first iteration gives an approximate search direction $\boldsymbol{p}_k$ with is identical to steepest descent. If one iterates with the conjugate gradient method to convergence, one obtains the Newton search direction. Stopping anywhere in between gives an approximate search direction between steepest descent and the Newton direction.

4.3.2 Trust Region Methods

Trust region methods construct a local model $m(\boldsymbol{s})$ of the function $f(\boldsymbol{x}_k + \boldsymbol{s})$ and then accept a step $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$, if $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ gives a similar decrease compared to the decrease predicted by the local model $m(\boldsymbol{s}_k)$. The two mostly used models are

Linear model: $m_l(\boldsymbol{s}) := f(\boldsymbol{x}_k) + \boldsymbol{s}^T\nabla f(\boldsymbol{x}_k)$

Quadratic model: $m_q(\boldsymbol{s}) := f(\boldsymbol{x}_k) + \boldsymbol{s}^T\nabla f(\boldsymbol{x}_k) + \frac{1}{2}\boldsymbol{s}^T B_k\boldsymbol{s}$, where the matrix B_k is an approximation to the Hessian at $\boldsymbol{x}_k$.

Since these models can not be precise for $\boldsymbol{s}$ big (the linear model is not even bounded from below), one imposes in addition the so called trust-region constraint,

$$||\boldsymbol{s}|| \leq R_k, \tag{4.32}$$

and one searches at each step of the trust region algorithm

$$\min_{||\boldsymbol{s}|| \leq R_k} m(\boldsymbol{s}).$$

A generic trust region algorithm is of the form

ALGORITHM 4.3. *Generic trust region algorithm*

For given x_0 and R_0;
$k = 0$;
while not converged
  $\boldsymbol{s}_k = \underset{||\boldsymbol{s}|| \leq R_k}{\operatorname{argmin}} m(\boldsymbol{s})$;
  $\rho = (f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{s}_k))/(f(\boldsymbol{x}_k) - m(\boldsymbol{s}_k))$;
  if $\rho \geq \rho_s$
    $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$;
    if $\rho \geq \rho_v$
      $R_{k+1} = \gamma_i R_k$;
    else
      $R_{k+1} = R_k$;
    end;
  else
    $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$;
    $R_{k+1} = \gamma_d R_k$; end;
  k=k+1;
end

After solving the trust region subproblem, the algorithm computes ρ, the ratio of how much decrease was achieved, to how much was predicted. If this ration is bigger than ρ_s, e.g. $\rho_i = 0.1$, a threshold for success, then the step is accepted. If the ratio is even bigger than ρ_v, $0 < \rho_s < \rho_v < 1$, e.g. $\rho_v = 0.9$, then the step was very successful, and we increase the trust region by the factor $\gamma_i \geq 1$, for example $\gamma_i = 2$. Otherwise, the size of the trust region is kept the same. If however the step was not a success, then we stay at the present position and decrease the trust region size by the factor γ_d, e.g. $\gamma_d = 0.5$.

One can show under certain hypotheses that the trust region algorithm 4.3 leads like the line search algorithm to one of the following results:

1. $\nabla f(\boldsymbol{x}_k) = 0$ for some $k \geq 0$.
2. $\lim_{k\to\infty} f(\boldsymbol{x}_k) = -\infty$.
3. $\lim_{k\to\infty} \nabla f(\boldsymbol{x}_k) = 0$.

To solve the trust region problem at each step, one can simply use an analytical solution in the case of the linear model, whereas for the quadratic model, there exist elegant numerical solutions.

Here is a Matlab implementation of the trust region algorithm with a linear model:

```
function [x,xk,Rk]=TrustRegionLinear(f,fp,x0,tol,maxiter,R0,rs,rv,gi,gd)
```

```
% TRUSTREGIONLINEAR trust region method with linear model
%   [x,xk,Rk]=TrustRegionLinear(f,fp,x0,tol,maxiter,r0,rs,rv,gi,gd)
%   finds an approximate minimum of the function f with gradient fp,
%   starting at the initial guess x0. The remaining parameters are
%   optional and default values are used if they are omited. xk
%   contains all the iterates of the method and rk the trust region
%   radii of all iterations.

if nargin<10, gd=0.5; end;
if nargin<9, gi=2; end;
if nargin<8, rv=0.9; end;
if nargin<7, rs=0.1; end;
if nargin<6, R0=1; end;
if nargin<5, maxiter=100; end;
if nargin<4, tol=1e-6; end;

x=x0;
xk=x0;
R=R0;
Rk=R0;
p=-feval(fp,x);
k=0;
while norm(p)>tol & k<maxiter
  s=R*p/norm(p);
  r=(feval(f,x)-feval(f,x+s))/(s'*p);
  if r>=rs
    x=x+s;
    p=-feval(fp,x);
    if r>-rv
      R=gi*R;
    end
  else
    R=gd*R
  end
  k=k+1;
  xk(:,k+1)=x;
  Rk(:,k+1)=R;
end;
```

The results for the same model problem as in Figure 4.12 are shown in Figure 4.14 on the left. We observe that the trust region method with a linear model has a similar behaviour to a line search method with steepest descent: the search can slow down dramatically in the bottom of a valley. On the right in Figure 4.14, we show for the first three steps how the trust region was adjusted to find an acceptable step.

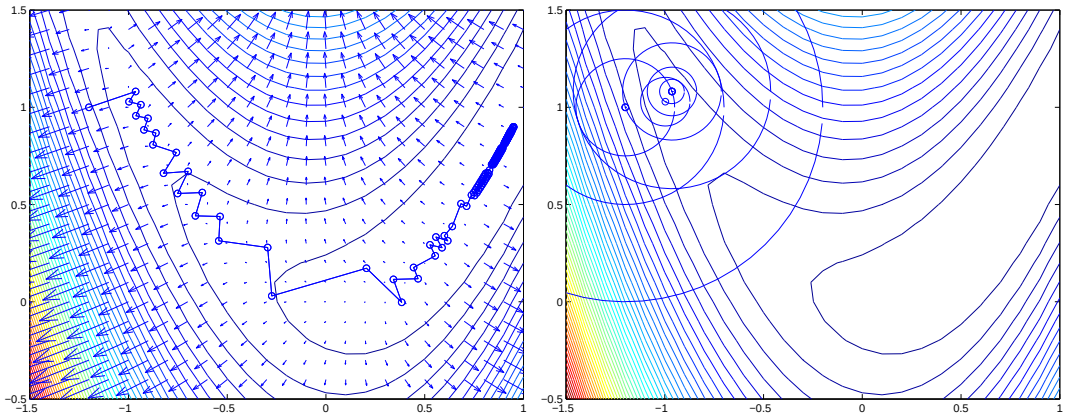


FIGURE 4.14.
Trust region method with a linear model for an example, showing that the method with the linear model has similar problems as line search with steepest descent.

4.3.3 Direct Methods

Direct optimization methods are methods which do not use derivative information, neither explicitly nor implicitly. We have seen an example of such a method in the introduction, the algorithm using the golden section to find the minimum of a scalar function, analogously to bisection to find the zero of a scalar function.

Probably the most successful direct method is the Nelder Mead algorithm, invented by two engineers in 1965. This algorithm uses a simplex, or a triangle in two dimensions, to explore the function $f : \mathbb{R}^n \to \mathbb{R}$ to be minimized. To start the algorithm, an initial simplex given by its corner points $\boldsymbol{x}_j$, $j = 1, 2, \ldots, n+1$ is needed. The first step in each iteration of the algorithm is to sort the corners such that $\boldsymbol{x}_1$ is the corner with the smallest, and $\boldsymbol{x}_{n+1}$ is the corner with the largest function value. Then the algorithm computes the center of gravity on the hypersurface described by all but $\boldsymbol{x}_{n+1}$,

$$\bar{\boldsymbol{x}} = \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j,$$

and the reflected point

$$\boldsymbol{x}_r = \bar{\boldsymbol{x}} + \rho(\bar{\boldsymbol{x}} - \boldsymbol{x}_{n+1}),$$

where ρ is a parameter, usually $\rho = 1$. An example for $n = 2$ is shown in Figure 4.15. Now the algorithm takes a first decision: if $f(\boldsymbol{x}_r) < f(\boldsymbol{x}_1)$, the direction $\boldsymbol{x}_r$ seems very promising, and the expansion point

$$\boldsymbol{x}_e = \bar{\boldsymbol{x}} + \chi(\boldsymbol{x}_r - \bar{\boldsymbol{x}})$$

is computed, with the parameter $\chi = 2$ usually, as shown in Figure 4.15. Now if $f(\boldsymbol{x}_e) < f(\boldsymbol{x}_r)$, then the expansion is accepted, and $\boldsymbol{x}_e$ replaces the worst point

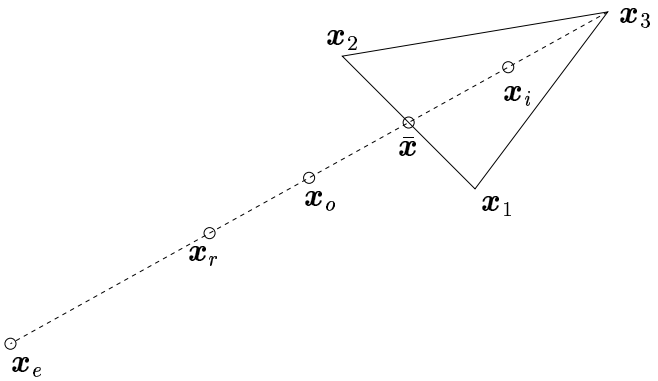


FIGURE 4.15.
The center of gravity $\bar{x}$, the reflection point x_r, the expansion point x_e and the two contraction points x_i and x_o, computed by the Nelder Mead algorithm.

x_{n+1} to obtain a new simplex, for the next iteration. Otherwise, if $f(x_r) < f(x_n)$, then the reflected point x_r is accepted and replaces the worst point x_{n+1} to form a new simplex for the next iteration. Finally, if $f(x_r) \geq f(x_n)$, the reflected point is not such a good direction, and a contraction step is tried. To do so, an inner and outer contraction point is computed,

$$x_i = \bar{x} - \gamma(x_r - \bar{x}), \quad x_o = \bar{x} + \gamma(x_r - \bar{x}),$$

with the usual parameter choice $\gamma = \frac{1}{2}$. This leads to the two points shown in Figure 4.15. Now if $f(x_r) < f(x_{n+1})$ and $f(x_o) < f(x_r)$, then x_o replaces x_{n+1} for a new simplex, and if $f(x_r) \geq f(x_{n+1})$ and $f(x_i) < f(x_{n+1})$, then x_i replaces x_{n+1} for a new simplex to restart the iteration. If neither of these conditions holds, a shrinking step is performed, in which all the vertices get closer to x_1, i.e.

$$x_j = x_1 + \sigma(x_j - x_1), \quad j = 2, \ldots, n+1,$$

with the parameter σ usually chosen to be $\frac{1}{2}$.

Here is a MATLAB implementation of the Nelder Mead algorithm:

```
function x=NelderMead(f,x0,tol,maxiter,r,c,g,s)
% NELDERMEAD direct minimization algorithm by Nelder Mead
%   x=NelderMead(f,x0,tol,maxiter,r,c,g,s) tries to find a minimum of
%   f, starting at x0. The remaining parameters are optional, and
%   defaults are chosen, if they are not given.

if nargin<8, s=0.5; end;
if nargin<7, g=0.5; end;
if nargin<6, c=2; end;
if nargin<5, r=1; end;
if nargin<4, maxiter=100; end;
```

```
if nargin<3, tol=1e-6; end;

x=x0;
k=0;
fk=feval(f,x);
line([x(:,1);x(1,1)],[x(:,2);x(1,2)])
pause
while k<maxiter
  [fk,id]=sort(fk);
  x=x(id,:);
  xb=mean(x(1:end-1,:));
  xr=xb+r*(xb-x(end,:));
  fr=feval(f,xr);
  if fr>=fk(1) & fr<fk(end-1)                % reflection step
    x(end,:)=xr; fk(end)=fr;
  elseif fr<fk(1)
    xe=xb+c*(xr-xb);
    fe=feval(f,xe);
    if fe<fr
      x(end,:)=xe; fk(end)=fe;
    else
      x(end,:)=xr; fk(end)=fr;
    end;
  else                                       % contraction step
    xi=xb-g*(xr-xb); fi=feval(f,xi);
    xo=xb+g*(xr-xb); fo=feval(f,xo);
    if fr<fk(end) & fo<fr
      x(end,:)=xo; fk(end)=fo;
    elseif fr>=fk(end) & fi<fk(end)
      x(end,:)=xi; fk(end)=fi;
    else                                     % shrinking step
      for j=2:size(x,1)
        x(j,:)=x(1,:)+s*(x(j,:)-x(1,:));
        fk(j)=feval(f,x(j,:));
      end;
    end;
  end;
  k=k+1;
  line([x(:,1);x(1,1)],[x(:,2);x(1,2)])
  pause
end;
```

Figure 4.16 shows how the Nelder Mead algorithm approaches the minimum in the model problem of the previous sections. It is impressive how well the

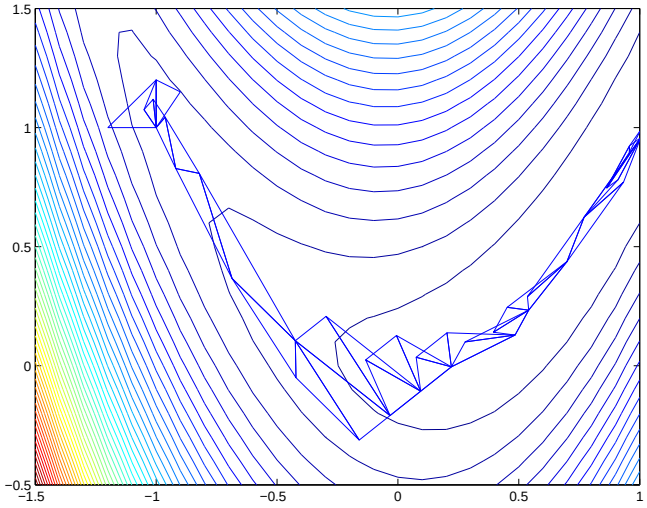


FIGURE 4.16. *Nelder Mead algorithm for an example.*

algorithm works on the model problem, and it has proved to work very well for innumerable applications. There is however now mathematical foundation for the Nelder Mead algorithm: the only convergence result known is for strictly convex functions in one (!) dimension (see Wright et al., 1998). For two dimensional problems, there exist examples due to McKinnon (1998), that even in the strictly convex case, the Nelder Mead algorithm can fail and converge to a degenerate simplex which does not indicate a minimum of the function. One does not even know if there exists a function in two dimensions for which Nelder Mead works for any initial guess. It currently remains a mystery why Nelder Mead is so successful for so many practical problems.

4.4 Constrained Optimization

4.4.1 Linear Programming

Linear programming refers to constraint optimization problems, where both the objective function and the equality and inequality constraints in the general optimization problem (4.13) are linear (or more precisely affine). After elimination of the equality constraints, we obtain the so called linear program

$$\begin{array}{ccccccccc} \tilde{c}_1\tilde{x}_1 & + & \tilde{c}_2\tilde{x}_2 & + & \dots & + & \tilde{c}_{\tilde{n}}\tilde{x}_{\tilde{n}} & \longrightarrow & \min, \\ \tilde{a}_{11}\tilde{x}_1 & + & \tilde{a}_{12}\tilde{x}_2 & + & \dots & + & \tilde{a}_{1\tilde{n}}\tilde{x}_{\tilde{n}} & \leq & b_1, \\ \tilde{a}_{21}\tilde{x}_1 & + & \tilde{a}_{22}\tilde{x}_2 & + & \dots & + & \tilde{a}_{2\tilde{n}}\tilde{x}_{\tilde{n}} & \leq & b_2, \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ \tilde{a}_{m1}\tilde{x}_1 & + & \tilde{a}_{m2}\tilde{x}_2 & + & \dots & + & \tilde{a}_{m\tilde{n}}\tilde{x}_{\tilde{n}} & \leq & b_m, \\ & & & & & & \tilde{x}_i & \geq & 0, \end{array} \tag{4.33}$$

which one can write in more compact matrix notation,

$$\begin{array}{rcl} \tilde{\boldsymbol{c}}^T\tilde{\boldsymbol{x}} & \longrightarrow & \min, \\ \tilde{A}\tilde{\boldsymbol{x}} & \leq & \boldsymbol{b}, \\ \tilde{\boldsymbol{x}} & \geq & 0. \end{array} \tag{4.34}$$

If some of the $\tilde{x}_i$ do not satisfy a sign constraint, one can artificially introduce two new variables $\tilde{x}_i^+ \geq 0$ and $\tilde{x}_i^- \geq 0$ and decompose $\tilde{x}_i = \tilde{x}_i^+ - \tilde{x}_i^-$ to fit the framework above. Danzig invented in 1951 an elegant algorithm, the simplex algorithm, to solve problems of the form (4.34), which led to a real boom and the formation of entire operations research departments at universities. We now derive the simplex algorithm and illustrate it with the model problem from the introduction, subsection 4.1.3 on operations research.

The first step is the introduction of slack variables y_i, to transform the inequality constraints in (4.34) into equality constraints. This is achieved by setting $\boldsymbol{y} = \boldsymbol{b} - A\tilde{\boldsymbol{x}}$, and the inequality constraint in (4.34) shows that the slack variables satisfy $y_i \geq 0$. This leads to the new equivalent problem

$$\begin{array}{rcl} \tilde{\boldsymbol{c}}^T\tilde{\boldsymbol{x}} & \longrightarrow & \min, \\ \tilde{A}\tilde{\boldsymbol{x}} + \boldsymbol{y} & = & \boldsymbol{b}, \\ \tilde{\boldsymbol{x}} & \geq & 0, \\ \boldsymbol{y} & \geq & 0. \end{array} \tag{4.35}$$

It is worthwhile to interpret this problem before continuing: if one of the slack variables is zero, the corresponding underlying inequality constraints in (4.33) holds with equality, which implies that we are on the boundary of the region of admissible values of $\tilde{\boldsymbol{x}}$, compare Figure 4.7 for the example in the introduction. In this example, one would have four slack variables, and setting one of them equal to zero gives one of the four lines which form the boundaries of the region of admissible solution points. Two more lines are obtained by looking at $\tilde{x}_i = 0$ from the inequality constraints on $\tilde{x}_i$. Hence the slack variables and the real variables play a very similar role: setting one equal to zero gives a hyperplane which can be part of the boundary of the set of admissible solution points. If we combine (4.35) the two vectors $\tilde{\boldsymbol{x}} \in \mathbb{R}^{\tilde{n}}$ and $\boldsymbol{y} \in \mathbb{R}^m$ into the larger vector $\boldsymbol{x} \in \mathbb{R}^n$, $n = \tilde{n} + m$, extend $\tilde{\boldsymbol{c}}$ by zeros to obtain $\boldsymbol{c} \in \mathbb{R}^n$, and form a new matrix $A \in \mathbb{R}^{m \times n}$ from $\tilde{A}$ and the identity matrix of size $m \times m$, we obtain in the new, more compact notation

$$\begin{array}{rcl} \boldsymbol{c}^T\boldsymbol{x} & \longrightarrow & \min, \\ A\boldsymbol{x} & = & \boldsymbol{b}, \\ \boldsymbol{x} & \geq & 0. \end{array} \tag{4.36}$$

Again setting any of the components x_i equal to zero means we are on a hyperplane which can be part of the boundary of the set of admissible solutions, and setting $\tilde{n}$ components equal to zero leads to a corner, provided the hyperplanes all intersect, which means the remaining matrix after deleting the corresponding columns in A is invertible and thus the linear system in (4.36) has a solution after the corresponding x_i have been set to zero. Furthermore if this solution is

non-negative, it is a corner on the boundary of the set of admissible solutions, and thus a candidate for the solution of the optimization problem.

For our model problem (4.12) from Subsection 4.1.3, we need to introduce four slack variables: x_3, x_4, x_5 and x_6. The problem then becomes

$$\begin{array}{rcl} (5,4,0,0,0,0)^T\boldsymbol{x} & \rightarrow & \min, \\ \left[\begin{array}{cccccc} 1 & 1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]\boldsymbol{x} & = & \left(\begin{array}{c} 8 \\ -5 \\ 4 \\ 7 \end{array}\right), \\ x_i & \geq & 0. \end{array} \tag{4.37}$$

Choosing for example $x_3 = x_5 = 0$ leads to an invertible linear system for the remaining variables,

$$\left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right]\left(\begin{array}{c} x_1 \\ x_2 \\ x_4 \\ x_6 \end{array}\right) = \left(\begin{array}{c} 8 \\ -5 \\ 4 \\ 7 \end{array}\right), \tag{4.38}$$

whose solution is $(x_1, x_2, x_4, x_6) = (4,4,3,3) \geq 0$, and hence is the right upper corner $(4,4)$ on the Figure 4.7, the intersection of the hyperplanes $x_3 = x_5 = 0$.

Without loss of generality, we assume in the sequel that A has full rank, since otherwise there is either no solution, or one can eliminate equations from the linear system in (4.36) without changing the solutions. Motivated by the previous interpretation of (4.36) when setting some components equal to zero, we introduce the partitioning of the columns of the matrix A into two parts using the index sets $B = \{b_1, \ldots, b_m\}$ and $R = \{r_1, \ldots, r_{n-m}\}$, $B \cup R = \{1, 2, \ldots, n\}$, $B \cap R = \emptyset$, and partition the matrix A column wise into A_B and A_R,

$$A = \left[\begin{array}{cccc} A_1 & A_2 & \ldots & A_n \end{array}\right],\ A_B = \left[\begin{array}{cccc} A_{b_1} & A_{b_2} & \ldots & A_{b_m} \end{array}\right],\ A_R = \left[\begin{array}{cccc} A_{r_1} & A_{r_2} & \ldots & A_{r_{n-m}} \end{array}\right],$$

and similarly we also partition the vectors,

$$\boldsymbol{x}_B = \left(\begin{array}{c} x_{b_1} \\ x_{b_2} \\ \vdots \\ x_{b_m} \end{array}\right), \quad \boldsymbol{x}_R = \left(\begin{array}{c} x_{r_1} \\ x_{r_2} \\ \vdots \\ x_{r_{n-m}} \end{array}\right), \quad \boldsymbol{c}_B = \left(\begin{array}{c} c_{b_1} \\ c_{b_2} \\ \vdots \\ c_{b_m} \end{array}\right), \quad \boldsymbol{c}_R = \left(\begin{array}{c} c_{r_1} \\ c_{r_2} \\ \vdots \\ c_{r_{n-m}} \end{array}\right).$$

In this new notation, the linear program (4.36) becomes

$$\begin{array}{rcl} \boldsymbol{c}_B^T\boldsymbol{x}_B + \boldsymbol{c}_R^T\boldsymbol{x}_R & \longrightarrow & \min, \\ A_B\boldsymbol{x}_B + A_R\boldsymbol{x}_R & = & \boldsymbol{b}, \\ \boldsymbol{x}_B & \geq & 0, \\ \boldsymbol{x}_R & \geq & 0, \end{array} \tag{4.39}$$

and setting $\boldsymbol{x}_R$ equal to zero corresponds to a corner of the set of admissible solutions, provided that A_B is invertible, and $A_B^{-1}\boldsymbol{b} \geq 0$. This observation motivates the following definition:

DEFINITION 4.4.1 (BASIS). *The set B is a basis, if A_B is invertible, and the associated vector*

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_B \\ \boldsymbol{x}_R \end{pmatrix} = \begin{pmatrix} A_B^{-1}\boldsymbol{b} \\ \boldsymbol{0} \end{pmatrix}$$

is called basis solution. It is an admissible basis solution, if $\boldsymbol{x} \geq 0$.

If the matrix A_B is invertible, we can eliminate the unknowns $\boldsymbol{x}_B$ from the problem (4.39), using the relation $\boldsymbol{x}_B = A_B^{-1}(\boldsymbol{b} - A_R\boldsymbol{x}_R)$, which leads to the new problem

$$\begin{array}{rcl} \boldsymbol{u}^T\boldsymbol{x}_R + z & \longrightarrow & \min, \\ S\boldsymbol{x}_R & \leq & \boldsymbol{t}, \\ \boldsymbol{x}_R & \geq & 0, \end{array} \tag{4.40}$$

where $\boldsymbol{u}^T = \boldsymbol{c}_R^T - \boldsymbol{c}_B^T A_B^{-1} A_R$, $z = \boldsymbol{c}_B^T A_B^{-1}\boldsymbol{b}$, $S = A_B^{-1}A_R$ and $\boldsymbol{t} = A_B^{-1}\boldsymbol{b}$, and Danzig collected all the relevant quantities in the simplex table

	R	
B	S	$\boldsymbol{t}$
	$\boldsymbol{u}^T$	z

(4.41)

THEOREM 4.9 (SIMPLEX CRITERION). *Let B be a basis of the linear program (4.36) and let (4.41) be the associated simplex table. If $\boldsymbol{t} \geq 0$ and $\boldsymbol{u} \geq 0$, then $\boldsymbol{x}_B = \boldsymbol{t}$ and $\boldsymbol{x}_R = 0$ is the solution of the linear program (4.36).*

PROOF. The basis B is admissible, since $\boldsymbol{t} = A_B^{-1}\boldsymbol{b} \geq 0$. In addition, $\boldsymbol{u} \geq 0$ for a point $\boldsymbol{x}$ which satisfies $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} \geq 0$ implies that

$$\boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{u}^T\boldsymbol{x}_R + z \geq z = \boldsymbol{c}^T \begin{pmatrix} \boldsymbol{t} \\ \boldsymbol{0} \end{pmatrix},$$

which shows that there is no other solution with smaller objective function, and hence concludes the proof. □

The simplex algorithm by Danzig is a directed search along the edges of the set of admissible solutions to find the corner which minimizes the objective function. It starts with an admissible basis solution and its associated simplex table, and then constructs iteratively new admissible basis solutions and simplex tables with smaller and smaller values of the objective function, until the simplex criterion is satisfied. To transform one simplex table into another one, the following theorem is useful.

THEOREM 4.10. *Let B be a basis, $R = \{1, \ldots, n\} \backslash B$, $k \in B$ and $l \in R$, with the associated simplex table*

	R	
B	S	$\boldsymbol{t}$
	$\boldsymbol{u}^T$	z

$=$

	l	j	
k	s_{kl}	s_{kj}	t_k
i	s_{il}	s_{kj}	t_i
	u_l	u_j	z

$i \in B\backslash\{k\},\ j \in R\backslash\{l\}.$

(4.42)

If the so called pivot $s_{k,l} \neq 0$, then

1. *$\tilde{B} := B \cup \{l\} \backslash \{k\}$ is also a basis,*
2. *the corresponding new simplex table is*

$$\begin{array}{|c|c|c|}\hline & \tilde{R} & \\ \hline \tilde{B} & \tilde{S} & \tilde{\boldsymbol{t}} \\ \hline & \tilde{\boldsymbol{u}}^T & \tilde{z} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline & k & j & \\ \hline l & \frac{1}{s_{kl}} & \frac{s_{kj}}{s_{kl}} & \frac{t_k}{s_{kl}} \\ \hline i & -\frac{s_{il}}{s_{kl}} & s_{ij} - \frac{s_{kj}s_{il}}{s_{kl}} & t_i - \frac{t_k s_{il}}{s_{kl}} \\ \hline & -\frac{u_l}{s_{kl}} & u_j - \frac{s_{kj}u_l}{s_{kl}} & z - \frac{t_k u_l}{s_{kl}} \\ \hline \end{array}. \tag{4.43}$$

PROOF. The k-th component of the equation $\boldsymbol{x}_B + S\boldsymbol{x}_R = \boldsymbol{t}$ is

$$x_k + \sum_{j \in R} s_{kj} x_j = t_k.$$

Since the pivot $s_{kl} \neq 0$, we can solve this equation for x_l for $l \in R$, and obtain

$$x_l + \frac{1}{s_{kl}} x_k + \sum_{j \in R \backslash \{l\}} \frac{s_{kj}}{s_{kl}} x_j = \frac{t_k}{s_{kl}}, \tag{4.44}$$

which corresponds to the l-th line in the new simplex table (4.43). Replacing now x_l from (4.44) into the i-th component of the equation $\boldsymbol{x}_B + S\boldsymbol{x}_R = \boldsymbol{t}$, $i \in B \backslash \{k\}$, we obtain

$$x_i + s_{il} \left(-\frac{1}{s_{kl}} x_k - \sum_{j \in R \backslash \{l\}} \frac{s_{kj}}{s_{kl}} x_j + \frac{t_k}{s_{kl}} \right) + \sum_{j \in R \backslash \{l\}} s_{ij} x_j = t_i,$$

which becomes after rearranging

$$x_i - \frac{s_{il}}{s_{kl}} x_k + \sum_{j \in R \backslash \{l\}} s_{ij} - \frac{s_{il} s_{kj}}{s_{kl}} x_j = t_i - \frac{t_k s_{il}}{s_{kl}},$$

which corresponds the the i-th line in the new simplex table (4.43). Similarly, one also obtains the last line of the new simplex table, which concludes the proof. □

Theorem 4.10 allows us to easily compute a new simplex table from a previous one. The simplex algorithm avoids the computation of simplex tables of little use:

```
while there exists l in R with u(l)<0
  if s(i,l)<=0 for all i in B
    error('no solution, the objective function is unbounded');
  else
    choose k in B such that t(k)/s(k,l) is minimal for k in B with s(k,l)>0
    exchange k and l using the Theorem
  end;
end;
```

We explain each line in the simplex algorithm separately:

- The algorithm stops as soon as the simplex criterion from Theorem 4.9 is obtained, $\boldsymbol{u} \geq 0$, which means the solution has been found.

- If for some $l \in R$, we have $u(l) < 0$, and if $s_{il} \leq 0$ for all $i \in B$, then the objective function is unbounded from below on the set of admissible points, as one can see as follows: let $\boldsymbol{x}_R(\alpha)$ be defined by $x_l(\alpha) = \alpha$ for $\alpha \geq 0$, and $x_j(\alpha) = 0$ for $j \in R \backslash \{l\}$. Then $\boldsymbol{x}_R(\alpha) \geq 0$ and

$$S\boldsymbol{x}_R(\alpha) = \alpha \begin{pmatrix} s_{1l} \\ s_{2l} \\ \vdots \\ s_{ml} \end{pmatrix} \leq \mathbf{0} \leq \boldsymbol{t}.$$

  Hence the points $\boldsymbol{x}_R(\alpha)$ are admissible for all $\alpha \geq 0$. The objective function can then however be made arbitrarily large negative, since

$$\boldsymbol{u}^T \boldsymbol{x}_R(\alpha) - z = u_l \alpha - z \longrightarrow -\infty, \quad \text{for } \alpha \longrightarrow \infty,$$

  and hence there is no minimum.

- The particular choice of l and k in the algorithm leads to a new admissible simplex table, $\tilde{\boldsymbol{t}} \geq 0$, with improved objective $-\tilde{z} \leq -z$, as one can see as follows: the new components of $\boldsymbol{t}$ for $i \in B \backslash \{k\}$ are $\tilde{t}_i = t_i - \frac{t_k s_{il}}{s_{kl}}$ which is bigger than or equal to $t_i > 0$, if $s_{il} \leq 0$, since $t_k \geq 0$ and $s_{kl} > 0$. If $s_{il} > 0$, then $\tilde{t}_i = s_{il}(\frac{t_i}{s_{il}} - \frac{t_k}{s_{kl}}) \geq 0$, because k was chosen such that the ratio was smallest in the algorithm.

  Now for the objective, we have $-\tilde{z} = -z + \frac{t_k u_l}{s_{kl}} \leq -z$, since $u_l < 0$, $t_k \geq 0$ and $s_{kl} > 0$, and thus the new objective is at least as good or better than the old one.

Here is an implementation of the simplex algorithm in MATLAB:

```
function R=Simplex(B,R,S,t,u,z);
% SIMPLEX Simplex algorithm starting with a simplex table
%   R=Simplex(B,R,S,t,u,z); implements the simplex algorithm starting
%   with a given simplex table consisting of B,R,S,t,u and z and
%   finds the optimal solution.

l=find(u<0);
while length(l)>0
  l=l(1);
  i=find(S(:,l)>0);
  if length(i)<1
    error('solution is unbounded');
  else
```

```
    [d,k]=min(t(i)./S(i,l));k=i(k);
    t([1:k-1 k+1:end])=t([1:k-1 k+1:end])-t(k)*S([1:k-1 k+1:end],l)/S(k,l);
    z=z+t(k)*u(l)/S(k,l);
    t(k)=t(k)/S(k,l);
    u([1:l-1 l+1:end])=u([1:l-1 l+1:end])-S(k,[1:l-1 l+1:end])*u(l)/S(k,l);
    u(l)=-u(l)/S(k,l);
    S([1:k-1 k+1:end],[1:l-1 l+1:end])=S([1:k-1 k+1:end],[1:l-1 l+1:end])...
      -S([1:k-1 k+1:end],l)*S(k,[1:l-1 l+1:end])/S(k,l);
    S(k,[1:l-1 l+1:end])=S(k,[1:l-1 l+1:end])/S(k,l);
    S([1:k-1 k+1:end],l)=-S([1:k-1 k+1:end],l)/S(k,l);
    S(k,l)=1/S(k,l);
    d=R(l);R(l)=B(k);B(k)=d;
    l=find(u<0);
    [0 R 0
     B' S  t
     0 u  z]
    pause
  end;
end;
```

The three lines before the command `pause` display the simplex tables generated and are for illustrative purposes only. To see how the simplex algorithm works, we apply it to the model problem (4.12) from Subsection 4.1.3. From the reformulation in (4.37), we define in MATLAB

```
A=[ 1  1 1 0 0 0
   -1 -1 0 1 0 0
    1  0 0 0 1 0
    0  1 0 0 0 1];
b=[8;-5;4;7];
c=[5;4;0;0;0;0];
```

and to start with the admissible basis solution found in (4.38), we define the sets B and R to be

```
B=[1 2 4 6]; R=[3 5];
```

We can now compute the corresponding simplex table, take a look at it, and then start the simplex algorithm:

```
S=A(:,B)\A(:,R);
t=A(:,B)\b;
u=c(R)'-c(B)'*(A(:,B)\A(:,R));
z=c(B)'*(A(:,B)\b);
[ 0 R 0
  B' S t
  0 u z]
Simplex(B,R,S,t,u,z);
```

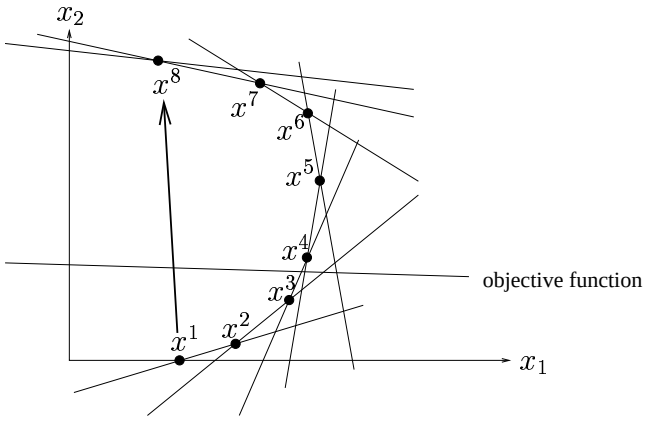


FIGURE 4.17.
An example where the simplex method could take many steps.

The algorithm computes from the starting simplex table two new ones and then stops:

```
0   3   5   0        0   4   5   0        0   4   1   0
1   0   1   4        1   0   1   4        5   0   1   4
2   1  -1   4        2  -1  -1   1        2  -1   1   5
4   1   0   3        3   1   0   3        3   1   0   3
6  -1   1   3        6   1   1   6        6   1  -1   2
0  -4  -1 -36        0   4  -1 -24        0   4   1 -20
```

In the first table, we see that the real

4.4.2 Penalty and Barrier Functions

The simplex algorithm is a very elegant method to march along the edges of the polytope described by the linear constraints toward the node with the smallest value of the objective function. The method can however be very slow, as illustrated in the example in Figure 4.17. Clearly it would be more effective to follow a more straight path, as indicated by the bold arrow. Karmarkar from the AT&T Bell Labs revolutionized linear programming in 1984, when he presented a so called interior point method which precisely followed a more straight path in the interior of the domain bounded by the constraints, instead of moving a long the boundary as the simplex algorithm does. A few years later, namely in 1989, AT&T Bell Labs offered a program by Korbk for sale for $ 8.9 Million. Very quickly public domain versions of such interior point methods followed, since their complexity is provably better than the worst case complexity of the simplex algorithm.

The fundamental idea of interior point methods is however not new: in an optimization problem with equality constraints,

$$\begin{aligned} f(\boldsymbol{x}) &\longrightarrow \min \\ \boldsymbol{c}_E(\boldsymbol{x}) &= 0, \end{aligned}$$

we can replace the constraint by a penalty function, and then solve the unconstrained optimization problem

$$\phi(\boldsymbol{x}) := f(\boldsymbol{x}) + \frac{1}{2\mu}||\boldsymbol{c}_E(\boldsymbol{x})||_2^2 \longrightarrow \min,$$

for smaller and smaller values of the parameter $\mu > 0$. The penalty function we used here is quadratic, but there are many other possibilities. One can show (see the exercises) that when μ converges to zero from above, some minima of ϕ converge to solutions of the constrained optimization problem. Unfortunately it is also possible that ϕ has other stationary points when μ converges to zero.

Similarly, in the optimization problem with inequality constraints

$$\begin{array}{rcl} f(\boldsymbol{x}) & \longrightarrow & \min \\ \boldsymbol{c}_I(\boldsymbol{x}) & \geq & 0, \end{array}$$

we can replace the constraint by a barrier function, and then solve the unconstrained optimization problem

$$\phi(\boldsymbol{x}) := f(\boldsymbol{x}) - \mu \sum_{i=1}^{m} \log((\boldsymbol{c}_I)_i(\boldsymbol{x})) \longrightarrow \min,$$

for smaller and smaller values of the parameter $\mu > 0$. The log barriar function is the one most often used. One can again show (see the exercises) that when μ converges to zero from above, some minima of ϕ converge to solutions of the constrained optimization problem, but again it is also possible that ϕ has other stationary points when μ converges to zero. Figure 4.18 shows two simple examples of the sequence of unconstrained problems one obtains with the log barrier function from a constrained one: on the left, we want to minimize $f(x) = x$ under the constraint $x \geq 0$, and on the right $f(\boldsymbol{x}) = x_1^2 + x_2^2$ under the constraint $x_1 + x_2^2 \geq 1$. One can clearly see in these two examples how the minimum of the unconstrained problem with the log barrier function approaches the minimum of the constraint problem. The graphics on the right in Figure 4.18 were obtained for various values of μ with the maple commands

```
f:=x1^2+x2^2;
phi:=f-mu*log(x1+x2^2-1);
mu:=1;P1:=plots[contourplot](phi,x1=0..2,x2=max(0,sign(1-x1^2)*sqrt(abs(1-x1^2)))..2
P2:=plot(sqrt(1-x1),x1=0..2):
plots[display](P1,P2);
```

4.4.3 Interior Point Methods

A generic interior point method uses a barrier function to convert a minimization problem with inequality constraints into a sequence of unconstraint minimization problems, which are then solved to get better and better approximations of the original problem:

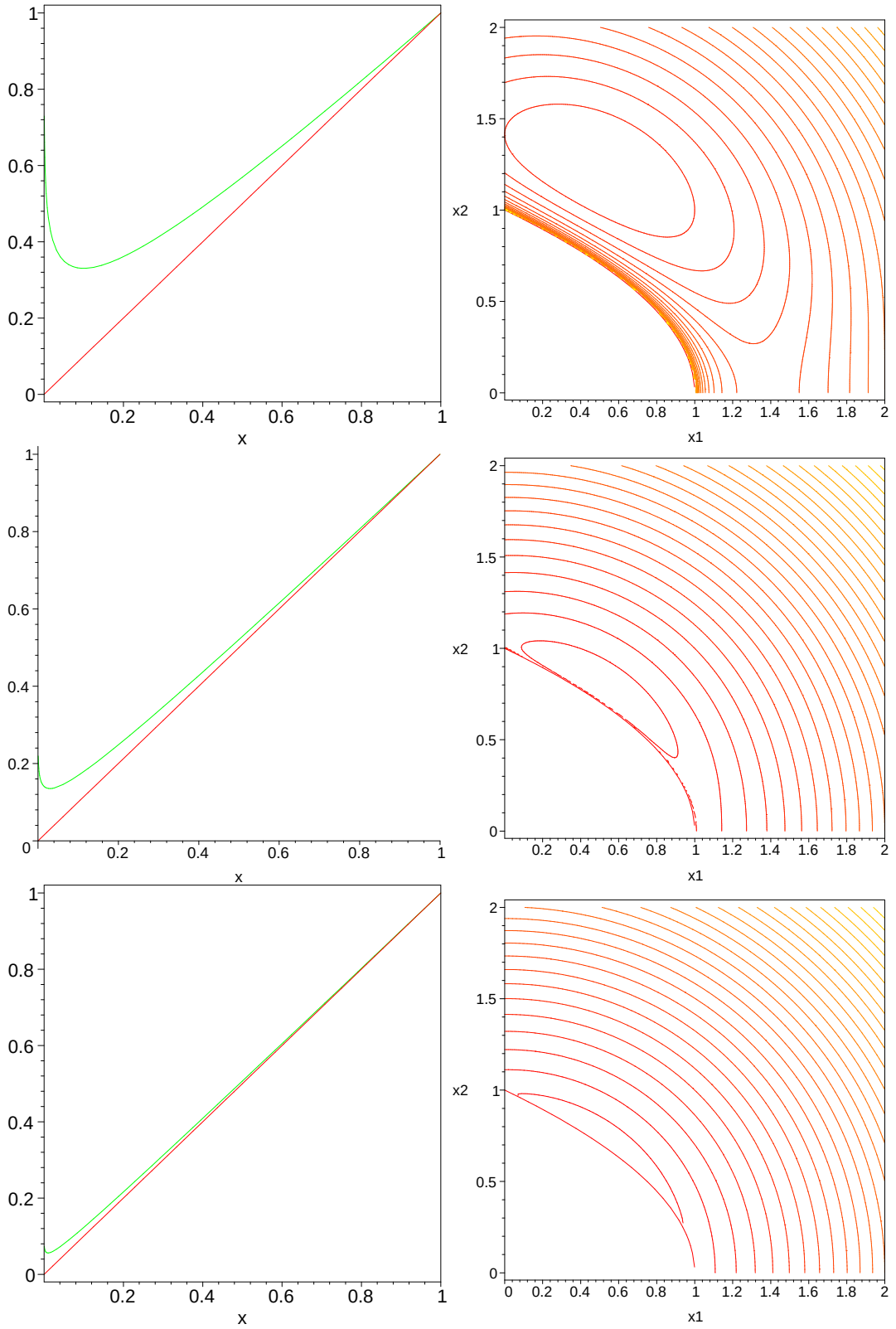


FIGURE 4.18.
Minimizing $f(x) = x$ under the constraint $x \geq 0$ using the log barrier function with $\mu = 0.1, 0.03, 0.01$ on the left, and $f(\boldsymbol{x}) = x_1^2 + x_2^2$ under the constraint $x_1 + x_2^2 \geq 1$ with the log barrier function and $\mu = 1, 0.1, 0.01$ on the right.

ALGORITHM 4.4. *Generic interior point method*

For given $\mu_0 > 0$;
$k = 0$;
while not converged
  find $\boldsymbol{x}_k^s$ with $\boldsymbol{c}(\boldsymbol{x}_k^s) > 0$; % e.g. $\boldsymbol{x}_k^s = \boldsymbol{x}_{k-1}^*$
  starting with $\boldsymbol{x}_k^s$, compute an approximate minimum
  $\boldsymbol{x}_k$ of $\phi(\boldsymbol{x}, \mu_k)$;
  compute $\mu_{k+1} > 0$ smaller than μ_k,
  s.t. $\lim_{k\to\infty} \mu_k = 0$; % e.g. $\mu_{k+1} = 0.1\mu_k$ or $\mu_{k+1} = \mu_k^2$
  k=k+1;
end

The following theorem shows that under suitable conditions, algorithm 4.4 produces a better and better approximation to a point satisfying the first order optimality conditions of the original problem, given in Theorem 4.4.

THEOREM 4.11. *For $f : \mathbb{R}^n \to \mathbb{R}$ and $\boldsymbol{g} : \boldsymbol{R}^n \to \mathbb{R}^m$ twice continuously differentiable, let*

$$(\lambda_k)_i := \frac{\mu_k}{g_i(\boldsymbol{x}_k)} \quad i = 1, 2, \ldots, m, \tag{4.45}$$

and assume that

$$||\nabla_x \phi(\boldsymbol{x}_k, \mu_k)||_2 \leq \epsilon_k, \tag{4.46}$$

where $\epsilon_k \to 0$ as $k \to \infty$, and that $\boldsymbol{x}_k \to \boldsymbol{x}$ with $\nabla g_i(\boldsymbol{x}^*)$ which are linearly independent for $i \in A$, the set of active constraints at $\boldsymbol{x}^*$. Then $\boldsymbol{x}^*$ satisfies the first order optimality conditions of Theorem 4.4, and $\lambda_k \to \lambda$, the associated Lagrange multipliers.

PROOF. Let I be the complement of A at $\boldsymbol{x}^*$, and let $G(\boldsymbol{x}) := \nabla \boldsymbol{g}(\boldsymbol{x})$, the Jacobian of $\boldsymbol{g}$, with $G_A(\boldsymbol{x})$ the lines $\nabla \boldsymbol{g}_i(\boldsymbol{x})$ with $i \in A$ and $G_I(\boldsymbol{x})$ the lines $\nabla \boldsymbol{g}_i(\boldsymbol{x})$ with $i \in I$. By the assumption that $G_A(\boldsymbol{x}^*)$ has full rank, we have that in a neighborhood of $\boldsymbol{x}^*$, the pseudo inverse

$$G_A^+(\boldsymbol{x}) := (G_A(\boldsymbol{x}) G_A(\boldsymbol{x})^T)^{-1} G_A(\boldsymbol{x})$$

is well defined. Now let $\boldsymbol{\lambda}_A := G_A^+(\boldsymbol{x}^*) \nabla f(\boldsymbol{x}^*)$ and $\boldsymbol{\lambda}_I := 0$. If the set I is not empty, using the definition (4.45), we can estimate

$$\begin{aligned} ||(\boldsymbol{\lambda}_k)_I||_2 &= \sqrt{\sum_{i\in I} \left(\frac{\mu_k}{g_i(\boldsymbol{x}_k)}\right)^2} \\ &\leq \mu_k \sqrt{\sum_{i\in I} \left(\frac{1}{\min_{j\in I} |g_j(\boldsymbol{x}_k)|}\right)^2} \\ &= \frac{\mu_k}{\min_{j\in I} |g_j(\boldsymbol{x}_k)|} \sqrt{|I|} \\ &\leq \frac{2\mu_k \sqrt{|I|}}{\min_{j\in I} |g_j(\boldsymbol{x}^*)|} \end{aligned} \tag{4.47}$$

for k large enough. Hence, we obtain together with assumption (4.46) that

$$\begin{aligned} ||\nabla f(\boldsymbol{x}_k) - G_A^T(\boldsymbol{\lambda}_k)_A||_2 &\leq ||\nabla f(\boldsymbol{x}_k) - G^T\boldsymbol{\lambda}_k||_2 + ||G_I^T(\boldsymbol{\lambda}_k)_I||_2 \\ &\leq \epsilon_k + \frac{2\mu_k\sqrt{|I|}}{\min_{j\in I}|g_j(\boldsymbol{x}^*)|} =: \tilde{\epsilon}_k. \end{aligned} \tag{4.48}$$

Now, since $G_A^+(\boldsymbol{x}_k)G_A(\boldsymbol{x}_k) = I$, we have

$$\begin{aligned} ||G_A^+(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k) - (\boldsymbol{\lambda}_k)_A||_2 &= ||G_A^+(\boldsymbol{x}_k)(\nabla f(\boldsymbol{x}_k) - G_A^T(\boldsymbol{x}_k)(\boldsymbol{\lambda}_k)_A||_2 \\ &\leq 2||G_A^+(\boldsymbol{x}^*)||_2\tilde{\epsilon}_k \end{aligned} \tag{4.49}$$

for k big enough. It therefore follows that

$$||\boldsymbol{\lambda}_A - (\boldsymbol{\lambda}_k)_A||_2 \leq ||G_A^+(\boldsymbol{x}^*)\nabla f(\boldsymbol{x}^*) - G_A^+(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k)|| + ||G_A^+(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k) - (\boldsymbol{\lambda}_k)_A||_2$$

and using (4.47) and (4.49) and the fact that $\boldsymbol{x}_k \to \boldsymbol{x}^*$, we obtain that $\boldsymbol{\lambda}_k \to \boldsymbol{\lambda}$ as $k \to \infty$. Furthermore, continuity of the gradients and (4.48) imply that

$$\nabla f(\boldsymbol{x}^*) - \nabla g^T(\boldsymbol{x}^*)\boldsymbol{\lambda} = 0,$$

and since $\boldsymbol{g}(\boldsymbol{x}_k) > 0$ for all k, we have $\boldsymbol{g}(\boldsymbol{x}^*) \geq 0$. Finally, the definition of $\boldsymbol{\lambda}_k$ in (4.45) gives

$$g_i(\boldsymbol{x}_k)(\boldsymbol{\lambda}_k)_i = \mu_k,$$

and with $\boldsymbol{\lambda}_k \to \boldsymbol{\lambda}$, we obtain

$$g_i(\boldsymbol{x}^*)\lambda_i = 0,$$

which concludes the proof. □

It is remarkable that the generic interior point method also computes an estimate for the Lagrange multipliers, as shown in the previous theorem. While this theorem establishes a basic convergence result for the generic interior point method, there are several issues that need to be addressed to make this a useful algorithm:

1. A fundamental problem in the algorithm is that the Hessian of the unconstrained problem to be solved in each iteration is becoming more and more ill-conditioned as the algorithm progresses. One can show that its condition number is proportional to $\frac{1}{\mu_k}$. The ill conditioning is evident for the example on the right in Figure 4.18, where the curvature is becoming extreme in the direction where the minimum is squeezed toward the boundary, whereas in the other direction, the curvature remains small.

2. Another important problem is the startin point $\boldsymbol{x}_k^s$. Using $\boldsymbol{x}_{k-1}^*$ can be a very unlucky choice, since one can show that from this point, a Newton step becomes asymptotically infeasible if $\mu_{k+1} < \frac{1}{2}\mu_k$.

These two problems can however be addressed using a perturbation argument in the optimality conditions, which leads to the powerful class of primal-dual interior point methods with both excellent theoretical and practical properties.

4.4.4 Sequential Quadratic Programming

Sequential quadratic programming (SQP) are methods to solve optimization problems with equality (or sometimes inequality) constraints based on Lagrange multipliers. The general minimization problem

$$\begin{array}{rcll} f(\boldsymbol{x}) & \rightarrow & \min & f:\mathbb{R}^n \rightarrow \mathbb{R}, \\ g(\boldsymbol{x}) & = & 0 & \boldsymbol{g}:\mathbb{R}^n \rightarrow \mathbb{R}^m, \end{array} \tag{4.50}$$

has the Lagrangian

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda}) = f(\boldsymbol{x}) - \boldsymbol{g}^T(\boldsymbol{x})\boldsymbol{\lambda}.$$

The system of first order optimality conditions

$$\begin{array}{rcl} \nabla_x \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda}) = \nabla f(\boldsymbol{x}) - (\nabla \boldsymbol{g}(\boldsymbol{x}))^T\boldsymbol{\lambda} & = & 0 \\ \boldsymbol{g}(\boldsymbol{x}) & = & 0 \end{array} \tag{4.51}$$

has $m+n$ equations for $m+n$ unknowns in $\boldsymbol{x}$ and $\boldsymbol{\lambda}$. Using Newtons method to compute an approximate solution of (4.51), we obtain the iteration

$$\begin{pmatrix} \boldsymbol{x}^{k+1} \\ \boldsymbol{\lambda}^{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}^{k} \\ \boldsymbol{\lambda}^{k} \end{pmatrix} - \begin{bmatrix} H(\boldsymbol{x}^k,\boldsymbol{\lambda}^k) & -\nabla \boldsymbol{g}(\boldsymbol{x}^k)^T \\ \nabla \boldsymbol{g}(\boldsymbol{x}^k) & 0 \end{bmatrix} \begin{pmatrix} \nabla_x \mathcal{L}(\boldsymbol{x}^k,\boldsymbol{\lambda}^k) \\ \boldsymbol{g}(\boldsymbol{x}^k) \end{pmatrix},$$

where $H(\boldsymbol{x},\boldsymbol{\lambda})$ denotes the Hessian with respect to $\boldsymbol{x}$ of the Lagrangian. Therefore, at each iteration of Newtons method, one has to solve the linear system of equations

$$\begin{bmatrix} H(\boldsymbol{x}^k,\boldsymbol{\lambda}^k) & -\nabla \boldsymbol{g}(\boldsymbol{x}^k)^T \\ \nabla \boldsymbol{g}(\boldsymbol{x}^k) & 0 \end{bmatrix} \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{\lambda} \end{pmatrix} = - \begin{pmatrix} \nabla_x \mathcal{L}(\boldsymbol{x}^k,\boldsymbol{\lambda}^k) \\ \boldsymbol{g}(\boldsymbol{x}^k) \end{pmatrix}.$$

This system can easily be symmetrized, by writing it in the form

$$\begin{bmatrix} H(\boldsymbol{x}^k,\boldsymbol{\lambda}^k) & \nabla \boldsymbol{g}(\boldsymbol{x}^k)^T \\ \nabla \boldsymbol{g}(\boldsymbol{x}^k) & 0 \end{bmatrix} \begin{pmatrix} \Delta\boldsymbol{x} \\ -\Delta\boldsymbol{\lambda} \end{pmatrix} = - \begin{pmatrix} \nabla_x \mathcal{L}(\boldsymbol{x}^k,\boldsymbol{\lambda}^k) \\ \boldsymbol{g}(\boldsymbol{x}^k) \end{pmatrix},$$

and hence one can use a Cholesky decomposition to solve it, if the matrix is positive definite, or a modification thereof, if the system is indefinite. If the system is large and sparse, one would use Conjugate Gradients in the positive definite case, and MINRES otherwise.

By linearity, we obtain from the first part of the system,

$$H(\boldsymbol{x}^k,\boldsymbol{\lambda}^k)\Delta\boldsymbol{x} - \nabla \boldsymbol{g}(\boldsymbol{x}^k)^T\Delta\boldsymbol{\lambda} = -\nabla_x \mathcal{L}(\boldsymbol{x}^k,\boldsymbol{\lambda}^k) = -\nabla f(\boldsymbol{x}^k) + (\nabla \boldsymbol{g}(\boldsymbol{x}^k))^T\boldsymbol{\lambda}^k,$$

an equivalent form,

$$H(\boldsymbol{x}^k,\boldsymbol{\lambda}^k)\Delta\boldsymbol{x} - (\nabla \boldsymbol{g}(\boldsymbol{x}^k))^T(\boldsymbol{\lambda}^k + \Delta\boldsymbol{\lambda}) = -\nabla f(\boldsymbol{x}^k),$$

and hence with the new variable $\bar{\boldsymbol{\lambda}} = \boldsymbol{\lambda}^k + \Delta\boldsymbol{\lambda}$, the system in Newtons method is also equivalent to

$$\begin{bmatrix} H(\boldsymbol{x}^k,\boldsymbol{\lambda}^k) & \nabla \boldsymbol{g}(\boldsymbol{x}^k)^T \\ \nabla \boldsymbol{g}(\boldsymbol{x}^k) & 0 \end{bmatrix} \begin{pmatrix} \Delta\boldsymbol{x} \\ -\bar{\boldsymbol{\lambda}} \end{pmatrix} = - \begin{pmatrix} \nabla f(\boldsymbol{x}^k) \\ \boldsymbol{g}(\boldsymbol{x}^k) \end{pmatrix}, \tag{4.52}$$

which will be useful later. Finally, one would often use an approximation B^k of the Hessian $H(\boldsymbol{x}^k, \boldsymbol{\lambda}^k)$, and thus solve the system

$$\begin{bmatrix} B^k & \nabla \boldsymbol{g}(\boldsymbol{x}^k)^T \\ \nabla \boldsymbol{g}(\boldsymbol{x}^k) & 0 \end{bmatrix} \begin{pmatrix} \Delta \boldsymbol{x} \\ -\bar{\boldsymbol{\lambda}} \end{pmatrix} = - \begin{pmatrix} \nabla f(\boldsymbol{x}^k) \\ \boldsymbol{g}(\boldsymbol{x}^k) \end{pmatrix}. \tag{4.53}$$

If B^k is invertible, one can solve (4.53) by using a Schur complement approach.

So where in all this does the name SQP come from ? To understand this, we need to consider the quadratic programming problem with linear constraint

$$\begin{aligned} \tfrac{1}{2}\boldsymbol{s}^T B^k \boldsymbol{s} + \boldsymbol{s}^T \nabla f(\boldsymbol{x}) &\rightarrow \min, \qquad (B^k)^T = B^k, \\ \nabla \boldsymbol{g}(\boldsymbol{x})\boldsymbol{s} &= -\boldsymbol{g}(\boldsymbol{x}). \end{aligned} \tag{4.54}$$

If $B^k \approx H(\boldsymbol{x})$, this is a quadratic model of the minimization problem

$$f(\boldsymbol{x} + \boldsymbol{s}) \rightarrow \min,$$

since by Taylor expansion, we have

$$f(\boldsymbol{x} + \boldsymbol{s}) = f(\boldsymbol{x}) + \boldsymbol{s}^T \nabla f(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{s}^T B^k \boldsymbol{s} + \ldots$$

with a linear approximation of the constraint

$$\boldsymbol{g}(\boldsymbol{x} + \boldsymbol{s}) = 0,$$

since by Taylor expansion, we have

$$\boldsymbol{g}(\boldsymbol{x} + \boldsymbol{s}) = \boldsymbol{g}(\boldsymbol{x}) + (\nabla \boldsymbol{g}(\boldsymbol{x}))^T \boldsymbol{s} + \ldots,$$

which means an approximation of our original problem (4.50). The Lagrangian for the model (4.54) is

$$\mathcal{L}^a(\boldsymbol{s}, \bar{\boldsymbol{\lambda}}) = \frac{1}{2}\boldsymbol{s}^T B^k \boldsymbol{s} + \boldsymbol{s}^T \nabla f(\boldsymbol{x}) - (\nabla \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x}))^T \bar{\boldsymbol{\lambda}},$$

and the first order optimality conditions are

$$\begin{aligned} B^k \boldsymbol{s} + \nabla f(\boldsymbol{x}) - \nabla(\boldsymbol{g}(\boldsymbol{x}))^T \bar{\boldsymbol{\lambda}} &= 0 \\ \nabla \boldsymbol{g}(\boldsymbol{x})\boldsymbol{s} &= -\boldsymbol{g}(\boldsymbol{x}), \end{aligned}$$

which is precisely the system (4.53) solved at each iteration by Newtons methods applied to the first order optimality conditions of the original problem (4.50), provided B^k is an approximation of the Hessian of the Lagrangian, and not the objective function only. One can therefore interpret Newtons method as a method solving a sequence of quadratic optimization problems with linear constraints, which explains the name sequential quadratic programming. A generic SQP method is thus given by

ALGORITHM 4.5. *Generic interior point method*

For given $\boldsymbol{x}^0$ and $\boldsymbol{\lambda}^0$;
$k = 0$;
while not converged
  compute $B^k \approx H(\boldsymbol{x}^k, \boldsymbol{\lambda}^k)$; $\quad \boldsymbol{s}^k = \text{argmin}_{\boldsymbol{s} \text{ s.t. } \nabla \boldsymbol{g}(\boldsymbol{x}^k)\boldsymbol{s} = -\boldsymbol{g}(\boldsymbol{x}^k)} \frac{1}{2}\boldsymbol{s}^T B^k \boldsymbol{s} +$
$\boldsymbol{s}\nabla f(\boldsymbol{x}^k)$;
  $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \boldsymbol{s}^k$;
  k=k+1;
end

This method is simple and fast: if $B^k = H(\boldsymbol{x}^k, \boldsymbol{\lambda}^k)$, then convergence is ultimately quadratic as in Newtons method.

If the underlying problem had been an optimization problem with inequality constraints, the subproblem in the generic SQP algorithm would simply have to be replaced by a subproblem of the of the form

$$\begin{array}{rcll} \frac{1}{2}\boldsymbol{s}^T B^k \boldsymbol{s} + \boldsymbol{s}^T \nabla f(\boldsymbol{x}) & \rightarrow & \min, & (B^k)^T = B^k, \\ \nabla \boldsymbol{g}(\boldsymbol{x})\boldsymbol{s} & \geq & -\boldsymbol{g}(\boldsymbol{x}). & \end{array} \tag{4.55}$$

To obtain a more robust method, one adds in general a line search method,

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \boldsymbol{s}^k,$$

and one chooses the line search parameter α^k such that a penalty function

$$\phi(\boldsymbol{x}, \mu) := f(\boldsymbol{x}) + \frac{1}{\mu}||\boldsymbol{g}(\boldsymbol{x})||$$

satisfies a criterion of sufficient decrease. As an alternative, one can also use a trust region approach.

4.5 Problems

References

[1] Bengt Aspvall and John R. Gilbert, *Graph Coloring Using Eigenvalue Decomposition*, SIAM J. ALG. DISC. METH. Vol 5, No. 4, December 1984.

[2] Pierre Baldi and Edward Posner, *Graph Coloring Bounds for Cellular Radio*, Computers Math. Applic. Vol. 19, No. 10, pp 91-97, 1990.

[3] A. Gamst and W. Rave, *On the Frequency Assignment in Mobile Automatic Telephone Systems*, in Proc. of GLOBECOM 82, pp. 309-315, 1982.

[4] Jin-Kao Hao and Raphaël Dorne, *Study of Genetic Search for the Frequency Assignment Problem*, Artificial Evolution European Conference, Springer-Verlag, pp. 333-44, 1996.

[5] A. Knällmann and A. Quellmalz, *Solving the Frequency Assignment Problem with Simulated Annealing*, Electromagnetic Compatibility, Conference Publication No 396, IEE.

[6] A. Quellmalz, A. Knällmann, B. Müller, *Efficient Frequency Assignment with Simulated Annealing*, Antennas and Propagation, Conference Publication No 407, IEE 1995.

[7] Teahoon Park and Chae Y. Lee, *Application of the Graph Coloring Algorithm to the Frequency Assignment Problem*, Journal of the Operations Research Society of Japan, Vol. 39, No. 2, June 1996.

[8] H. D. Simon, *Partitioning of Unstructured Problems for Parallel Processing*, Computing Systems in Engineering, vol 2, no 2/3, pp 135-148, 1991.

[9] J. Zander, *Performance of Optimum Transmitter Power Control in Cellular Radio Systems*, IEEE Transactions on Vehicular Technology, Vol. 41, No. 1, February 1992.