7 Numerical optimization

Let $f : \mathbb{R}^n \to \mathbb{R}$, $n \geq 1$, be a function that we call *cost function* or *objective function*. The problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \tag{7.1}$$

is called unconstrained (or free) optimization problem, whereas

$$\min_{\mathbf{x}\in\Omega} f(\mathbf{x}) \tag{7.2}$$

over a closed proper subset $\Omega \subset \mathbb{R}^n$, is called constrained optimization problem. The set Ω will be determined by either equality or inequality constraints that will be dictated by the nature of the problem to solve. For instance, should we find the optimal allocation of n bounded resources x_i $(i = 1, \ldots, n)$, the constraints will be expressed by inequalities as $0 \leq x_i \leq C_i$ (with C_i given constants) and the set Ω will be the subset of $\mathbb{R}^n$ determined by the fulfilment of the constraints

$$\Omega = \{\mathbf{x} = (x_1, \ldots, x_n) : \ 0 \leq x_i \leq C_i, \ i = 1, \ldots, n\}.$$

Since computing the maximum of a function f is equivalent to compute the minimum of $g = -f$, for the sake of simplicity we will only consider algorithms suitable for minimization problems.

Often, more interesting than the minimum value of the given function is the point at which that minimum is achieved, that we call *minimizer*, the latter of course may not be unique.

This chapter will be essentially devoted to numerical methods for unconstrained optimization and, at a lesser extent, to that of constrained optimization.

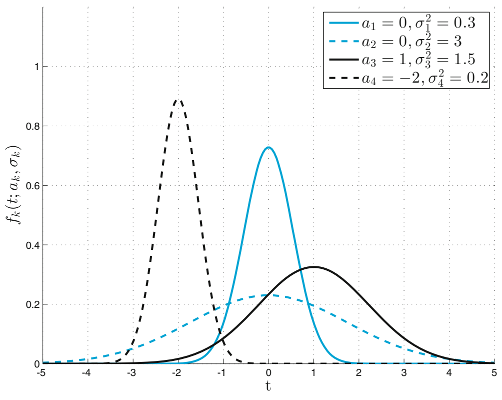

Figure 7.1. Gaussian functions

7.1 Some representative problems

Problem 7.1 (Population dynamics) A colony of 250 bacteries living in an isolated environment self reproduces according to the so called Verhulst model

$$b(t) = \frac{2500}{1 + 9e^{-t/3}}, \quad t > 0$$

where t represents the time (expressed in days) past after the start of the colture ($t = 0$). We would like to find after how many days the population growth rate is maximum. For the solution of this problem, see Examples 7.1 and 7.2. ■

Problem 7.2 (Signal analysis) Applications involving automatic vocal identification, like those implemented on smartphones, compress the acoustic signal into a set of parameters that fully characterize it. The signal intensity is modeled as a sum of m Gaussian functions (also called peaks)

$$f_k(t; a_k, \sigma_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-(t-a_k)^2/(2\sigma_k^2)}, \quad \text{for } k = 1, \ldots, m, \ t \in [t_0, t_f], \tag{7.3}$$

characterized by 2 coefficients for every peak: the *expected value* a_k of the kth peak, that is the center of the peak itself, and its *variance* σ_k^2 (see Figure 7.1). The knowledge of the variance of each peak allows the evaluation of both its height $h_k = 1/\sqrt{2\pi\sigma_k^2}$ and amplitude $w_k = 2\sqrt{\log 4\sigma_k^2}$.

We set now

$$f(t; \mathbf{a}, \boldsymbol{\sigma}) = \sum_{k=1}^{m} f_k(t; a_k, \sigma_k), \tag{7.4}$$

where we have set $\mathbf{a} = [a_1, \ldots, a_m]$ and $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_m]$. The computation of the vectors $\mathbf{a}$ and $\boldsymbol{\sigma}$ entails the solution of the following minimization problem

$$\min_{\mathbf{a},\boldsymbol{\sigma}} \sum_{i=1}^{n} (f(t_i; \mathbf{a}, \boldsymbol{\sigma}) - y_i)^2, \tag{7.5}$$

where (t_i, y_i), $i = 1, \ldots, n$ represent a sampling of our signal. (7.5) is a nonlinear least squares problem that is solved in Example 7.12.

The model (7.4) is also named *Gaussian Mixture Model (GMM)* and is used in statistics for *data mining* and *pattern recognition*. ■

Problem 7.3 (Mesh optimization) Consider a given triangulation of the domain $D \subset \mathbb{R}^2$, as in Figure 7.2, left. We want to modify the position of vertices of the internal triangles in order to optimize the triangles' quality in the sense specified below. Let $(x_i^{(k)}, y_i^{(k)})$ (for $i = 0, 1, 2$) be the coordinates of the vertices of the kth triangle T_k. Define the matrix

$$\mathrm{A}_k = \begin{pmatrix} x_1^{(k)} - x_0^{(k)} & x_2^{(k)} - x_0^{(k)} \\ y_1^{(k)} - y_0^{(k)} & y_2^{(k)} - y_0^{(k)} \end{pmatrix}$$

and the scalar quantity

$$\mu_k = \frac{4\det(\mathrm{A}_k)}{\sqrt{3}\|\mathrm{A}_k \mathrm{W}^{-1}\|_F^2}, \tag{7.6}$$

where $\mathrm{W} = [1, 1/2; 0, \sqrt{3}/2]$ while $\|\mathrm{B}\|_F = \sqrt{\sum_{i,j=1}^2 b_{ij}^2}$ denotes the Frobenius norm of the matrix B. Should T_k be equilateral then $\mu_k = 1$; the more T_k departs from being an equilateral triangle, the more μ_k approaches zero. To optimize our mesh we minimize the function $\sum_{k=1}^{Ne} \mu_k^{-1}$ with respect to the position of the vertices of the internal triangles of D, under the constraints $\det(\mathrm{A}_k) \geq \tau$ (for a given τ), and that no inversion occurs in the ordering of the nodes ([Mun07]). The solution of this problem will be given in Example 7.16.

In Figure 7.2 we display the original triangulation and the optimized one. This kind of problems are faced in the numerical solution of partial differential equations by the finite element method (see Chapter 9). ■

Problem 7.4 (Finance) A given capital is placed in investment funds whose expected rate of interest is 6%, 10%, and 12%, respectively. The risk associated with this investment is modeled by a function that depends on the fractions x_i of the entire capital invested into the three funds, the risk probability of the funds, and their correlations

$$r(\mathbf{x}) = 0.04x_1^2 + 0.25x_2^2 + 0.64x_3^2 + 0.1x_1x_2 + 0.208x_2x_3. \tag{7.7}$$

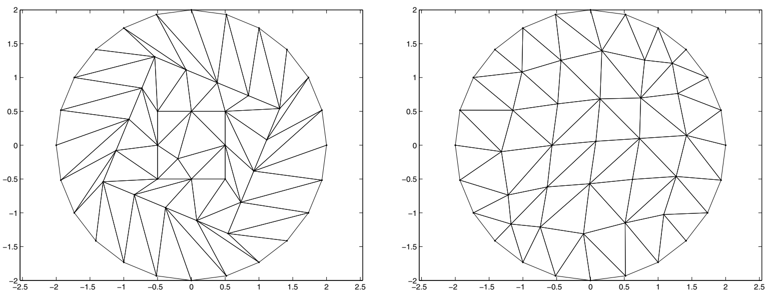

Figure 7.2. Mesh of Problem 7.3. At left the original one, at right that optimized

The goal is to minimize the risk while guaranteeing that the expected return be equal to 10.4%. The solution of this problem is provided in Example 7.14. ■

Problem 7.5 (Road network) Consider a network of n roads and p cross roads as represented in Figure 7.3. Every minute M vehicles travel through the network; on the jth road the maximum speed limit is $v_{j,m}$ km/min, moreover no more than $\rho_{j,m}$ vehicles per km can transit on the jth road s_j.

We aim at finding the optimal density ρ_j (vehicles/km) on the road s_j (with $0 \leq \rho_j \leq \rho_{j,m}$) in order to minimize the average travel time from the inlet (1st node in Figure 7.3) and the outlet (7th node in Figure 7.3). It is assumed that the speed of vehicles traveling along the jth road depends on both the maximum speed $v_{j,m}$ and the maximum density according to the formula $v_j = v_{j,m}(1 - \rho_j/\rho_{j,m})$ km/min. Consequently, the flowrate of vehicles on the jth street is $q_j = \rho_j v_j = \rho_j v_{j,m}(1 - \rho_j/\rho_{j,m})$ vehicles/min. By denoting with t_j (in min) the time necessary to cover the jth road of length L_j km, we find $t_j = L_j/v_j = L_j/(v_{j,m}(1 - \rho_j/\rho_{j,m}))$ min. The objective function to be minimized is

$$f(\boldsymbol{\rho}) = \frac{\sum_{j=1}^{n} t_j \rho_j}{\sum_{j=1}^{n} \rho_j}. \tag{7.8}$$

At every node of the network the vehicles inflow should balance the outflow. By denoting with positive sign those incoming in the ith node ($q_{i,j\text{in}}$) and negative sign those outgoing ($q_{i,j\text{out}}$), the following equations need to be satisfied

$$\sum_{j\text{in}} q_{i,j\text{in}} - \sum_{j\text{out}} q_{i,j\text{out}} = 0 \qquad \text{for } i = 1, \ldots, p. \tag{7.9}$$

This is a constrained minimization problem whose constraints are expressed by both equations and inequalities. See Example 7.17 for its solution. ■

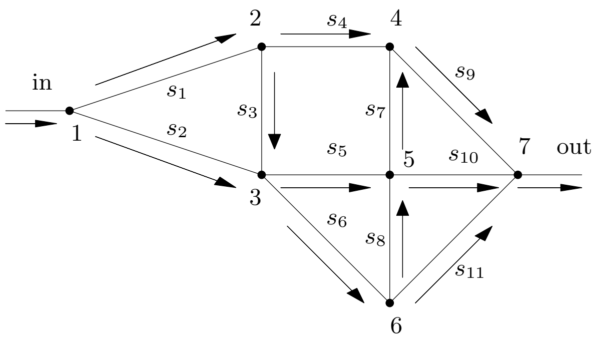

Figure 7.3. The road network of Problem 7.5

7.2 Unconstrained optimization

When minimizing an objective function one might be interested in finding either a local or a global minimizer. A point $\mathbf{x}^* \in \mathbb{R}^n$ is called a *global minimizer* for f if

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in \mathbb{R}^n,$$

while $\mathbf{x}^*$ is a *local minimizer* for f if there exists a ball $B_r(\mathbf{x}^*) \subset \mathbb{R}^n$ centered at $\mathbf{x}^*$ and with radius $r > 0$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in B_r(\mathbf{x}^*).$$

We denote by

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}(\mathbf{x}), \ldots, \frac{\partial f}{\partial x_n}(\mathbf{x}) \right)^T \tag{7.10}$$

the *gradient* of f at point $\mathbf{x} \in \mathbb{R}^n$, provided f is differentiable in $\mathbb{R}^n$.

Moreover we denote by $\mathrm{H}(\mathbf{x})$ the *Hessian matrix* of f at the point $\mathbf{x}$, whose elements are

$$h_{ij}(\mathbf{x}) = \frac{\partial^2 f(\mathbf{x})}{\partial x_j \partial x_i}, \qquad i, j = 1, \ldots, n,$$

provided that second derivatives of f at that point $\mathbf{x}$ do exist.

If $f \in C^2(\mathbb{R}^n)$, that is all first and second order derivatives of f exist and are continuous, than $\mathrm{H}(\mathbf{x})$ is symmetric for every $\mathbf{x} \in \mathbb{R}^n$. A point $\mathbf{x}^*$ is called a *stationary (or critical) point* for f if $\nabla f(\mathbf{x}^*) = \mathbf{0}$, a *regular point* if $\nabla f(\mathbf{x}^*) \neq \mathbf{0}$.

A function f defined on $\mathbb{R}^n$ does not necessarily admit a minimizer; moreover, should such point exist, it is not necessarily unique. For instance $f(\mathbf{x}) = x_1 + 3x_2$ is unbounded in $\mathbb{R}^2$, while $f(\mathbf{x}) =$

$\sin(x_1)\sin(x_2)\cdots\sin(x_n)$ admits an infinite number of minimizers and maximizers in $\mathbb{R}^n$ (either local and global).

The function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $\forall \alpha \in [0, 1]$,

$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha)f(\mathbf{y}); \tag{7.11}$$

it is *Lipschitz continuous* if there exists a costant $L > 0$ such that

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L\|\mathbf{x} - \mathbf{y}\| \qquad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n. \tag{7.12}$$

The following result holds.

Proposition 7.1 (Optimality conditions) *Let $\mathbf{x}^* \in \mathbb{R}^n$. If $\mathbf{x}^*$ is a minimizer for f (either local or global) and if there exists $r > 0$ such that $f \in C^1(B_r(\mathbf{x}^*))$, then $\nabla f(\mathbf{x}^*) = \mathbf{0}$. Moreover, if $f \in C^2(B_r(\mathbf{x}^*))$, the matrix $\mathrm{H}(\mathbf{x}^*)$ is positive semidefinite.*
Viceversa, let $r > 0$ exist such that $f \in C^2(B_r(\mathbf{x}^*))$. If $\nabla f(\mathbf{x}^*) = \mathbf{0}$ and $\mathrm{H}(\mathbf{x})$ is positive definite for all $\mathbf{x} \in B_r(\mathbf{x}^*)$, then $\mathbf{x}^*$ is a local minimizer for f.
Finally, if f is differentiable and convex in $\mathbb{R}^n$ and $\nabla f(\mathbf{x}^*) = \mathbf{0}$, then $\mathbf{x}^*$ is a global minimizer for f.

In view of the numerical solution of optimization problems, an ideal situation is that of a cost function featuring a unique global minimizer. However, most often there exist several local minimizers. This is why in this chapter we will describe numerical methods for the approximation of local minimizers.

Most often, methods for numerical optimization are of iterative type. They may be classified into two categories depending on whether or not they require the knowledge of the derivative of the cost function.

The so called *derivative free* methods make use of direct comparison between values taken by the cost function in order to investigate its local behavior. Among these methods, some make use of numerical approximation (tipically, through finite differences, see Section 9.2.1) of the derivatives, see Section 7.3.

Methods using exact derivatives can take advantage of accurate information on the local function behaviour to achieve faster convergence to the minimizer. As a matter of fact, if f is differentiable at $\overline{\mathbf{x}}$ and $\nabla f(\overline{\mathbf{x}})$ is different than zero, then the maximum growth of f, moving away from $\overline{\mathbf{x}}$, occurs along the (signed) direction given by the vector $\nabla f(\overline{\mathbf{x}})$.

Among the minimization methods that make use of exact derivatives, the two most important classes (based on complementary strategies) are: *descent* (or *line search* methods) and *trust region* methods that will be described in Sections 7.5 and 7.6, respectively.

7.3 Derivative free methods

In this section we describe two simple numerical methods for the minimization of univariate real valued functions or multivariate real valued functions along a one-dimensional direction. We will then describe the Nelder and Mead method for the minimization of functions of several variables.

7.3.1 Golden section and quadratic interpolation methods

Let $f : (a, b) \to \mathbb{R}$ be a continuous function featuring a unique minimizer $x^* \in (a, b)$. We set $I_0 = (a, b)$ and for $k \geq 0$ we generate a sequence of intervals $I_k = (a^{(k)}, b^{(k)})$ of decreasing length, each of those containing x^*.

For any given k, the next interval I_{k+1} is determined as follows. Let $c^{(k)}, d^{(k)} \in I_k$, with $c^{(k)} < d^{(k)}$, be two points such that both

$$\frac{b^{(k)} - a^{(k)}}{d^{(k)} - a^{(k)}} = \frac{d^{(k)} - a^{(k)}}{b^{(k)} - d^{(k)}} = \varphi \tag{7.13}$$

and

$$\frac{b^{(k)} - a^{(k)}}{b^{(k)} - c^{(k)}} = \frac{b^{(k)} - c^{(k)}}{c^{(k)} - a^{(k)}} = \varphi \tag{7.14}$$

be the golden ratio $\varphi = \dfrac{1+\sqrt{5}}{2} \simeq 1.628$.

Thanks to (7.13), (7.14) we find

$$\boxed{c^{(k)} = a^{(k)} + \frac{1}{\varphi^2}(b^{(k)} - a^{(k)}) \quad \text{and} \quad d^{(k)} = a^{(k)} + \frac{1}{\varphi}(b^{(k)} - a^{(k)})} \tag{7.15}$$

The points $c^{(k)}$ and $d^{(k)}$ are symmetrically distributed about the midpoint of I_k, that is

$$\frac{a^{(k)} + b^{(k)}}{2} - c^{(k)} = d^{(k)} - \frac{a^{(k)} + b^{(k)}}{2}. \tag{7.16}$$

Indeed, if we replace $c^{(k)}$ and $d^{(k)}$ in (7.16) with the corresponding expressions given in (7.15), after dividing by the common factor $(b^{(k)} - a^{(k)})/\varphi^2$ we obtain the identity $\varphi^2 - \varphi - 1 = 0$.

Setting $a^{(0)} = a$ and $b^{(0)} = b$, the golden section algorithm is formulated as follows (see Figure 7.4): for $k = 0, 1, \ldots$ until convergence

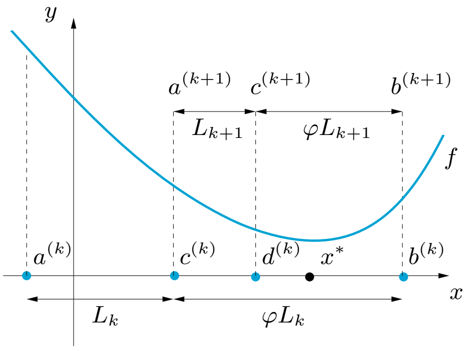

Figure 7.4. A generic iteration of the golden section method for seeking the minimizer of the function f; φ is the golden ratio, while $L_k = c^{(k)} - a^{(k)}$

```
compute $c^{(k)}$ and $d^{(k)}$ through (7.15)
if $f(c^{(k)}) \geq f(d^{(k)})$
   set $I_{k+1} = (a^{(k+1)}, b^{(k+1)}) = (c^{(k)}, b^{(k)})$
else
   set $I_{k+1} = (a^{(k+1)}, b^{(k+1)}) = (a^{(k)}, d^{(k)})$
endif
```
(7.17)

From (7.13) and (7.14) it also follows that $c^{(k+1)} = d^{(k)}$ if $I_{k+1} = (c^{(k)}, b^{(k)})$, while $d^{(k+1)} = c^{(k)}$ if $I_{k+1} = (a^{(k)}, d^{(k)})$.

A stopping criterion for the previous iterations is

$$\boxed{\frac{b^{(k+1)} - a^{(k+1)}}{|c^{(k+1)}| + |d^{(k+1)}|} < \varepsilon} \tag{7.18}$$

where ε is a given tolerance. In the successful case, the midpoint of the last interval I_{k+1} can be taken as an approximation of the desired minimizer x^*.

Still using (7.13) (or (7.14)) we obtain

$$|b^{(k+1)} - a^{(k+1)}| = \frac{1}{\varphi}|b^{(k)} - a^{(k)}| = \ldots = \frac{1}{\varphi^{k+1}}|b^{(0)} - a^{(0)}|, \tag{7.19}$$

that is the golden section method converges linearly with a convergence rate $\varphi^{-1} \simeq 0.618$.

This method is implemented in Program 7.1: `fun` is either an *anonymous function* or an *inline function* representing the function f, `a` and `b` are the endpoints of the search interval, `tol` is the tolerance ε and `kmax` is the maximum allowed number of iterations.

The output variable **xmin** contains the value of the minimizer, **fmin** the minimum value of f in (a, b), **iter** the number of iterations carried out by the algorithm.

Program 7.1. golden: golden section method

```
function [xmin,fmin,iter]=golden(fun,a,b,tol,...
                                     kmax,varargin)
%GOLDEN Approximates a minimizer of 1D-functions
%  XMIN=GOLDEN(FUN,A,B,TOL,KMAX) approximates a
%  minimizer of the function FUN in the interval
%  [A,B] by the golden section method.
%  If the search fails, the program returns an
%  error message. FUN is either an anonymous
%  function, or an inline function, or a function
%  defined in a M-file.
%  XMIN=GOLDEN(FUN,A,B,TOL,KMAX,P1,P2,...) passes
%  parameters P1, P2,... to the function
%  FUN(X,P1,P2,...).
%  [XMIN,FMIN,ITER]= GOLDEN(FUN,...) returns
%  the value of FUN at XMIN and the number of
%  iterations required to compute XMIN.
phi=(1+sqrt(5))/2; phi1=1/phi; phi2=1/(phi+1);
c=a+phi2*(b-a); d=a+phi1*(b-a);
err=tol+1; k=0;
while err>tol & k< kmax
  if(fun(c) >= fun(d))
    a=c; c=d; d=a+phi1*(b-a);
  else
    b=d; d=c; c=a+phi2*(b-a);
  end
  k=k+1; err=abs(b-a)/(abs(c)+abs(d));
end
xmin=(a+b)/2; fmin=fun(xmin); iter=k;
if  (iter==kmax & err > tol)
 fprintf(['The golden section method stopped \n',...
 'without converging to the desired tolerance \n',...
 'because the maximum number of iterations was \n',...
 'reached\n']);
end
```

Example 7.1 Let us solve Problem 7.1 using the golden section method. The function $f(t) = -b'(t) = -7500e^{t/3}/(e^{t/3} + 9)^2$ admits a global minimizer in the interval $[6, 7]$ as it appears from its plot. We use Program 7.1 with tolerance equal to 10^{-8} using the following instructions:

```
f=@(t)[-(7500*exp(t/3))/(exp(t/3) + 9)^2]
a=0; b=10; tol=1.e-8; kmax=100;
[tmin,fmin,iter]=golden(f,a,b,tol,kmax)
```

After 38 iterations we find

```
xmin=6.591673759332620        fmin=-2.083333333333333e+02
```

The maximum growth rate is of $208.\overline{3}$ bacteria per day and occurs about after 6.59 days since the start of the colture. ■

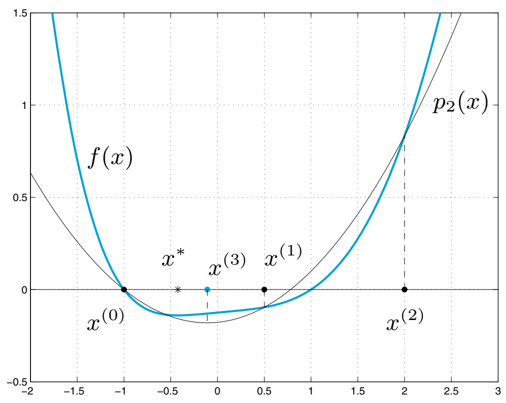

Figure 7.5. The first step of the quadratic interpolation method

The quadratic interpolation method is often used as an alternative to the golden section algorithm.

Let f be a continuous function. Starting from three distinct points $x^{(0)}$, $x^{(1)}$ and $x^{(2)}$, a sequence of points $x^{(k)}$, with $k \geq 3$, is built in a way that $x^{(k+1)}$ represents the vertex (and therefore the minimizer) of the parabola $p_2^{(k)}$ interpolating f at the points $x^{(k)}$, $x^{(k-1)}$, and $x^{(k-2)}$, see Figure 7.5:

$$\begin{aligned} p_2^{(k)}(x) = f(x^{(k-2)}) + f[x^{(k-2)}, x^{(k-1)}](x - x^{(k-2)}) + \\ f[x^{(k-2)}, x^{(k-1)}, x^{(k)}](x - x^{(k-2)})(x - x^{(k-1)}). \end{aligned}$$

Here,

$$f[x_i, x_j] = \frac{f(x_j) - f(x_i)}{x_j - x_i}, \quad f[x_i, x_j, x_\ell] = \frac{f[x_j, x_\ell] - f[x_i, x_j]}{x_\ell - x_i} \tag{7.20}$$

are the so called *Newton divided differences* (see [QSS07, Ch. 8]). By solving the first order equation ${p_2^{(k)}}'(x^{(k+1)}) = 0$ we obtain

$$\boxed{x^{(k+1)} = \frac{1}{2}\left(x^{(k-2)} + x^{(k-1)} - \frac{f[x^{(k-2)}, x^{(k-1)}]}{f[x^{(k-2)}, x^{(k-1)}, x^{(k)}]}\right)} \tag{7.21}$$

We iterate until $|x^{(k+1)} - x^{(k)}| < \varepsilon$ for a prescribed tolerance $\varepsilon > 0$.

Provided for every k the divided difference $f[x^{(k-2)}, x^{(k-1)}, x^{(k)}]$ does not vanish, this method converges super-linearly to the minimizer with a convergence rate $p \simeq 1.3247$ (see [Bre02]). Otherwise, the method may not terminate. For this reason the quadratic interpolation method is tipically used in combination with other methods, such as the golden section method, whose convergence is always guaranteed.

The command MATLAB `fminbnd` implements the combination of these two methods. The calling sintax is `x = fminbnd(fun,a,b)` where `fun` is the function handle associated with the cost function and `a, b` represent the endpoints of the interval containing the minimizer. The output `x` provides the approximation of the minimizer.

`fminbnd`

Example 7.2 We use function `fminbnd` to solve the same problem described in Example 7.1. We use the following commands:

```
a=0; b=10; tol=1.e-8; kmax=100;
[tmin1,fmin1,exitflag,output]=fminbnd(f,a,b,...
                    optimset('TolX',1.e-8));
```

Convergence to `fmin1= 6.591673708945312` is achieved in 8 iterations, much fewer than the 38 iterations requested by the golden section method. The command `optimset` allows fixing the tolerance to a desired value (`tol=1.e-8` in the current case) different than the one that would be otherwise set by default (`tol=1.e-4`). The output optional parameters are: `fmin1` containing the evaluation of f at the minimizer, `exitflag` indicating the termination state, and `output` containing the number of iterations carried out as well as the global number of function evaluations requested by the whole algorithm. ■

`optimset`

As noticed, the two previous methods are genuinely one dimensional, yet they can be used to solve multidimensional optimization problems provided they are restricted to the search of optimizers along a given one dimensional direction (see Section 7.5).

7.3.2 Nelder and Mead method

Let $n > 1$ and $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function.

The $n-$*simplex* with $n+1$ vertices $\mathbf{x}_i \in \mathbb{R}^n$ (with $i = 0, \dots, n$) is the set

$$S = \{\mathbf{y} \in \mathbb{R}^n : \ \mathbf{y} = \sum_{i=0}^{n} \lambda_i \mathbf{x}_i, \ \lambda_i \in \mathbb{R} \text{ and } \lambda_i \geq 0 : \ \sum_{i=0}^{n} \lambda_i = 1\}, \quad (7.22)$$

under the assumption that the n vectors $\mathbf{x}_i - \mathbf{x}_0$ ($i = 1, \dots, n$) be linearly independent (S is a segment in $\mathbb{R}$, a triangle in $\mathbb{R}^2$ and a tethraedron in $\mathbb{R}^3$).

The method of Nelder and Mead [NM65] is a derivative free minimization algorithm which generates a sequence of simplices $\{S^{(k)}\}_{k\geq 0}$ in $\mathbb{R}^n$ that run after or circumscribe the minimizer $\mathbf{x}^* \in \mathbb{R}^n$ of the cost function f. It uses the evaluations of f at the simplices' vertices, as well as simple geometrical transformations such as reflections, expansions and contractions.

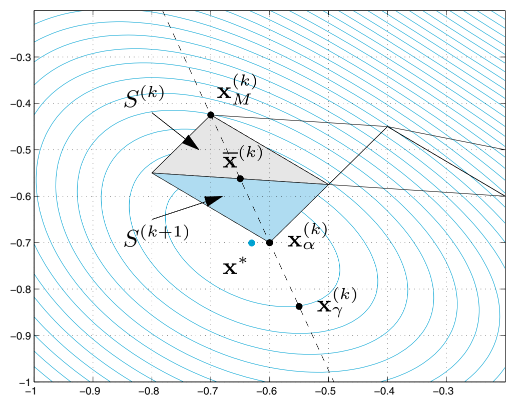
$S^{(k)}$
$\mathbf{x}_M^{(k)}$
$\overline{\mathbf{x}}^{(k)}$
$S^{(k+1)}$
$\mathbf{x}_\alpha^{(k)}$
$\mathbf{x}^*$
$\mathbf{x}_\gamma^{(k)}$

Figure 7.6. One step of the Nelder and Mead method, the point $\mathbf{x}_M^{(k)}$ is replaced by $\mathbf{x}_\alpha^{(k)}$

To generate the initial simplex $S^{(0)}$ we take a point $\tilde{\mathbf{x}} \in \mathbb{R}^n$ and a positive real number η, and set $\mathbf{x}_i^{(0)} = \tilde{\mathbf{x}} + \eta \mathbf{e}_i$ for $i = 0, \dots, n$, where $\{\mathbf{e}_i\}$ are the vectors of the standard basis in $\mathbb{R}^n$.

For every $k \geq 0$ (until convergence) we select the "worst" vertex of $S^{(k)}$

$$\mathbf{x}_M^{(k)} = \underset{0 \leq i \leq n}{\operatorname{argmax}} \, f(\mathbf{x}_i^{(k)}) \tag{7.23}$$

then replace it by a new point to form the new simplex $S^{(k+1)}$.

The new point is chosen as follows. First we select

$$\mathbf{x}_m^{(k)} = \underset{0 \leq i \leq n}{\operatorname{argmin}} \, f(\mathbf{x}_i^{(k)}) \quad \text{and} \quad \mathbf{x}_\mu^{(k)} = \underset{\substack{0 \leq i \leq n \\ i \neq M}}{\operatorname{argmax}} \, f(\mathbf{x}_i^{(k)}) \tag{7.24}$$

and define the *centroid* of the hyperplane $H^{(k)}$ passing through the vertices $\{\mathbf{x}_i^{(k)}, \ i = 0, \dots, n, \ i \neq M\}$

$$\overline{\mathbf{x}}^{(k)} = \frac{1}{n} \sum_{\substack{i=0 \\ i \neq M}}^{n} \mathbf{x}_i^{(k)}. \tag{7.25}$$

(When $n = 2$, the centroid is the midpoint of the edge of $S^{(k)}$ opposite to $\mathbf{x}_M^{(k)}$, see Fig. 7.6.)

Then we compute the reflection $\mathbf{x}_\alpha^{(k)}$ of $\mathbf{x}_M^{(k)}$ with respect to the hyperplane $H^{(k)}$, i.e.

$$\mathbf{x}_\alpha^{(k)} = (1 - \alpha)\overline{\mathbf{x}}^{(k)} + \alpha \mathbf{x}_M^{(k)}, \tag{7.26}$$

where $\alpha < 0$ is the reflection coefficient (tipically, $\alpha = -1$). The point $\mathbf{x}_\alpha^{(k)}$ lies on the straight line joining $\overline{\mathbf{x}}^{(k)}$ and $\mathbf{x}_M^{(k)}$, on the side of $\overline{\mathbf{x}}^{(k)}$ far from $\mathbf{x}_M^{(k)}$ (see Fig. 7.6).

Now, we compare $f(\mathbf{x}_\alpha^{(k)})$ with the values of f at the other vertices of the simplex, before accepting $\mathbf{x}_\alpha^{(k)}$ as the new vertex. Meanwhile, we try to move $\mathbf{x}_\alpha^{(k)}$ on the straight line joining $\overline{\mathbf{x}}^{(k)}$ and $\mathbf{x}_M^{(k)}$, to set the new simplex $S^{(k+1)}$. More precisely we proceed as follows.

- If $f(\mathbf{x}_\alpha^{(k)}) < f(\mathbf{x}_m^{(k)})$, i.e. the reflection has produced a new minimum, we compute
$$\mathbf{x}_\gamma^{(k)} = (1-\gamma)\overline{\mathbf{x}}^{(k)} + \gamma \mathbf{x}_M^{(k)}, \tag{7.27}$$
where $\gamma < -1$ (tipically, $\gamma = -2$). Then, if $f(\mathbf{x}_\gamma^{(k)}) < f(\mathbf{x}_m^{(k)})$, $\mathbf{x}_M^{(k)}$ is replaced by $\mathbf{x}_\gamma^{(k)}$; otherwise $\mathbf{x}_M^{(k)}$ is replaced by $\mathbf{x}_\alpha^{(k)}$; then we proceed by incrementing k by one.
- If $f(\mathbf{x}_m^{(k)}) \le f(\mathbf{x}_\alpha^{(k)}) < f(\mathbf{x}_\mu^{(k)})$, $\mathbf{x}_M^{(k)}$ is replaced by $\mathbf{x}_\alpha^{(k)}$; then we proceed by incrementing k by one.
- If $f(\mathbf{x}_\mu^{(k)}) \le f(\mathbf{x}_\alpha^{(k)}) < f(\mathbf{x}_M^{(k)})$ we compute
$$\mathbf{x}_\beta^{(k)} = (1-\beta)\overline{\mathbf{x}}^{(k)} + \beta \mathbf{x}_\alpha^{(k)}, \tag{7.28}$$
where $\beta > 0$ (tipically, $\beta = 1/2$). Now, if $f(\mathbf{x}_\beta^{(k)}) > f(\mathbf{x}_M^{(k)})$ define the vertices of the new simplex $S^{(k+1)}$ by
$$\mathbf{x}_i^{(k+1)} = \frac{1}{2}(\mathbf{x}_i^{(k)} + \mathbf{x}_m^{(k)}) \tag{7.29}$$
otherwise $\mathbf{x}_M^{(k)}$ is replaced by $\mathbf{x}_\beta^{(k)}$; then we proceed by incrementing k by one.
- If $f(\mathbf{x}_\alpha^{(k)}) > f(\mathbf{x}_M^{(k)})$ we compute
$$\mathbf{x}_\beta^{(k)} = (1-\beta)\overline{\mathbf{x}}^{(k)} + \beta \mathbf{x}_M^{(k)}, \tag{7.30}$$
(again $\beta > 0$), if $f(\mathbf{x}_\beta^{(k)}) > f(\mathbf{x}_M^{(k)})$ define the vertices of the new simplex $S^{(k+1)}$ by (7.29), otherwise $\mathbf{x}_M^{(k)}$ is replaced by $\mathbf{x}_\beta^{(k)}$; then we proceed by incrementing k by one.

As soon as the stopping criterium $\max\limits_{i=0,\ldots,n} \|\mathbf{x}_i^{(k)} - \mathbf{x}_m^{(k)}\|_\infty < \varepsilon$ is fulfilled, $\mathbf{x}_m^{(k)}$ will be retained as an approximation of the minimizer.

The convergence of Nelder and Mead method is guaranteed in very special cases only (see example [LRWW99]); in fact a stagnation may occur in which case the algorithm needs to be restarted. Nevertheless, this algorithm is quite robust and efficient for small dimensional problems. Its rate of convergence is severely affected by the choice of the initial simplex. The Nelder and Mead method is implemented by the MATLAB command `fminsearch`; its sintax is described in the following example. `fminsearch`

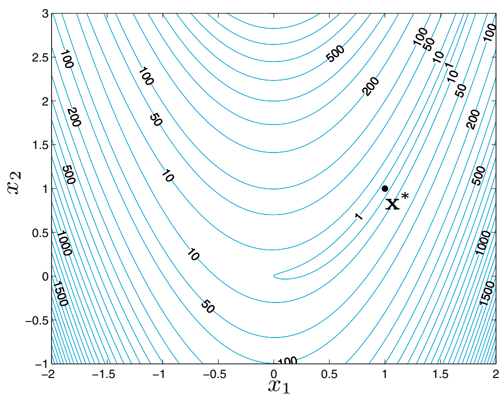

Figure 7.7. Contour lines of the Rosenbrock function

Example 7.3 (The Rosenbrock function) The Rosenbrock function

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

whose contour lines are displayed in Figure 7.7 ([Ros61]), is often used to test both efficiency and robustness of minimization algorithms. Its global minimum is attained at the point $\mathbf{x}^* = (1, 1)$, however its variation around $\mathbf{x}^*$ is fairly low, making algorithms' convergence quite problematic. Through the following command

```
fun=@(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2; x0=[-1.2,1]
xstar=fminsearch(fun,x0)
```

we get

```
xstar =
    1.000022021783570       1.000042219751772
```

In MATLAB, by replacing the second instruction with the expanded one

```
[xstar,fval,exitflag,output]=fminsearch(fun,x0)
```

we would obtain additional information on the minimum value of f `fval=8.1777e-10`, on the number of iterations, `output.iterations=85` as well as the total number of function evaluations `output.funcCount=159`. Finally, the error tolerance can be modified by using the command `optimset`, as already discussed in Example 7.2. ■

See Exercises 7.1-7.3.

7.4 The Newton method

Assume that $f : \mathbb{R}^n \to \mathbb{R}$ $(n \geq 1)$ is of class $C^2(\mathbb{R}^n)$ and that we know how to compute its first and second order partial derivatives. We can apply Newton's method, already introduced in Chapter 2 for the solution of the system $\mathbf{F}(\mathbf{x}) = \nabla f(\mathbf{x}) = \mathbf{0}$, whose Jacobian matrix $J_{\mathbf{F}}(\mathbf{x}^{(k)})$ is nothing but the Hessian matrix of $\mathbf{F}$ computed at the generic iteration point $\mathbf{x}^{(k)}$. The method reads as follows: for a given $\mathbf{x}^{(0)} \in \mathbb{R}^n$, for $k = 0, 1, \ldots$, until convergence

$$\boxed{\begin{array}{ll} \text{solve} & \mathrm{H}(\mathbf{x}^{(k)})\boldsymbol{\delta}\mathbf{x}^{(k)} = -\nabla f(\mathbf{x}^{(k)}) \\ \text{set} & \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)} \end{array}} \tag{7.31}$$

For a given tolerance $\varepsilon > 0$, a suitable stopping test is $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \leq \varepsilon$.

Example 7.4 The function

$$f(\mathbf{x}) = \frac{2}{5} - \frac{1}{10}(5x_1^2 + 5x_2^2 + 3x_1x_2 - x_1 - 2x_2)e^{-(x_1^2+x_2^2)} \tag{7.32}$$

is displayed in Figure 7.8, right. We apply Newton's method to approximate its global minimizer $\mathbf{x}^* \simeq (-0.63065832, -0.7007420)$ (rounded to the first 7 significant digits). We take $\mathbf{x}^{(0)} = (-0.9, -0.9)$ and tolerance $\varepsilon = 10^{-5}$. After 5 iterations the method (7.31) converges to `x=[-0.63058;-0.70074]`. Should we have chosen $\mathbf{x}^{(0)} = (-1, -1)$, after 400 iterations the stopping test would not be fulfilled. In fact a necessary condition for convergence of Newton's method is that $\mathbf{x}^{(0)}$ should be sufficiently close to the minimizer $\mathbf{x}^*$ (see Section 2.3). This is known as *local convergence* of Newton's method.

The reader should be aware that Newton's method may converge to any stationary point (not necessarily to a minimizer). For instance, by taking $\mathbf{x}^{(0)} = (0.5, -0.5)$ after 5 iterations the method converges to the saddle point `x=[0.80659; -0.54010]`. ■

A general convergence criterium for the method (7.31) is as follows: if $f \in C^2(\mathbb{R}^n)$, $\mathbf{x}^*$ is a stationary point, the Hessian matrix $\mathrm{H}(\mathbf{x}^*)$ is positive definite, the components of $\mathrm{H}(\mathbf{x})$ are Lipschitz continuous in a neighbourhood of $\mathbf{x}^*$ and $\mathbf{x}^{(0)}$ is sufficiently close to $\mathbf{x}^*$, then the Newton method (7.31) converges quadratically to $\mathbf{x}^*$ (see, for instance [SY06, pag. 132], [NW06]).

In spite of its simple implementation, Newton's method is computationally demanding when n is large (as it requires the analytic expression of the derivatives and, at each iteration, the computation of both the gradient and the Hessian matrix of f). Besides, $\mathbf{x}^{(0)}$ has to be chosen close enough to $\mathbf{x}^*$.

A suitable strategy to build up efficient and robust minimization algorithms relies on combining locally convergent with globally convergent methods, as described in the next section.

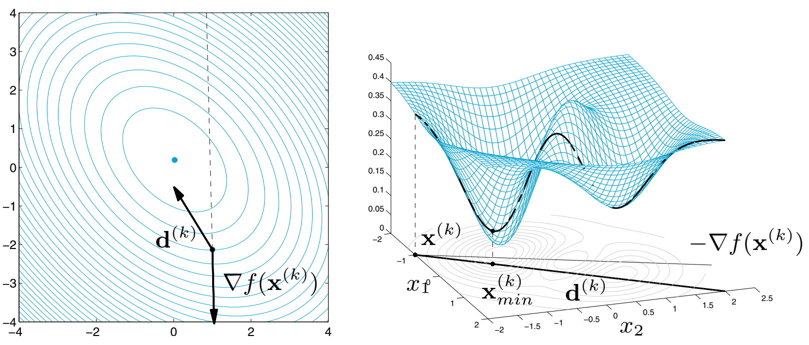

Figure 7.8. At left, countour lines of a function $f(\mathbf{x})$, its gradient evaluated at $\mathbf{x}^{(k)}$ and a suitable descent direction $\mathbf{d}^{(k)}$. At right, the restriction of the function $f(\mathbf{x})$ (7.32) along a descent direction $\mathbf{d}^{(k)}$ and the minimizer $\mathbf{x}^{(k)}_{min}$ along $\mathbf{d}^{(k)}$

7.5 Descent (or line search) methods

In this Section we assume for simplicity that $f \in C^2(\mathbb{R})$ and is bounded from below.

Descent methods (also known as line search methods) are iterative methods in which, for every $k \geq 0$, $\mathbf{x}^{(k+1)}$ depends on $\mathbf{x}^{(k)}$, on a vector $\mathbf{d}^{(k)}$ depending on $\nabla f(\mathbf{x}^{(k)})$ and on a suitable parameter $\alpha_k \in \mathbb{R}$. Given $\mathbf{x}^{(0)} \in \mathbb{R}^n$, the method reads as follows:
for $k = 0, 1, \dots,$ until convergence

$$\boxed{\begin{array}{l}\text{find a direction } \mathbf{d}^{(k)} \in \mathbb{R}^n\\ \text{compute the step } \alpha_k \in \mathbb{R}\\ \text{set } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\end{array}} \tag{7.33}$$

The vector $\mathbf{d}^{(k)}$ must be a *descent direction*, meaning that

$$\begin{array}{ll}\mathbf{d}^{(k)^T}\nabla f(\mathbf{x}^{(k)}) < 0 & \text{if } \nabla f(\mathbf{x}^{(k)}) \neq \mathbf{0},\\ \mathbf{d}^{(k)} = \mathbf{0} & \text{if } \nabla f(\mathbf{x}^{(k)}) = \mathbf{0}.\end{array} \tag{7.34}$$

The name *descent direction* arises from the property that the vector $\nabla f(\mathbf{x}^{(k)})$ provides in $\mathbb{R}^n$ the direction with sign of maximum positive growth of f moving from $\mathbf{x}^{(k)}$. As $\mathbf{d}^{(k)^T}\nabla f(\mathbf{x}^{(k)})$ represents the directional derivative of f along $\mathbf{d}^{(k)}$, the first condition in (7.34) ensures that we are moving along a direction opposite to the gradient, that is towards a minimizer of f, as displayed in Figure 7.8.

Some popular descent directions will be reported in the next Section.

Once $\mathbf{d}^{(k)}$ is determined, the optimum value of $\alpha_k \in \mathbb{R}$ is the one that guarantees the maximum variation of f along $\mathbf{d}^{(k)}$ and can therefore be computed by solving a one-dimensional minimization problem (that is minimizing the restriction of f along $\mathbf{d}^{(k)}$), see Figure 7.8.

However, as the computation of α_k is quite involved when f is not a quadratic function, we will report in Section 7.5.2 some alternative techniques aimed at approximating α_k.

7.5.1 Descent directions

The most widely used descent directions are:

1. *Newton's directions*

$$\boxed{\mathbf{d}^{(k)} = -(\mathrm{H}(\mathbf{x}^{(k)}))^{-1}\nabla f(\mathbf{x}^{(k)})} \tag{7.35}$$

2. *quasi-Newton directions*

$$\boxed{\mathbf{d}^{(k)} = -\mathrm{H}_k^{-1}\nabla f(\mathbf{x}^{(k)})} \tag{7.36}$$

   where H_k represents an approximation of the true Hessian matrix $\mathrm{H}(\mathbf{x}^{(k)})$. This choice is a valuable alternative to Newtons' method when second derivatives of f are heavy to compute (see Section 7.5.4);
3. *gradient directions*

$$\boxed{\mathbf{d}^{(k)} = -\nabla f(\mathbf{x}^{(k)})} \tag{7.37}$$

   (they can be regarded as a trivial example of quasi-Newton directions);
4. *conjugate gradient directions*

$$\boxed{\begin{aligned} &\mathbf{d}^{(0)} = -\nabla f(\mathbf{x}^{(0)}) \\ &\mathbf{d}^{(k+1)} = -\nabla f(\mathbf{x}^{(k+1)})-\beta_k\mathbf{d}^{(k)}, \ k \geq 0 \end{aligned}} \tag{7.38}$$

   The coefficients β_k can be chosen according to different criteria, see Section 7.5.5, however they coincide with those of Conjugate Gradient method for linear systems (see (5.66)) when f is a quadratic function.

The descent direction (7.37) fulfills the condition (7.34), then (7.35) and (7.36) assure that $\mathrm{H}(\mathbf{x}^{(k)})$ and H_k, respectively, are positive definite matrices. The vectors (7.38) fulfill (7.34) provided that the coefficients β_k are suitably chosen, as we will see in Section 7.5.5.

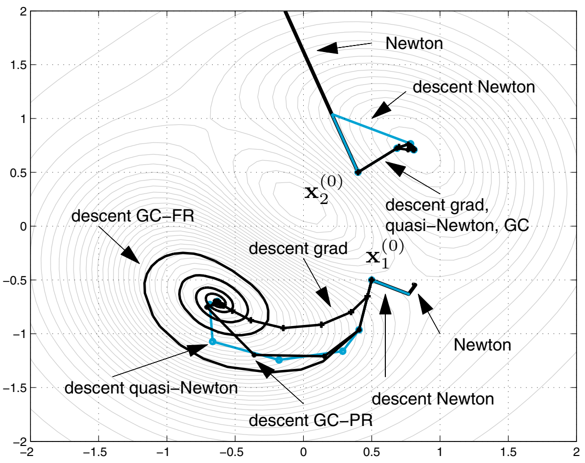

Figure 7.9. Convergence history of Newton's and descent methods for the function of the Example 7.5

Example 7.5 Consider again the function $f(\mathbf{x})$ (7.32), featuring two local minimizers, one local maximizer and two saddle points. See Figure 7.8, right. We compare the sequences $\{\mathbf{x}^{(k)}\}$ generated by Newton's method (7.31) and descent methods with descent directions given by (7.35)–(7.38).

Consider first $\mathbf{x}_1^{(0)} = (0.5, -0.5)$ as initial point. In Figure 7.9 we see that Newton's method (7.31) converges to the saddle point $(.8065, -.5401)$; the descent method with Newton direction (7.35) breaks down at the second iteration as it generates a matrix $\mathrm{H}(\mathbf{x}^{(1)})$ which is not definite positive. (See Remark 7.2 on how to overcome this drawback.) The other descent methods with directions given by (7.36), (7.37), and (7.38) (for the latter, two different criteria for the determination of the parameters β_k have been used, named GC-FR and GC-PR, see Section 7.5.5) converge to the local minimizer $(-0.6306, -0.7007)$. The faster convergence is achieved in 9 iterations using quasi-Newton directions (7.36), see the blue path in Figure 7.9. By choosing a different initial point $\mathbf{x}_2^{(0)} = (0.4, 0.5)$, the Newton method diverges while method (7.35), even though it shares the same first descent direction with Newton's method, builds up a short steplength α_k which then allows convergence to the local minimizer $(0.8095, 0.7097)$ in only 4 iterations. All the other descent methods with directions (7.36), (7.37), and (7.38) converge in 10 to 15 iterations to the same local minimizer. ■

The choice of the steplength α_k will be discussed in Section 7.5.2, while the analysis of different descent directions is deferred to Sections 7.5.3–7.5.5.

7.5.2 Strategies for choosing the steplength α_k

Once the descent direction $\mathbf{d}^{(k)}$ is chosen, the steplength α_k has to be determined in such a way that the new iterate $\mathbf{x}^{(k+1)}$ be the minimizer of f along such a direction, that is

$$\alpha_k = \underset{\alpha\in\mathbb{R}}{\operatorname{argmin}}\ f(\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)}).$$

A second order Taylor expansion of f around $\mathbf{x}^{(k)}$ yields

$$\begin{aligned} f(\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)}) = f(\mathbf{x}^{(k)}) &+ \alpha{\mathbf{d}^{(k)}}^T\nabla f(\mathbf{x}^{(k)}) + \\ &\frac{\alpha^2}{2}{\mathbf{d}^{(k)}}^T \mathrm{H}(\mathbf{x}^{(k)})\mathbf{d}^{(k)} + o(\|\alpha\mathbf{d}^{(k)}\|^2). \end{aligned} \tag{7.39}$$

In the special case in which f is a quadratic function, that is

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathrm{A}\mathbf{x} - \mathbf{x}^T\mathbf{b} + c$$

with $\mathrm{A} \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$ symmetric and positive definite and $c \in \mathbb{R}$, the expansion in (7.39) is exact, that is the infinitesimal residual is null. As $\mathrm{H}(\mathbf{x}^{(k)}) = \mathrm{A}$ for every $k \geq 0$ and $\nabla f(\mathbf{x}^{(k)}) = \mathrm{A}\mathbf{x}^{(k)} - \mathbf{b} = -\mathbf{r}^{(k)}$ (see (5.35)), by differentiating (7.39) with respect to α and setting the derivative equal to zero we obtain

$$\boxed{\alpha_k = \frac{{\mathbf{d}^{(k)}}^T\mathbf{r}^{(k)}}{{\mathbf{d}^{(k)}}^T\mathrm{A}\mathbf{d}^{(k)}}} \tag{7.40}$$

In the case (7.37), we find $\mathbf{d}^{(k)} = \mathbf{r}^{(k)}$ thus we obtain the well known gradient method described in Chapter 5, which obeys the convergence estimate (5.59).

Instead, should the direction $\mathbf{d}^{(k)}$ be chosen as in (7.38), by setting

$$\beta_k = -\frac{(\mathrm{A}\mathbf{d}^{(k)})^T\mathbf{r}^{(k+1)}}{{\mathbf{d}^{(k)}}^T\mathrm{A}\mathbf{d}^{(k)}} \tag{7.41}$$

we would recover the conjugate gradient method (5.66) for linear systems which fulfills the convergence estimate (5.67).

If f is a generic (non quadratic) function, the computation of the optimal α_k would require an iterative method to solve numerically the above minimization problem along the direction $\mathbf{d}^{(k)}$. In these cases an approximate (rather than exact) value of α_k can be chosen by requiring that the new iterate $\mathbf{x}^{(k+1)}$ defined in (7.33) ensures that

$$f(\mathbf{x}^{(k+1)}) < f(\mathbf{x}^{(k)}). \tag{7.42}$$

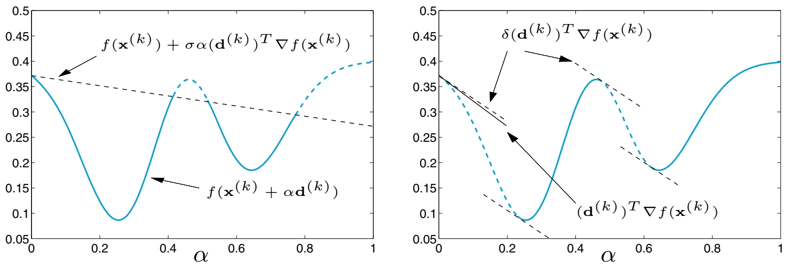

Figure 7.10. At left, the terms comparing in the first inequality in (7.43) when $\sigma = 0.2$. $(7.43)_1$ is satisfied for those values of α providing the continuous lightblue line. At right, some straightlines with slope $\delta\mathbf{d}^{(k)^T}\nabla f(\mathbf{x}^{(k)})$ and $\delta = 0.9$, $(7.43)_2$ is fulfilled for those α corresponding to the continuous lightblue line. The Wolfe conditions are simultaneously fulfilled when either $0.23 \leq \alpha \leq 0.41$ or $0.62 \leq \alpha \leq 0.77$

In this respect, a natural strategy is that of assigning α_k a large value and then reducing it iteratively until when (7.42) is satisfied. Unfortunately, this strategy does not guarantee that the associated sequence $\{\mathbf{x}^{(k)}\}$ converges to the desired minimizer $\mathbf{x}^*$. See Exercise 7.4 and the associated Figure 10.8, left, where steplengths are too long. See also Exercise 7.5 and the associated Figure 10.8, right, where steplengths are now too short.

A better criterium for the choice of $\alpha_k > 0$ is the one based on the *Wolfe's conditions*:

$$\boxed{\begin{aligned}&f(\mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)}) \leq f(\mathbf{x}^{(k)}) + \sigma\alpha_k\mathbf{d}^{(k)^T}\nabla f(\mathbf{x}^{(k)})\\ &\mathbf{d}^{(k)^T}\nabla f(\mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)}) \geq \delta\mathbf{d}^{(k)^T}\nabla f(\mathbf{x}^{(k)})\end{aligned}} \tag{7.43}$$

where σ and δ, such that $0 < \sigma < \delta < 1$, are two given constants and $\mathbf{d}^{(k)^T}\nabla f(\mathbf{x}^{(k)})$ represents the directional derivative of f along the direction $\mathbf{d}^{(k)}$.

The first inequality in (7.43) is named *Armijo's rule*, and it inhibits too little variations of f with respect to the steplength α_k (see Figure 7.10, left). More precisely, the larger α_k the higher the variation of f.

The second Wolfe condition states that at the new point $\mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)}$ the value of the directional derivative of f should be larger than δ times the same derivative at the previous value $\mathbf{x}^{(k)}$ (see Figure 7.10, right).

From the example depicted in Figure 7.10 one can see that Wolfe's conditions might also be fulfilled far from the minimizer of f along $\mathbf{d}^{(k)}$ and even when the directional derivative of f takes large values. More restricitive conditions than (7.43) are the *strong Wolfe's conditions*

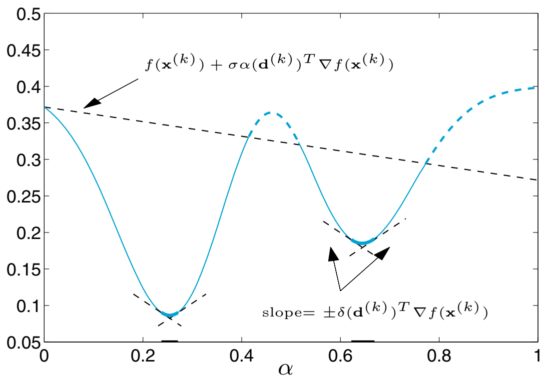

Figure 7.11. The strong Wolfe's conditions (7.44) are fulfilled when α belongs to small intervals around the minimizers, in correspondence with the thick lightblue piece of curve. $\sigma = 0.2$ and $\delta = 0.9$ have been considered.

$$\boxed{\begin{aligned} & f(\mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)}) \leq f(\mathbf{x}^{(k)}) + \sigma\alpha_k{\mathbf{d}^{(k)}}^T\nabla f(\mathbf{x}^{(k)}), \\ & |{\mathbf{d}^{(k)}}^T\nabla f(\mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)})| \leq -\delta{\mathbf{d}^{(k)}}^T\nabla f(\mathbf{x}^{(k)}) \end{aligned}} \tag{7.44}$$

being $0 < \sigma < \delta < 1$ suitable fixed constants.

The first condition is the same as in (7.43), whereas the second one gives rise to $(7.43)_2$ as well as to ${\mathbf{d}^{(k)}}^T\nabla f(\mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)}) \leq -\delta{\mathbf{d}^{(k)}}^T\nabla f(\mathbf{x}^{(k)})$ (having recalled that the right hand side of $(7.44)_2$ is positive because of $(7.34)_1$). Conditions $(7.44)_2$ inhibits f to vary too strongly at $\mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)}$ (see Figure 7.11 for an example).

It can be proved (see, e.g., [NW06, Lemma 3.1]) that if $\mathbf{d}^{(k)}$ is a descent direction in $\mathbf{x}^{(k)}$ and $f \in C^1(\mathbb{R}^n)$ is lower bounded in the set $\{\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)},\ \alpha > 0\}$, then for every σ, δ such that $0 < \sigma < \delta < 1$, there exist intervals of steplengths α_k satisfying (7.43) and (7.44).

In practice, σ is chosen very small, e.g. $\sigma = 10^{-4}$ ([NW06]), while δ is large ($\delta = 0.9$) for Newton, quasi-Newton and gradient directions, small ($\delta = 0.1$) for the conjugate gradient directions.

A simple strategy to determine the steplength α_k satisfying Wolfe's conditions is *backtracking*: it consists of starting with $\alpha = 1$ and then reducing it by a prescribed factor ρ (tipically, $\rho \in [1/10, 1/2)$) until when the first condition (7.43) is satisfied. In pseudocode: for a given $\mathbf{x}^{(k)}$ and a descent direction $\mathbf{d}^{(k)}$, for $\sigma \in (0, 1)$, $\rho \in [1/10, 1/2)$

```
set $\alpha = 1$
while $f(\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)}) > f(\mathbf{x}^{(k)}) + \sigma\alpha{\mathbf{d}^{(k)}}^T\nabla f(\mathbf{x}^{(k)})$
    $\alpha = \alpha\rho$
end
set $\alpha_k = \alpha$
```
(7.45)

The second condition in (7.43) is never checked because the backtracking technique intrinsically computes steplengths that are not too small.

Remark 7.1 The backtracking technique is often combined with replacing f by a quadratic or cubic interpolant of f along $\mathbf{d}^{(k)}$. The chosen steplength α_k yields a new point $\mathbf{x}^{(k+1)}$ which represents the minimizer of the interpolant of f along $\mathbf{d}^{(k)}$. The corresponding algorithm is named quadratic or cubic line search, respectively. See [NW06, Ch. 3] for further details on this approach. ■

The Program `backtrack` 7.2 implements the strategy (7.45). Parameters `fun` and `grad` are function handles rispectively associated with the functions $f(\mathbf{x})$ and $\nabla f(\mathbf{x})$; `xk` and `dk` respectively contain the point $\mathbf{x}^{(k)}$ and the descent direction $\mathbf{d}^{(k)}$, while `sigma` and `rho` contain the parameter values σ and ρ. When `sigma` and `rho` are not specified, the default values $\sigma = 10^{-4}$ and $\rho = 1/4$ are set. The output variable `x` contains the new point $\mathbf{x}^{(k+1)}$.

Program 7.2. backtrack: backtracking strategy

```
function  [x,alphak]= backtrack(fun,xk,gk,dk,varargin)
%BACKTRACK Backtracking strategy for line search.
%   [X,ALPHAK] = BACKTRACK(FUN,XK,GK,DK) computes the
%   new point x_{k+1}=x_k+alpha_k d_k, where alpha_k
%   is determined by the backtracking technique
%   with sigma=1.e-4 and rho=1/4.
%   [X,ALPHAK] = BACKTRACK(FUN,XK,GK,DK,SIGMA,RHO)
%   allows to specify the parameters sigma and rho.
%   Tipically 1.e-4<sigma<0.1 and 1/10< rho <1/2.
%   FUN is the function handle associated with the cost
%   function. XK, GK, and DK contain respectively the
%   point x_k, the gradient of f at x_k and the
%   descent direction d_k.
if nargin==4
    sigma=1.e-4; rho=1/4;
else
    sigma=varargin{1}; rho=varargin{2};
end
alphamin=1.e-5; % minimum value allowed for alpha_k
alphak = 1; fk = fun(xk);
k=0; x=xk+alphak*dk;
while fun(x)>fk+sigma*alphak*gk'*dk & alphak>alphamin
    alphak = alphak*rho;
    x = xk+alphak*dk; k = k+1;
end
```

The Program `descent` 7.3 implements the descent method (7.33) with directions (7.35)–(7.38) and steplengths α_k determined according to the backtracking strategy. The stopping criterium is (see [JS96])

$$\boxed{\max_{1\le i\le n}\left|\frac{[\nabla f(\mathbf{x}^{(k+1)})]_i \max\{|\mathbf{x}_i^{(k+1)}|, \mathrm{typ}(\mathbf{x}_i)\}}{\max\{|f(\mathbf{x}^{(k+1)})|, \mathrm{typ}(f(\mathbf{x}))\}}\right| \le \varepsilon} \qquad (7.46)$$

for a given $\varepsilon > 0$, where $\mathrm{typ}(x)$ is a characteristic value expressing the order of magnitude of the x variable. Its presence prevents test failure when either $\mathbf{x}^*$ or $f(\mathbf{x}^*)$ are null.

Parameters `fun` and `grad` are function handles associated with $f(\mathbf{x})$ and $\nabla f(\mathbf{x})$, respectively, `x0` contains the initial value of the sequence, `tol` the tolerance of the stopping criterium and `kmax` the maximum allowed number of iterations. The variable `meth` sorts the descent direction: Newton's directions correspond to `meth=1`, quasi-Newton's to `meth=2`, gradient directions to `meth=3`, while `meth=41, 42, 43` select three different directions of the conjugate gradient: CG-FR, CG-PR, and CG-HS, respectively, as we will see in Section 7.5.5.

Program 7.3. descent: descent method

```
function [x,err,iter]= descent(fun,grad,x0,tol,kmax,...
                               meth,varargin)
%DESCENT Descent method for optimization
%  [X,ERR,ITER]=DESCENT(FUN,GRAD,X0,TOL,KMAX,METH,HESS)
%  computes a local minimizer of function FUN by the
%  descent method with Newton directions (METH=1),
%  quasi-Newton directions (BFGS) (METH=2), gradient
%  directions (METH=3) or conjugate gradient directions
%  with Fletcher and Reeves beta_k (METH=41),
%  Polak and Ribiere beta_k (METH=42),
%  Hestenes and Stiefel beta_k (METH=43).
%  The steplength is computed by the backtracking
%  technique.  FUN, GRAD and HESS (the latter being
%  used only if METH=1) are function handles associated
%  with the cost function, its gradient and its Hessian
%  matrix, respectively. If METH=2, HESS is a matrix
%  approximating the Hessian of FUN at the initial
%  point X0. TOL is the tolerance for the stopping
%  test, while KMAX is the maximum allowed number of
%  iterations. The function backtrack is called inside.
if nargin>6
if meth==1, hess=varargin{1};
elseif meth==2, H=varargin{1}; end
end
err=tol+1; k=0; xk=x0(:); gk=grad(xk); dk=-gk;
eps2=sqrt(eps);
while err>tol & k< kmax
if meth==1;         H=hess(xk); dk=-H\gk; % Newton
elseif meth==2      dk=-H\gk;             % BFGS
elseif meth==3      dk=-gk;               % gradient
```

```
end
[xk1,alphak]= backtrack(fun,xk,gk,dk);
gk1=grad(xk1);
if meth==2 % BFGS update
  yk=gk1-gk; sk=xk1-xk; yks=yk'*sk;
  if yks> eps2*norm(sk)*norm(yk)
  Hs=H*sk;
  H=H+(yk*yk')/yks-(Hs*Hs')/(sk'*Hs);
  end
elseif meth>=40 % CG update
  if meth == 41
   betak=-(gk1'*gk1)/(gk'*gk); % FR
  elseif meth == 42
   betak=-(gk1'*(gk1-gk))/(gk'*gk); % PR
  elseif meth == 43
   betak=-(gk1'*(gk1-gk))/(dk'*(gk1-gk)); % HS
  end
  dk=-gk1-betak*dk;
end
xk=xk1; gk=gk1; k=k+1; xkt=xk1;
for i=1:length(xk1); xkt(i)=max([abs(xk1(i)),1]); end
err=norm((gk1.*xkt)/max([abs(fun(xk1)),1]),inf);
end
x=xk; iter=k;
if (k==kmax & err > tol)
 fprintf(['Descent method stopped \n',...
 'without converging to the desired tolerance \n',...
 'because the maximum number of iterations was \n',...
 'reached\n']);
end
```

Example 7.6 Consider again function $f(\mathbf{x})$ (7.32). To approximate its global minimizer $(-0.6306, -0.7007)$, we use the `diff` command introduced in Section 1.5.3 for the symbolic computation of both the gradient of f and the Hessian matrix H of f. Then we define the function handles `f`, `grad_f`, and `hess` respectively associated with f, ∇f, and H and call the Program 7.3 with the following instructions:

```
x0=[0.5;-0.5]; tol=1.e-5; kmax=200;
meth=1;  % Newton's directions
[x1,err1,k1]= descent(f,grad_f,x0,tol,kmax,meth,hess);
meth=2; hess=eye(2); % quasi-Newton directions
[x2,err2,k2]= descent(f,grad_f,x0,tol,kmax,meth,hess);
meth=3; % gradient directions
[x3,err3,k3]= descent(f,grad_f,x0,tol,kmax,meth);
meth=42; % CG-PR directions
[x4,err4,k4]= descent(f,grad_f,x0,tol,kmax,meth);
```

We choose $\mathbf{x}^{(0)} = (0.5, -0.5)$, tolerance 10^{-5} and maximum number of iterations equal to 200 and obtain these results:

```
descent Newton k=200,      x=[ 7.7015e-01,-6.3212e-01]
descent quasi-Newton k=9,  x=[-6.3058e-01,-7.0075e-01]
descent gradient k=17,     x=[-6.3058e-01,-7.0075e-01]
descent CG-PR k=17,        x=[-6.3060e-01,-7.0073e-01]
```

Note that the descent method with Newton's direction has not achieved convergence because directions can be generated that do not fulfill condition (7.34). ■

In the next sections we indicate how to compute the approximate Hessian matrices H_k and the parameters β_k in (7.36) and (7.38). Moreover, we will comment on the convergence properties of the various methods introduced so far.

7.5.3 The descent method with Newton's directions

Consider a lower bounded function $f \in C^2(\mathbb{R}^n)$ and the descent method (7.33) with Newton's descent directions (7.35) and steplengths α_k fulfilling the Wolfe's conditions (7.43).

Assume that for every $k \geq 0$ the Hessian matrix $\mathrm{H}(\mathbf{x}^{(k)})$ in (7.35), besides being symmetric thanks to the assumption on f, is positive definite. Moreover, setting $\mathrm{B}_k = \mathrm{H}(\mathbf{x}^{(k)})$ we suppose that

$$\exists M > 0 : \ K(\mathrm{B}_k) = \|\mathrm{B}_k\| \|\mathrm{B}_k^{-1}\| \leq M \qquad \forall k \geq 0. \tag{7.47}$$

(Note that $K(\mathrm{B}_k)$ coincides with the spectral condition number of B_k, see (5.31).)

Then the sequence $\mathbf{x}^{(k)}$ generated by (7.33) converges to a stationary point $\mathbf{x}^*$ of f. Moreover, by choosing $\alpha_k = 1$ from a given $\overline{k}$ on (that is when we are sufficiently close to $\mathbf{x}^*$) the convergence order is quadratic. See [NW06, Thm. 3.2] for the proof.

Remark 7.2 Since the Hessian matrices are positive definite, the stationary point $\mathbf{x}^*$ must necessarily be a minimizer.

However, should $\mathrm{H}(\mathbf{x}^{(k)})$ fail to be positive definite for a given k, the corresponding $\mathbf{d}^{(k)}$ in (7.35) could fail to be a descent direction and the Wolfe conditions might become meaningless. To overcome this problem the Hessian matrix could be replaced by $\mathrm{B}_k = \mathrm{H}(\mathbf{x}^{(k)}) + \mathrm{E}_k$ for a suitable matrix E_k (either diagonal or not) in such a way that B_k is positive definite and $\mathbf{d}^{(k)} = -\mathrm{B}_k^{-1}\nabla f(\mathbf{x}^{(k)})$ turns out to be a descent direction. ■

The descent method with Newton's directions is implemented in Program 7.3.

Example 7.7 Let us compute the global minimizer of the function $f(\mathbf{x})$ (7.32) by using the descent method (7.33), with the Newton's directions (7.35) and steplengths α_k satisfying the Wolfe conditions. We use a tolerance $\varepsilon = 10^{-5}$ for the stopping criterium and we start from $\mathbf{x}^{(0)} = (-1, -1)$. By using Program 7.3 with `meth=1`, after 4 iterations, we have convergence to `x=[-0.63058;-0.70074]`. Choosing instead $\mathbf{x}^{(0)} = (0.5, -0.5)$, the method stagnates as $\mathrm{H}(\mathbf{x}^{(0)})$ is not positive definite, yielding a vector $\mathbf{d}^{(0)}$ which is not a descent direction; consequently, the backtracking technique is unable to find a value $\alpha_0 > 0$ that fulfills the Wolfe conditions. ■

7.5.4 Descent methods with quasi-Newton directions

When using the directions (7.36) we need a strategy to build H_k. For a given symmetric and positive definite matrix H_0, a popular recursive technique is that based on the so called *rank-one update* of Broyden's method (2.19) for the solution of nonlinear systems. The matrices H_k are required:

– to satisfy the secant condition

$$\mathrm{H}_{k+1}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)});$$

– to be symmetric, as $\mathrm{H}(\mathbf{x})$;
– to be positive definite to guarantee that vectors $\mathbf{d}^{(k)}$ are descent directions;
– to satisfy the condition

$$\lim_{k\to\infty} \frac{\|(\mathrm{H}_k - \mathrm{H}(\mathbf{x}^*))\mathbf{d}^{(k)}\|}{\|\mathbf{d}^{(k)}\|} = 0,$$

which, from one hand, ensures that H_k is a good approximation of $\mathrm{H}(\mathbf{x}^*)$ along the descent direction $\mathbf{d}^{(k)}$ and, from the other hand, guarantees a super-linear rate of convergence.

The strategy due to Broyden, Fletcher, Goldfarb, and Shanno (BFGS) is based on the following recursivity relationship

$$\boxed{\mathrm{H}_{k+1} = \mathrm{H}_k + \frac{\mathbf{y}^{(k)}\mathbf{y}^{(k)T}}{\mathbf{y}^{(k)T}\mathbf{s}^{(k)}} - \frac{\mathrm{H}_k\mathbf{s}^{(k)}\mathbf{s}^{(k)T}\mathrm{H}_k}{\mathbf{s}^{(k)T}\mathrm{H}_k\mathbf{s}^{(k)}}} \qquad (7.48)$$

where $\mathbf{s}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$ and $\mathbf{y}^{(k)} = \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)})$. These matrices are symmetric and positive definite under the condition $\mathbf{y}^{(k)T}\mathbf{s}^{(k)} > 0$, which is fulfilled provided the steplengths α_k satisfy the Wolfe conditions (either (7.43) or (7.44)). See [JS96].

The corresponding BFGS method (implemented in Program 7.3) can be summarized as follows: for a given $\mathbf{x}^{(0)} \in \mathbb{R}^n$ and a suitable symmetric and positive definite matrix $\mathrm{H}_0 \in \mathbb{R}^{n\times n}$ which approximates $\mathrm{H}(\mathbf{x}^{(0)})$, for $k = 0, 1, \ldots$, until convergence:

$$\boxed{\begin{array}{ll} \text{solve} & \mathrm{H}_k\mathbf{d}^{(k)} = -\nabla\mathbf{f}(\mathbf{x}^{(k)}) \\ \text{compute} & \alpha_k \text{ satisfying Wolfe's conditions} \\ \text{set} & \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)} \\ & \mathbf{s}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \\ & \mathbf{y}^{(k)} = \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)}) \\ \text{compute} & \mathrm{H}_{k+1} \text{ using (7.48)} \end{array}} \qquad (7.49)$$

Under the condition that $f \in C^2(\mathbb{R}^n)$ is lower bounded and the matrices H_k are positive definite with a condition number uniformly bounded (see (7.47)), the BFGS method converges to a minimizer with (superlinear) convergence order $p \in (1,2)$ (see for instance [JS96, NW06]).

Example 7.8 We apply the BFGS method (7.49) to compute the minimizer of the (yet another time) function $f(\mathbf{x})$ (7.32). We choose $\varepsilon = 10^{-5}$ for the stopping criterium and H_0 equal to the identity matrix (which is obviously symmetric and positive definite). The latter choice is more convenient than choosing $\mathrm{H}_0 = \mathrm{H}(\mathbf{x}^{(0)})$, i.e. the exact Hessian in $\mathbf{x}^{(0)}$, as it yields a faster convergence. Program 7.3 with `meth=2` and `hess=eye(2)` converges to `x=[-0.63058;-0.70074]` in 6 iterations if $\mathbf{x}^{(0)} = (-1,-1)$ and in 9 iterations if $\mathbf{x}^{(0)} = (0.5,-0.5)$. ■

Remark 7.3 As in Broyden method (2.19), the computational cost of order $\mathcal{O}(n^3)$ for the calculation of $\mathbf{d}^{(k)} = -\mathrm{H}_k^{-1}\nabla f(\mathbf{x}^{(k)})$ can be reduced to order $\mathcal{O}(n^2)$, by using QR factorizations of H_k (see [GM72]).
An alternative strategy is based on the use of the inverse $\widetilde{\mathrm{H}}_k$ of H_k both in (7.48) and (7.49). This strategy can be implemented in order of $\mathcal{O}(n^2)$ operations per step, however in practice it is less stable than the more standard (7.48). ■

The BFGS method (as well as several other minimization methods) is implemented in the MATLAB function `fminunc` included in the *optimization toolbox*. By the following commands:

fminunc

```
fun=@(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2; x0=[1.2;-1];
options = optimset('LargeScale','off');
[x,fval,exitflag,output]=fminunc(fun,x0,options)
```

the function `fminunc` computes the minimizer of the Rosenbrock function using the BFGS method (which corresponds to using the value `'off'` to initialize the option `'LargeScale'`). The output parameters have the same meaning as those of the function `fminsearch` described in Example 7.3. Convergence is achieved in 24 iterations with a tolerance $\varepsilon = 10^{-6}$; this has required 93 function evaluations.

With the previous options the gradient of the function f is approximated in `fminunc` by using finite difference methods (see Section 9.2.1). However, in case an exact expression of the gradient of f is available, it can be passed to the function as follows:

```
fun=@(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2; x0=[1.2;-1];
grad_fun=@(x)[-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
   200*(x(2)-x(1)^2)];
options = optimset('LargeScale','off','GradObj','on');
[x,fval,exitflag,output]=fminunc({fun,grad_fun},...
                         x0,options)
```

Note the changement in the command `options`. Convergence is achieved in 25 iterations with 32 function evaluations.

bfgsmin

Octave 7.1 The BFGS method is implemented in the Octave function `bfgsmin`. The Octave command `fminunc` instead implements the *trust region* method that we describe in Section 7.6. ■

7.5.5 Gradient and conjugate gradient descent methods

Let us consider the descent method (7.33) with gradient directions (7.37). As already noticed, the latter are descent directions.

If $f \in C^2(\mathbb{R}^n)$ is lower bounded and the steplengths α_k satisfy Wolfe's conditions, this method converges linearly to a steady point ([NW06]). See Program 7.3 for its implementation.

Example 7.9 We consider once more the function (7.32). We fix the tolerance $\varepsilon = 10^{-5}$ for the stopping criterium and call Program 7.3 setting `meth=3` (this corresponds to gradient directions). Choosing $\mathbf{x}^{(0)} = (-0.9, -0.9)$, $\mathbf{x}^{(0)} = (-1, -1)$ and $\mathbf{x}^{(0)} = (0.5, -0.5)$, the method converges to the global minimizer `x=[-0.63058;-0.70074]` in 11, 12, and 17 iterations, respectively. Choosing instead $\mathbf{x}^{(0)} = (0.9, 0.9)$, which is closer to the local minimizer $\mathbf{x}^* = (.8094399, .7097390)$, the method converges to the latter in 21 iterations. ■

Consider now the conjugate gradient directions (7.38). Several options are available for the choice of β_k (see for instance [SY06, NW06]). Among those we quote the following:

1. *Fletcher–Reeves (1964)*

$$\boxed{\beta_k^{FR} = -\frac{\|\nabla f(\mathbf{x}^{(k)})\|^2}{\|\nabla f(\mathbf{x}^{(k-1)})\|^2}} \tag{7.50}$$

2. *Polak–Ribière (1969) (also known as Polak–Ribière–Polyak parameters)*

$$\boxed{\beta_k^{PR} = -\frac{\nabla f(\mathbf{x}^{(k)})^T(\nabla f(\mathbf{x}^{(k)}) - \nabla f(\mathbf{x}^{(k-1)}))}{\|\nabla f(\mathbf{x}^{(k-1)})\|^2}} \tag{7.51}$$

3. *Hestenes–Stiefel (1952)*

$$\boxed{\beta_k^{HS} = -\frac{\nabla f(\mathbf{x}^{(k)})^T(\nabla f(\mathbf{x}^{(k)}) - \nabla f(\mathbf{x}^{(k-1)}))}{\mathbf{d}^{(k-1)^T}(\nabla f(\mathbf{x}^{(k)}) - \nabla f(\mathbf{x}^{(k-1)}))}} \tag{7.52}$$

In fact, all these choices reduce to (7.41) if f is a quadratic convex function.

For coherence, we will indicate with FR (respectively, PR, HS) the directions associated with β_k^{FR} (respectively, β_k^{PR}, β_k^{HS}).

The following are sufficient conditions for the FR conjugate gradient converge to a steady point ([NW06, SY06]): $f \in C^1(\mathbb{R}^n)$, its gradient is Lipschitz continuous, the initial point $\mathbf{x}^{(0)}$ is such that the set $A = \{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{x}^{(0)})\}$ is bounded and the steplengths α_k satisfy the strong Wolfe's conditions (7.44) with $0 < \sigma < \delta < 1/2$.

Under the same assumptions on f and $\mathbf{x}^{(0)}$ and under the condition that β_k^{PR} is replaced by $\beta_k^{PR+} = \max\{-\beta_k^{PR}, 0\}$ also the PR conjugate gradient method with these modified coefficients converges to a steady point, provided however that the steplengths α_k undergo a variant of the strong Wolfe's conditions (7.44). Same conclusions hold for the HS conjugate gradient algorithm. We refer to [Noc92, NW06, SY06] for the proof and a more in-depth analysis.

The conjugate gradient method with FR, PR, and HS directions and steplengths α_k computed by the *backtracking* technique are all implemented in Program 7.3.

Example 7.10 Still on the function (7.32) we fix a tolerance $\varepsilon = 10^{-5}$ for the stopping criterium and call Program 7.3 by setting `meth=41, 42, 43`, which correspond to the conjugate gradient method associated with directions FR, PR, and HS, respectively. The number of iterations are reported in the table below.

Directions	$\mathbf{x}^{(0)}$		
	$(-1, -1)$	$(1, 1)$	$(0.5, -0.5)$
FR	20	12	>400
PR	21	28	17
HS	23	40	28

For both choices $\mathbf{x}^{(0)} = (-1, -1)$ and $\mathbf{x}^{(0)} = (0.5, -0.5)$, the method converges to the global minimizer `x=[-0.63058;-0.70074]`, whereas with $\mathbf{x}^{(0)} = (1, 1)$ all the variants converge to the local minimizer `x=[0.8094;0.7097]`. ■

Several remarks are in order.

From the previous table and Fig. 7.9, we see that directions PR and HS are more efficient than FR. The latter may be quite inefficient and generate very tiny steplengths. This may yield very slow convergence or even stagnation; in the latter case the algorithm can be restarted by using a gradient direction $\mathbf{d}^{(k)} = -\nabla f(\mathbf{x}^{(k)})$.

When the steplengths α_k are computed exactly (as described at the beginning of Sect. 7.5.1) the rate of convergence of the conjugate gradient method is simply linear, that of Newton methods quadratic, while that of quasi-Newton's super-linear. In spite of that, the conjugate gradient method is simple to implement: it does not require the Hessian matrix (neither its approximations) and only one evaluation of f and its gradient is required at every iteration. It is definitely preferable on large dimensional optimization problems, whereas Newton and quasi-Newton methods are in general more efficient on small dimensional problems.

See Exercises 7.4-7.6.

7.6 Trust region methods

At the generic kth step, line search methods determine the descent direction $\mathbf{d}^{(k)}$ first and then the steplength α_k. Instead trust region methods choose direction and steplength simultaneously by building a ball centered at $\mathbf{x}^{(k)}$ and radius δ_k (the so called trust region), a quadratic approximation $\tilde{f}_k$ of the objective function f and choosing the new value $\mathbf{x}^{(k+1)}$ as the minimizer of $\tilde{f}_k$ in the trust region, see Figure 7.12.

More precisely, we start by a "trust" value $\delta_k > 0$, we use second order Taylor development of f about $\mathbf{x}^{(k)}$ to compute $\tilde{f}_k$,

$$\tilde{f}_k(\mathbf{s}) = f(\mathbf{x}^{(k)}) + \mathbf{s}^T \nabla f(\mathbf{x}^{(k)}) + \frac{1}{2}\mathbf{s}^T \mathrm{H}_k \mathbf{s} \qquad \forall \mathbf{s} \in \mathbb{R}^n \tag{7.53}$$

where H_k is either Hessian of f at $\mathbf{x}^{(k)}$ or a suitable approximation of it, then we compute

$$\mathbf{s}^{(k)} = \underset{\mathbf{s}\in\mathbb{R}^n:\ \|\mathbf{s}\|\leq\delta_k}{\operatorname{argmin}} \tilde{f}_k(\mathbf{s}). \tag{7.54}$$

At this stage we compute

$$\rho_k = \frac{f(\mathbf{x}^{(k)} + \mathbf{s}^{(k)}) - f(\mathbf{x}^{(k)})}{\tilde{f}_k(\mathbf{s}^{(k)}) - \tilde{f}_k(\mathbf{0})}, \tag{7.55}$$

then we proceed as follows:
i) If ρ_k is close to one, we accept $\mathbf{s}^{(k)}$ and move to the next iteration. However, if the minimizer of $\tilde{f}_k$ lies on the border of the trust region, we extend the latter before proceeding with the next iteration.
ii) If ρ_k is either negative or positive and small (much smaller than one), we reduce the trust region and look for a new $\mathbf{s}^{(k)}$ by solving again problem (7.54).
iii) Finally if ρ_k is much larger than one, we accept $\mathbf{s}^{(k)}$, we keep the trust region as is, and move to the next iteration.

Should the second derivative of f be available we could take H_k equal to the Hessian (or, in case the latter fails to be positive definite, one of its variants described in Remark 7.2). Otherwise, H_k can be built recursively as done for quasi-Newton descent direction method (see Sect. 7.5.4).

Assume that: H_k is symmetric positive definite and $\|\mathrm{H}_k^{-1}\nabla f(\mathbf{x}^{(k)})\| \leq \delta_k$; then (7.54) admits $\mathbf{s}^{(k)} = \mathrm{H}_k^{-1}\nabla f(\mathbf{x}^{(k)})$ as minimizer in the trust region. Otherwise the minimizer of $\tilde{f}_k$ lies at the exterior of the trust region; in that case one has to solve a minimization problem for $\tilde{f}_k$ constrained to the circumference centered at $\mathbf{x}^{(k)}$ with radius δ_k, that is

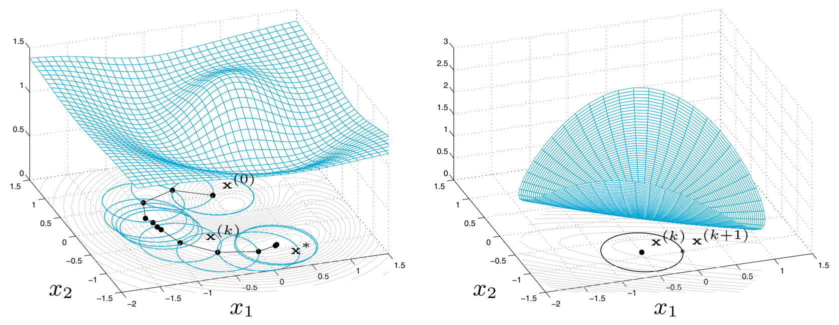

Figure 7.12. Convergence history of trust region method *(at left)* and the quadratic model $\tilde{f}_k$ at step $k = 8$ *(at right)*

$$\min_{\mathbf{s}\in\mathbb{R}^n:\ \|\mathbf{s}\|=\delta_k} \tilde{f}_k(\mathbf{s}). \tag{7.56}$$

To solve (7.56) we can use the Lagrange multipliers approach (see Section 7.8.2), that is we look for the saddle point of the Lagrangian $\mathcal{L}_k(\mathbf{s},\lambda) = \tilde{f}_k(\mathbf{s}) + \frac{1}{2}\lambda(\mathbf{s}^T\mathbf{s} - \delta_k^2)$, i.e. for a vector $\mathbf{s}^{(k)}$ and a scalar $\lambda^{(k)} > 0$ satisfying:

$$\begin{array}{l}(\mathrm{H}_k + \lambda^{(k)}\mathrm{I})\mathbf{s}^{(k)} = -\nabla f(\mathbf{x}^{(k)}),\\ (\mathrm{H}_k + \lambda^{(k)}\mathrm{I}) \text{ is semidefinite positive}\\ \|\mathbf{s}^{(k)}\| - \delta_k = 0.\end{array} \tag{7.57}$$

From $(7.57)_1$ we formally derive $\mathbf{s}^{(k)} = \mathbf{s}^{(k)}(\lambda^{(k)})$ and we replace it into $(7.57)_3$ to get the nonlinear equation

$$\varphi(\lambda^{(k)}) = \frac{1}{\|\mathbf{s}^{(k)}(\lambda^{(k)})\|} - \frac{1}{\delta_k} = 0.$$

The reason for using instead of $(7.57)_3$ its reciprocal is that the latter is easier to solve numerically. Indeed few Newton iterations (tipically, 3 or less) suffice. Precisely, for a given $\lambda_0^{(k)}$, setting $\mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k)})$, we proceed as follows:

```
for $\ell = 0, \ldots, 2$
    compute $\mathbf{s}_\ell^{(k)} = -(\mathrm{H}_k + \lambda_\ell^{(k)}\mathrm{I})^{-1}\mathbf{g}^{(k)}$
    evaluate $\varphi(\lambda_\ell^{(k)}) = \frac{1}{\|\mathbf{s}_\ell^{(k)}\|} - \frac{1}{\delta_k}$
    evaluate $\varphi'(\lambda_\ell^{(k)})$
    update $\lambda_{\ell+1}^{(k)} = \lambda_\ell^{(k)} - \frac{\varphi(\lambda_\ell^{(k)})}{\varphi'(\lambda_\ell^{(k)})}$
```

The vector $\mathbf{s}_\ell^{(k)}$ is obtained by using the Cholesky factorization (5.18) of $\mathrm{B}_\ell^{(k)} = (\mathrm{H}_k + \lambda_\ell^{(k)}\mathrm{I})$ provided this matrix is positive definite. (Notice that $\mathrm{B}_\ell^{(k)}$ is symmetric, in view of the definition of H_k, and its eigenvalues are all real.) More in general, instead of $\mathrm{B}_\ell^{(k)}$ we use $(\mathrm{B}_\ell^{(k)} + \beta\mathrm{I})$ where β is larger than the negative eigenvalue of maximum modulus of $\mathrm{B}_\ell^{(k)}$.

By suitably representing the derivative of $\varphi(\lambda^{(k)})$, problem (7.54) can be solved by using the following algorithm: for $\mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k)})$ and a given δ_k,

```
solve $\mathrm{H}_k\mathbf{s} = -\mathbf{g}^{(k)}$
if $\|\mathbf{s}\| \leq \delta_k$ and $\mathrm{H}_k$ is positive definite
   set $\mathbf{s}^{(k)} = \mathbf{s}$
else
   compute $\beta_1 =$ the negative eigenvalue of $\mathrm{H}_k$
      with largest modulus
   set $\lambda_0^{(k)} = 2|\beta_1|$
   for $\ell = 0, \ldots, 2$
      compute $\mathrm{R}$ : $\mathrm{R}^T\mathrm{R} = \mathrm{H}_k + \lambda_\ell^{(k)}\mathrm{I}$
      solve $\mathrm{R}^T\mathrm{R}\mathbf{s} = -\mathbf{g}^{(k)}$, $\mathrm{R}^T\mathbf{q} = \mathbf{s}$
      update $\lambda_{\ell+1}^{(k)} = \lambda_\ell^{(k)} + \left(\frac{\|\mathbf{s}\|}{\|\mathbf{q}\|}\right)^2 \frac{\|\mathbf{s}\| - \delta_k}{\delta_k}$
   set $\mathbf{s}^{(k)} = \mathbf{s}$
endif
```
(7.58)

In conclusion, we provide the trust region algorithm in its simplest form for the solution of the minimization problem (7.1) ([CL96a, CL96b]). Consider an initial point $\mathbf{x}^{(0)}$, a maximum value $\hat{\delta} > 0$ for the trust region radii and an initial radius $0 < \delta_0 < \hat{\delta}$. Consider then four real parameters η_1, η_2, γ_1 and γ_2 such that $0 < \eta_1 < \eta_2 < 1$ and $0 < \gamma_1 < 1 < \gamma_2$ for updating the trust region and a real parameter $0 \leq \mu < \eta_1$ for the acceptability of the solution. For $k = 0, 1, \ldots$, until convergence

```
compute $f(\mathbf{x}^{(k)})$, $\nabla f(\mathbf{x}^{(k)})$ and $\mathrm{H}_k$,
solve $\min_{\|\mathbf{s}\|_2 \le \delta_k} \tilde{f}_k(\mathbf{s})$ by (7.58)
compute $\rho_k$ using (7.55),
if $\rho_k > \mu$
    set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{s}^{(k)}$
else
    set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$
endif                                                        (7.59)
if $\rho_k < \eta_1$
    set $\delta_{k+1} = \gamma_1 \delta_k$
elseif $\eta_1 \le \rho_k \le \eta_2$
    set $\delta_{k+1} = \delta_k$
elseif $\rho_k > \eta_2$ and $\|\mathbf{s}^{(k)}\| = \delta_k$
    set $\delta_{k+1} = \min\{\gamma_2 \delta_k, \hat{\delta}\}$
endif
```

A possible choice of parameters is $\eta_1 = 1/4$, $\eta_2 = 3/4$, $\gamma_1 = 1/4$, $\gamma_2 = 2$ (see [NW06]). By choosing $\mu = 0$ we accept any step yielding a decrease of f; choosing instead $\mu > 0$ we only accept steps for which the variation of f be at least μ times that of its quadratic model $\tilde{f}_k$.

Remark 7.4 (Approximate solution of (7.54)) Problem (7.54) can be solved approximately, using however an approximation that does not affect the convergence properties of the trust region method. A possible strategy consists in solving the problem not in the whole $\mathbb{R}^n$ but rather in a subspace of dimension two. More precisely, we look for the solution of

$$\min_{\mathbf{s} \in \mathcal{S}_k:\ \|\mathbf{s}\| \le \delta_k} \tilde{f}_k(\mathbf{s}). \tag{7.60}$$

If H_k is positive (or negative) definite, $\mathcal{S}_k = \mathrm{span}\{\nabla f(\mathbf{x}^{(k)}), \mathrm{H}_k^{-1} \nabla f(\mathbf{x}^{(k)})\}$; otherwise we compute its negative eigenvalue β_1 with maximum modulus and set $\mathcal{S}_k = \mathrm{span}\{\nabla f(\mathbf{x}^{(k)}), (\mathrm{H}_k + \alpha I)^{-1} \nabla f(\mathbf{x}^{(k)})\}$, with $\alpha \in (-\beta_1, -2\beta_1]$. The choice of these subspaces is motivated by the search of the so-called Cauchy point, the minimizer of $\tilde{f}_k$ along the directional gradient and internal to the trust region ([NW06]). The most demanding computational effort when solving (7.60) consists in the factorization of either H_k or $\mathrm{H}_k + \alpha I$ and in computing its eigenvalue β_1. However, the computational cost required by (7.60) is definitely lower than that necessary to solve (7.54). ■

The algorithm (7.59) is implemented in Program 7.4. Parameters `fun, grad, x0, tol, kmax` have the same meaning as in the Program `descent` 7.3. Moreover, `delta0` is the radius of the initial trust region, `meth` characterizes the choice of matrices H_k: if `meth=1`, `hess` contains the function handle of the Hessian of f and H_k is the exact Hessian. If `meth` is different than one there is no need to pass the input variable `hess`; in this case H_k is a rank-one approximation of the Hessian computed as in (7.48).

Program 7.4. trustregion: trust region method

```
function [x,err,iter]= trustregion(fun,grad,x0,...
                delta0,tol,kmax,meth,hess)
%TRUSTREGION Trust region method for minimization
%  [X,ERR,ITER]=TRUSTREGION(FUN,GRAD,X0,TOL,KMAX,...
%  METH,HESS) computes a local minimizer of function
%  f by the trust region method. FUN and GRAD
%  (and HESS) are the function handles of the cost
%  function, its gradient (and its Hessian).
%  If METH=1, the Hessian HESS of f is used, otherwise
%  rank-one updates approximations of the Hessian are
%  built as in BFGS and the variable HESS is not requi-
%  red. X0 is the initial point for the sequence gene-
%  rated by the method. TOL is the tolerance for the
%  stopping test, KMAX is the maximum number of
%  iterations allowed.
delta=delta0; err=tol+1; k=0; mu=0.1;
eta1=0.25; eta2=0.75; gamma1=0.25; gamma2=2; deltam=5;
xk=x0(:);   gk=grad(xk);  eps2=sqrt(eps);
if meth==1 Hk=hess(xk); else Hk=eye(length(xk)); end
while err>tol & k< kmax
[s]=trustone(Hk,gk,delta);
rho=(fun(xk+s)-fun(xk))/(s'*gk+0.5*s'*Hk*s);
if rho> mu, xk1=xk+s; else, xk1=xk; end
if rho<eta1
     delta=gamma1*delta;
elseif rho> eta2 & abs(norm(s)-delta)<sqrt(eps)
    delta=min([gamma2*delta,deltam]);
end
gk1=grad(xk1);
err=norm((gk1.*xk1)/max([abs(fun(xk1)),1]),inf);
if meth==1 % Newton
   xk=xk1; gk=gk1; Hk=hess(xk);
else       % quasiNewton
  gk1=grad(xk1); yk=gk1-gk; sk=xk1-xk;
  yks=yk'*sk;
  if yks> eps2*norm(sk)*norm(yk)
  Hs=Hk*sk;
  Hk=Hk+(yk*yk')/yks-(Hs*Hs')/(sk'*Hs);
  end
  xk=xk1; gk=gk1;
end
k=k+1;
end
x=xk; iter=k;
if (k==kmax & err > tol)
 fprintf(['The trust region method stopped \n',...
 'without converging to the desired tolerance \n',...
 'because the maximum number of iterations was \n',...
 'reached\n']);
end
```

```
end

function [s]=trustone(Hk,gk,delta)
s=-Hk\gk; d = eigs(Hk,1,'sa');
if norm(s)>delta | d<0
lambda=abs(2*d); I=eye(size(Hk));
for l=1:3
R=chol(Hk+lambda*I);
s=-R \ (R'\gk); q=R'\s;
lambda=lambda+(s'*s)/(q'*q)*(norm(s)-delta)/delta;
if lambda< -d, lambda=abs(lambda*2); end
end
end
end
```

Example 7.11 Let us compute the minimizer of $f(x_1, x_2) = (x_1 + 2x_2 + 2x_1x_2 - 5x_1^2 - 5x_2^2)/(5e^{x_1^2+x_2^2}) + 7/5$ using method (7.59). As seen in Figure 7.12, this function features a local maximum, a saddle point and two local minima, one $\mathbf{x}_{m1}$ in proximity of $(-1., 0.2)$ and the other $\mathbf{x}_{m2}$ in proximity of $(0.3, -0.9)$; the latter is also a global minimizer. We choose $\mathbf{x}^{(0)} = (0, 0.5)$ and compute the matrices H_k recursively, according to (7.48). By calling the Program `trustregion` with the following instructions:

```
fun=@(x)7/5+(x(1)+2*x(2)+2*x(1)*x(2)-5*x(1)^2-...
     5*x(2)^2)/(5*exp(x(1)^2+x(2)^2));
grad_fun=@(x)[(1+2*x(2)-10*x(1)-2*x(1)*(x(1)+2*x(2)+...
2*x(1)*x(2)-5*x(1)^2-5*x(2)^2))/(5*exp(x(1)^2+x(2)^2));
(2+2*x(1)-10*x(2)-2*x(2)*(x(1)+2*x(2)+...
2*x(1)*x(2)-5*x(1)^2-5*x(2)^2))/(5*exp(x(1)^2+x(2)^2))];
delta0=0.5; tol=1.e-5; kmax=100; meth=2; x0=[0;0.5];
[x,err,iter]= trustregion(fun,grad_fun,x0,delta0,...
              tol,kmax,meth)
```

convergence to the point $(.27849, -.89695)$ is achieved in 24 iterations.

Using instead `meth=1` and at each step the exact Hessian matrix, convergence will be achieved in 12 iterations. In both cases a slowing down in convergence is observed when the iterates $\mathbf{x}^{(k)}$ are near the local minimum $\mathbf{x}_{m1}$. The convergence history when using the exact Hessian is reported in Figure 7.12, left, while Figure 7.13, corresponds to using a non-exact Hessian. ■

For the convergence analysis of the trust region method we refer to [NW06, Sez. 4.2] and [SSB85].

The MATLAB command `fminunc` with the `'LargeScale'` option initialized to the value `'on'` implements the trust region method, and the function handle `grad_fun` contains the gradient of the objective function. For instance, using the following instructions:

```
fun=@(x)100*(x(2)-x(1)^2)^2+(1-x(1))^2;   x0=[1.2;-1];
grad_fun=@(x)[-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
   200*(x(2)-x(1)^2)];
options = optimset('LargeScale','on','GradObj','on');
[x,fval,exitflag,output]=fminunc({fun,grad_fun},...
          x0,options)
```

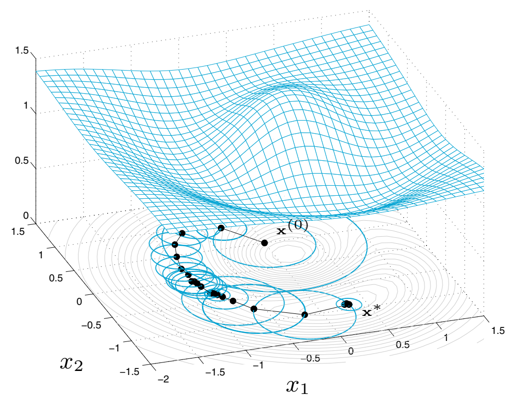

Figure 7.13. Convergence history of the trust region method when H_k are built as in (7.48)

we converge to the minimizer of the Rosenbrock function in 8 iterations; only 9 function evaluations are requested.

Octave 7.2 The `fminunc` command in Octave implements the trust region method with approximated Hessian matrices H_k, computed according to the BFGS recursive formula (7.48). The option `'LargeScale'` is not used in this case. ■

See Exercise 7.7.

7.7 The nonlinear least squares method

In Chapter 3 we have introduced the least squares method for the approximation of either functions or a discrete set of data, by a polynomial (3.29) or another function $\tilde{f}$ linearly depending on a set of unknown coefficients a_j, $j = 1, \dots, m$. When such a dependence is nonlinear, we face a nonlinear least squares problem.

In abstract terms, let $\mathbf{R}(\mathbf{x}) = (r_1(\mathbf{x}), \dots, r_n(\mathbf{x}))^T$, with $r_i : \mathbb{R}^m \to \mathbb{R}$, be a given function, and consider the following minimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^m} \Phi(\mathbf{x}) \quad \text{with} \quad \Phi(\mathbf{x}) = \frac{1}{2}\|\mathbf{R}(\mathbf{x})\|^2 = \frac{1}{2}\sum_{i=1}^{n} r_i^2(\mathbf{x}). \tag{7.61}$$

When the function r_i are nonlinear, Φ might not be convex, featuring several stationary points. All the methods considered thus far, that is Newton's (7.31), descents (7.33) and trust region (7.59), can virtually be used to solve (7.61).

Thanks to the special form of Φ, its gradient and Hessian can be written in terms of the Jacobian $\mathrm{J}_{\mathbf{R}}(\mathbf{x}) \in \mathbb{R}^{n\times m}$ and of the first and second derivatives of $\mathbf{R}$, as follows:

$$\begin{aligned} &\nabla\Phi(\mathbf{x}) = \mathrm{J}_{\mathbf{R}}(\mathbf{x})^T\mathbf{R}(\mathbf{x}),\\ &\mathrm{H}(\mathbf{x}) = \mathrm{J}_{\mathbf{R}}(\mathbf{x})^T\mathrm{J}_{\mathbf{R}}(\mathbf{x}) + \mathrm{S}(\mathbf{x}),\\ &\text{with } \mathrm{S}_{\ell j}(\mathbf{x}) = \sum_{i=1}^{n}\frac{\partial^2 r_i}{\partial x_\ell \partial x_j}(\mathbf{x})r_i(\mathbf{x}),\ \ \ell, j = 1,\ldots,m. \end{aligned} \tag{7.62}$$

Exact calculation of the Hessian can be cumbersome when m and n are large, especially due to the presence of the matrix $\mathrm{S}(\mathbf{x})$. On the other hand, in several cases the latter matrix is less influent than $\mathrm{J}_{\mathbf{R}}(\mathbf{x})^T\mathrm{J}_{\mathbf{R}}(\mathbf{x})$ and could be approximated or even neglected in the construction of $\mathrm{H}(\mathbf{x})$. This is the case of the two methods that we are going to present in the next two sections.

7.7.1 Gauss-Newton method

This method is a variant of the Newton method (7.31) for the solution of (7.61) in which the exact Hessian $\mathrm{H}(\mathbf{x})$ is approximated by neglecting $\mathrm{S}(\mathbf{x})$ in (7.62).

Its formulation is as follows: given $\mathbf{x}^{(0)} \in \mathbb{R}^m$, for $k = 0, 1, \ldots,$ until convergence:

$$\boxed{\begin{aligned} &\text{solve } \left[\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})\right]\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T\mathbf{R}(\mathbf{x}^{(k)})\\ &\text{set } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)} \end{aligned}} \tag{7.63}$$

If $\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})$ has not full rank, the linear system $(7.63)_1$ features infinitely many solutions, in which case the Gauss-Newton method can stagnate, diverge, or converge to a non-stationary point.

If instead $\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})$ has full rank, system $(7.63)_1$ features the form (5.42) and can be solved using either a QR factorization or a singular value decomposition of $\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})$ as seen in Section 5.7.

It can be proved (see Exercise 7.8) that neglecting $\mathrm{S}(\mathbf{x}^{(k)})$ at the step k of the minimization process amounts to approximate $\mathbf{R}(\mathbf{x})$ with its Taylor development centered at $\mathbf{x}^{(k)}$ and truncated at the first order

$$\widetilde{\mathbf{R}}_k(\mathbf{x}) = \mathbf{R}(\mathbf{x}^{(k)}) + \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}). \tag{7.64}$$

The convergence of Gauss-Newton method is not always guaranteed. It actually depends on both the property of Φ and the choice of the initial point. The following result is proved in [JS96]: if $\mathbf{x}^*$ is a stationary point for Φ and $\mathrm{J}_{\mathbf{R}}(\mathbf{x})$ has full rank in a suitable neighborhood of $\mathbf{x}^*$, we have:

1. if $\mathrm{S}(\mathbf{x}^*) = 0$, which is the case if $\mathbf{R}(\mathbf{x})$ is linear or $\mathbf{R}(\mathbf{x}^*) = \mathbf{0}$, the Gauss-Newton method is locally (quadratically) convergent (in fact it coincides with Newton's method);
2. if $\|\mathrm{S}(\mathbf{x}^*)\|_2$ is small with respect to the minimum (positive) eigenvalue of $\mathrm{J}_{\mathbf{R}}(\mathbf{x}^*)^T\mathrm{J}_{\mathbf{R}}(\mathbf{x}^*)$, then Gauss-Newton method converges linearly. This is for instance the case if $\mathbf{R}(\mathbf{x})$ is nonlinear with a small non-linearity or if $\mathbf{R}(\mathbf{x}^*)$ is small;
3. if $\|\mathrm{S}(\mathbf{x}^*)\|_2$ is large with respect to the minimum (positive) eigenvalue of $\mathrm{J}_{\mathbf{R}}(\mathbf{x}^*)^T\mathrm{J}_{\mathbf{R}}(\mathbf{x}^*)$, the Gauss-Newton method might not converge even if $\mathbf{x}^{(0)}$ is very close to $\mathbf{x}^*$. This happens if $\mathbf{R}(\mathbf{x})$ is strongly nonlinear or its residual $\mathbf{R}(\mathbf{x}^*)$ is large.

Remark 7.5 Line search techniques can be used in combination with the Gauss-Newton method by replacing $(7.63)_2$ with $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \boldsymbol{\delta}\mathbf{x}^{(k)}$, where the computation of the steplengths α_k is described in Section 7.5.1. If $\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})$ has full rank, the matrix $\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})^T\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})$ is symmetric and positive definite and $\boldsymbol{\delta}\mathbf{x}^{(k)}$ is a descent direction for Φ (see Exercise 7.9). In this case, under suitable assumptions on Φ, we obtain a globally convergent method, called *damped Gauss-Newton* method. ■

The Gauss-Newton method is implemented in Program 7.5; `r` and `jr` are function handles associated with the function $\mathbf{R}(\mathbf{x})$ and its Jacobian $\mathrm{J}_{\mathbf{R}}(\mathbf{x})$, respectively, `x0` is the initial vector, while `tol` and `kmax` contain the tolerance for the stopping test and the maximum number of iterations allowed. The output variable `x` contains the computed solution, `err` an estimate of the error at the last iteration and `iter` the number of iterations required to converge.

Program 7.5. gaussnewton: Gauss-Newton method

```
function [x,err,iter]= gaussnewton(r,jr,x0,tol,...
    kmax,varargin)
%GAUSSNEWTON  Solves nonlinear least squares problems
%  [X,ERR,ITER]=GAUSSNEWTON(R,JR,X0,TOL,KMAX)
%  solves the nonlinear least squares by the Gauss-
%  Newton method. R and JR are the function handles
%  associated with the function R and its Jacobian,
%  respectively. X0 is the initial point for the se-
%  quence. TOL is the tolerance for the stopping test,
%  KMAX is the maximum number of allowed iterations.
err=tol+1; k=0; xk=x0(:);
rk=r(xk,varargin{:}); jrk=jr(xk,varargin{:});
while err>tol & k< kmax
[Q,R]=qr(jrk,0); dk=-R \ (Q'*rk);
xk1=xk+dk;
rk1=r(xk1,varargin{:});
jrk1=jr(xk1,varargin{:});
k=k+1;  err=norm(xk1-xk);
xk=xk1; rk=rk1; jrk=jrk1;
end
x=xk; iter=k;
if (k==kmax & err > tol)
```

```
 fprintf(['Gauss-Newton method stopped \n',...
 'without converging to the desired tolerance \n',...
 'because the maximum number of iterations was \n',...
 'reached\n']);
end
```

Example 7.12 Let us consider Problem 7.2 under the form (7.5) (a special case of (7.61)). We use the Gauss-Newton method (7.63), we storage vector $\mathbf{a}$ in the upper part of $\mathbf{x}$ and $\boldsymbol{\sigma}$ in the lower one, yielding

$$r_i(\mathbf{x}) = f(t_i;\mathbf{a},\boldsymbol{\sigma}) - y_i = \sum_{k=1}^{m} f_k(t_i;a_k,\sigma_k) - y_i,$$

$$\frac{\partial r_i}{\partial a_k} = f_k(t_i;a_k,\sigma_k)\frac{t_i-a_k}{\sigma_k^2}, \qquad \frac{\partial r_i}{\partial \sigma_k} = f_k(t_i;a_k,\sigma_k)\left[\frac{(t_i-a_k)^2}{\sigma_k^3} - \frac{1}{2\sigma_k}\right].$$

We generate the n points (t_i, y_i) with $i = 1,\dots,n$, $0 \le t_i \le 10$, by summing 5 Gaussian functions of the form (7.3) taking $\mathbf{a} = [2.3, 3.25, 4.82, 5.3, 6.6]$, $\boldsymbol{\sigma} = [0.2, 0.34, 0.50, 0.23, 0.39]$ and adding a random noise:

```
a=[2.3,3.25,4.82,5.3,6.6]; m=length(a);
sigma=[0.2,0.34,0.50,0.23,0.39];
gaussian=@(t,a,sigma)...
  exp(-((t-a)/(sqrt(2)*sigma)).^2)/(sqrt(pi*2)*sigma);
n=2000; t=linspace(0,10,n)'; y=zeros(n,1);
for k=1:m, y=y+gaussian(t,a(k),sigma(k)); end
y=y+0.05*randn(n,1);
```

We now call Program 7.5 using the following instructions:

```
x0=[2,3,4,5,6,0.3,0.3,0.6,0.3,0.3];
tol=3.e-5; kmax=200;
[x,err,iter]=gaussnewton(@gmmr,@gmmjr,x0,tol,kmax,t,y)
xa=x(1:m); xsigma=x(m+1:end);
h=1./(sqrt(2*pi)*xsigma); w=2*sqrt(log(4))*xsigma;
```

where gmmr and gmmjr are the functions defining $\mathbf{R}(\mathbf{x})$ and $\mathrm{J}_{\mathbf{R}}(\mathbf{x})$, respectively.

```
function [R]=gmmr(x,t,y)
x=x(:);
m=length(x)/2; a=x(1:m); sigma=x(m+1:end);
n=length(t); R=zeros(n,1);
gaussian=@(t,a,sigma)[exp(-((t-a)/(sqrt(2)*sigma))...
     .^2)/(sqrt(pi*2)*sigma)];
for k=1:m, R=R+gaussian(t,a(k),sigma(k)); end, R=R-y;

function [Jr]=gmmjr(x,t,y)
x=x(:); m=length(x)/2; a=x(1:m); sigma=x(m+1:end);
n=length(t); Jr=zeros(n,m*2);
gaussian=@(t,a,sigma)[exp(-((t-a)/(sqrt(2)*sigma))...
    .^2)/(sqrt(pi*2)*sigma)];
fk=zeros(n,m);
for k=1:m, fk(:,k)=gaussian(t,a(k),sigma(k)); end
for k=1:m, Jr(:,k)=(fk(:,k).*(t-a(k))/sigma(k)^2)'; end
for k=1:m, Jr(:,k+m)=(fk(:,k).*((t-a(k)).^2/...
          sigma(k)^3-1/(2*sigma(k))))'; end
```

Convergence is achieved in 22 iterations. The vectors `xa` and `xsigma` contain the approximation of vectors $\mathbf{a}$ and $\boldsymbol{\sigma}$, respectively, while `h` and `w` contain the

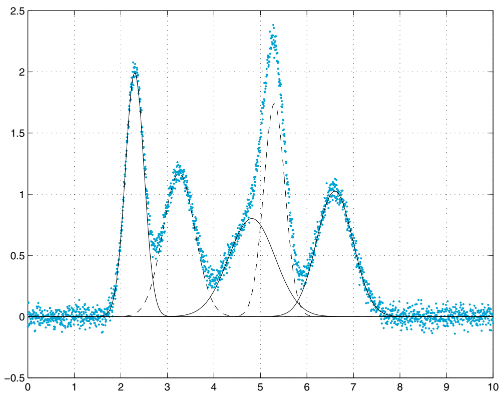

Figure 7.14. I dati (in azzurro) e la soluzione (in nero) dell'Esempio 7.12

height and amplitude, respectively, of the Gaussian functions we are looking for.

We display in Figure 7.14 the points (t_i, y_i) (in blue) representing the signal and the 5 Gaussian functions (7.3) (black lines) built on the obtained numerical solution. This is the case with large residual: as a matter of fact $\Phi(\mathbf{x}^*) = 1.0385e + 03$, $\mathbf{x}^*$ being the solution vector. By a slight change of the initial data, for instance by simply modifying the last component of $\mathbf{x}^{(0)}$ from 0.3 to 0.5, the method would not converge any more. This remark prompts us to a convenient choice of $\mathbf{x}^{(0)}$. ■

7.7.2 Levenberg-Marquardt's method

This is a trust region method for the solution of the minimization problem (7.61). Following algorithm (7.59), after replacing f with Φ (see (7.61)) and $\tilde{f}$ with $\tilde{\Phi}$, at each step k we solve the minimization problem

$$\min_{\mathbf{s}\in\mathbb{R}^n:\ \|\mathbf{s}\|\leq\delta_k} \tilde{\Phi}_k(\mathbf{s})$$

with

$$\tilde{\Phi}_k(\mathbf{s}) = \frac{1}{2}\|\mathbf{R}(\mathbf{x}^{(k)}) + \mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})\mathbf{s}\|^2. \tag{7.65}$$

Note that $\tilde{\Phi}_k(\mathbf{x})$ (7.65) is a quadratic approximation of $\Phi(\mathbf{x})$ around $\mathbf{x}^{(k)}$, obtained by approximating $\mathbf{R}(\mathbf{x})$ with its linear model $\widetilde{\mathbf{R}}_k(\mathbf{x})$ (7.64) (see Exercise 7.11).

Even though $J_{\mathbf{R}}(\mathbf{x})$ does not have full rank, this method is well suited for minimization problems featuring a strong non-linearity or a large residual $\Phi(\mathbf{x}^*) = \frac{1}{2}\|\mathbf{R}(\mathbf{x}^*)\|^2$ in correspondence with a local minimizer $\mathbf{x}^*$.

Since the approximation of the Hessian matrix is the same as for the Gauss-Newton method, the two methods share the same local convergence properties. In particular, should the Levenberg-Marquardt iterations converge, convergence rate is quadratic if the residual is null at local minimizer, linear otherwise.

See Exercises 7.8-7.11.

Let us summarize

1. For the minimization of the function f, the derivative free methods are those using only the functional values of f. They are quite robust in practice even though very little is known about their theoretical convergence;
2. descent methods exploit the knowledge of the function derivatives and compute at each step a descent direction and a steplength, based on line search strategies;
3. descent methods with Newton directions associated with linear search strategies are globally convergent when the matrices $\mathrm{H}(\mathbf{x}^{(k)})$ are positive definite. They feature quadratic convergence rate in proximity of the minimizer. They are well suited for small and medium size problems;
4. descent methods with quasi-Newton directions make use of approximate Hessian matrices H_k at every iteration. When associated with line search strategies, they are globally convergent provided H_k are positive definite, with superlinear convergence order. They too are well suited for small and medium size problems;
5. descent methods with conjugate gradient type descent directions, associated with line search strategies, are globally convergent with linear rate of convergence. They are recommended for large size problems;
6. trust region strategies are more recent and less diffused than line search ones. They replace the objective function with a quadratic approximation and look for a minimizer of the latter in a n-dimensional ball.

7.8 Constrained optimization

In this Section we introduce two simple strategies for the solution of minimization problems with constraints: the penalty method for problems with both equality and inequality constraints and the so-called augmented Lagrangian method for problems featuring equality constraints only.

These two methods allow the solutions of simple problems and provide the basic tools for more robust and complex algorithms that we will not address here (see however [NW06, SY06, BDF+10]).

The constrained optimization problem is formulated as follows: we consider the minimization problem (7.2) for which the domain Ω can be either given by

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : \ h_i(\mathbf{x}) = 0, \ \text{for } i = 1, \dots, p\}, \tag{7.66}$$

where $h_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, \dots, p$, are given functions, or by

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : \ g_j(\mathbf{x}) \geq 0, \ \text{for } j = 1, \dots, q\}, \tag{7.67}$$

where $g_j : \mathbb{R}^n \to \mathbb{R}$ for $j = 1, \dots, q$; p and q are given natural numbers. In the more general case, however, Ω is defined by both equality and inequality constraints, that is

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : h_i(\mathbf{x}) = 0, \ \text{for } i = 1, \dots, p, \ g_j(\mathbf{x}) \geq 0, \ \text{for } j = 1, \dots, q\}. \tag{7.68}$$

The three different situations (7.66), (7.67), and (7.68) undergo a unique notation,

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : h_i(\mathbf{x}) = 0, \ \text{for } i \in \mathcal{I}_h, \ g_j(\mathbf{x}) \geq 0, \ \text{for } j \in \mathcal{I}_g\},$$

for two suitable chosen sets $\mathcal{I}_h$ and $\mathcal{I}_g$, under the convention that $\mathcal{I}_h = \emptyset$ in (7.67) and $\mathcal{I}_g = \emptyset$ in (7.66).

Problem (7.2) can thus be written as

$$\boxed{\begin{array}{l} \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}), \ \text{subject to} \\ h_i(\mathbf{x}) = 0 \quad \forall i \in \mathcal{I}_h, \\ g_j(\mathbf{x}) \geq 0 \quad \forall j \in \mathcal{I}_g \end{array}} \tag{7.69}$$

Everywhere in this section we will assume that f, h_i, and g_j be C^1 functions on $\mathbb{R}^n$.

The points of $\mathbf{x} \in \Omega$ are called *admissibile* (as they fulfill all the constraints); Ω is the set of admissible points.

A point $\mathbf{x}^* \in \Omega \subset \mathbb{R}^n$ is a *global minimizer* for problem (7.2) if

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in \Omega,$$

whereas $\mathbf{x}^*$ is a *local minimizer* for (7.2) if there exists a ball $B_r(\mathbf{x}^*) \subset \mathbb{R}^n$ with radius $r > 0$ and centered at $\mathbf{x}^*$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in B_r(\mathbf{x}^*) \cap \Omega.$$

A constraint is said *active* at $\mathbf{x} \in \Omega$ if it is satisfied with equality at $\mathbf{x} \in \Omega$. According to this definition, active constraints at $\mathbf{x}$ are all the h_i as well as those g_j such that $g_j(\mathbf{x}) = 0$.

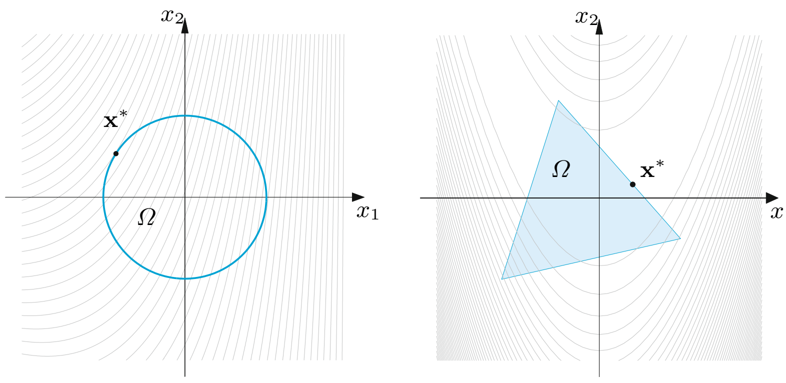

Figure 7.15. The contour lines of the cost function f, the admissibility set Ω and the global minimizer $\mathbf{x}^*$ constrained to Ω. The plot at left is relative to Problem 1 (7.70), that at right to Problem 2 (7.71)

Example 7.13 Consider the following constrained optimization problems: *Problem 1:*

$$\begin{aligned}&\min_{\mathbf{x}\in\mathbb{R}^2} f(\mathbf{x}), \quad \text{with } f(\mathbf{x}) = \frac{3}{5}x_1^2 + \frac{1}{2}x_1x_2 - x_2 + 3x_1,\\ &\text{under the following constraint}\\ &h_1(\mathbf{x}) = x_1^2 + x_2^2 - 1 = 0;\end{aligned} \tag{7.70}$$

Problem 2:

$$\begin{aligned}&\min_{\mathbf{x}\in\mathbb{R}^2} f(\mathbf{x}), \quad \text{with } f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,\\ &\text{under the following constraints:}\\ &g_1(\mathbf{x}) = -34x_1 - 30x_2 + 19 \geq 0,\\ &g_2(\mathbf{x}) = 10x_1 - 5x_2 + 11 \geq 0,\\ &g_3(\mathbf{x}) = 3x_1 + 22x_2 + 8 \geq 0.\end{aligned} \tag{7.71}$$

The contour lines of the two cost functions and the associated set of admissible points Ω are displayed in Figure 7.15. Note that Ω is a closed curve for Problem 1, while it is a closed convex set in $\mathbb{R}^2$ for Problem 2. For both problems there is one active constraint. ■

The Weierstrass theorem guarantees the existence of both the maximum and the minimum for f in Ω when the latter is a non-empty, bounded and closed set. Consequently, problem (7.69) admits a solution.

We recall that a function $f : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$ is *strongly convex* in Ω if $\exists \rho > 0$ such that $\forall \mathbf{x}, \mathbf{y} \in \Omega$ and $\forall \alpha \in [0, 1]$,

$$f(\alpha\mathbf{x} + (1 - \alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1 - \alpha)f(\mathbf{y}) - \alpha(1 - \alpha)\rho\|\mathbf{x} - \mathbf{y}\|^2. \tag{7.72}$$

This reduces to the definition of convexity (7.11) when $\rho = 0$.

Proposition 7.2 (Optimality conditions) *Let $\Omega \subset \mathbb{R}^n$ be a convex set, and $\mathbf{x}^* \in \Omega$ be such that $f \in C^1(B_r(\mathbf{x}^*))$ for a suitable $r > 0$. If $\mathbf{x}^*$ is a local minimizer for (7.2) then*

$$\nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) \geq 0 \qquad \forall \mathbf{x} \in \Omega. \tag{7.73}$$

Moreover, if f is convex in Ω and (7.73) is satisfied, $\mathbf{x}^*$ is a global minimizer for (7.2).
Finally, under the additional requirement for Ω to be closed and f strongly convex, the minimizer for (7.2) is unique.

Let us introduce the *Lagrangian function* associated with problem (7.2)

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x}) - \sum_{j \in \mathcal{I}_g} \mu_j g_j(\mathbf{x}). \tag{7.74}$$

Here $\boldsymbol{\lambda} = (\lambda_i)$ (for $i \in \mathcal{I}_h$) and $\boldsymbol{\mu} = (\mu_j)$ (for $j \in \mathcal{I}_g$) play the role of *Lagrangian multipliers* associated with equality and inequality constraints, respectively. A point $\mathbf{x}^*$ is called a *Karush–Kuhn–Tucker* (KKT) point for $\mathcal{L}$ if there exist $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that the triplet $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ satisfies the following conditions, called *Karush–Kuhn–Tucker conditions*:

$$\begin{aligned}
&\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) - \sum_{j \in \mathcal{I}_g} \mu_j^* \nabla g_j(\mathbf{x}^*) = \mathbf{0}\\
&h_i(\mathbf{x}^*) = 0 \quad \forall i \in \mathcal{I}_h\\
&g_j(\mathbf{x}^*) \geq 0 \quad \forall j \in \mathcal{I}_g\\
&\mu_j^* \geq 0 \quad \forall j \in \mathcal{I}_g\\
&\mu_j^* g_j(\mathbf{x}^*) = 0 \quad \forall j \in \mathcal{I}_g
\end{aligned}$$

For a given point $\mathbf{x}$, the constraints are said to satisfy the LICQ (*linear independence constraint qualification*) condition at $\mathbf{x}$ if the gradients $\nabla h_i(\mathbf{x})$ and $\nabla g_j(\mathbf{x})$ associated with the sole active constraints at $\mathbf{x}$ provide a set of linear independent vectors.

The following result holds (see, e.g., [NW06, Thm. 12.1]).

Theorem 7.1 (First order KKT necessary conditions) *If* $\mathbf{x}^*$ *is a local minimizer for problem* (7.69)*, the functions* f*,* h_i*, and* g_j *are of class* $C^1(\Omega)$*, and the constraints satisfy the LICQ condition at* $\mathbf{x}^*$*, then there exist* $\boldsymbol{\lambda}^*$ *and* $\boldsymbol{\mu}^*$ *such that* $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ *is a KKT point.*

Thanks to this theorem, the local minimizers for (7.69) should be sought for among the KKT points and those points where LICQ condition is not fulfilled.

When the set $\mathcal{I}_g$ is empty (only equality constraints are present) the Lagrangian function reads $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i\in\mathcal{I}_h} \lambda_i h_i(\mathbf{x})$ and the KKT conditions reduce to the classical necessary (*Lagrangian*) conditions

$$\boxed{\begin{aligned} &\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \nabla f(\mathbf{x}^*) - \sum_{i\in\mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0} \\ &h_i(\mathbf{x}^*) = 0 \quad \forall i \in \mathcal{I}_h \end{aligned}} \tag{7.75}$$

Sufficient conditions for a KKT point to be a minimizer for f constrained in Ω would require the knowledge of the Hessian matrix of $\mathcal{L}$ or else an assumption of strict convexity for both f and the constraint functions ([NW06, SY06]).

In general terms, a constrained optimization problem can be written as an unconstrained problem using either the penalized formulation or the augmented Lagrangian formulation, as we will explain in the next two sections.

Remark 7.6 If at a point $\mathbf{x}^*$ that minimizes f no active constraints are present, the Lagrangian function coincides with the cost function f therein, as $\mathcal{I}_h = \emptyset$ and $\mu_j^* = 0$ for all $j \in \mathcal{I}_g$ thanks to the KKT conditions. In this case our problem reduces to an unconstrained minimization problem that can be solved by using the methods discussed in the previous sections. ■

A remarkable instance of constrained optimization is that of *Quadratic Programming*: this is precisely the case where f is a quadratic function, the constraints are expressed by linear functions, thus problem (7.69) can be written under the special form:

$$\boxed{\begin{array}{ll} \min\limits_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}), & f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathrm{A}\mathbf{x} + \mathbf{x}^T\mathbf{b} \\ \text{subject to the constraints} & \mathrm{C}\mathbf{x} - \mathbf{d} = \mathbf{0}, \quad \mathrm{D}\mathbf{x} - \mathbf{e} \geq \mathbf{0} \end{array}} \tag{7.76}$$

where $\mathrm{A} \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$, $\mathrm{C} \in \mathbb{R}^{p\times n}$, $\mathbf{d} \in \mathbb{R}^p$, $\mathrm{D} \in \mathbb{R}^{q\times n}$, $\mathbf{e} \in \mathbb{R}^q$, p, q are suitable positive integers and the notations $\mathbf{v} \geq \mathbf{0}$ means $v_i \geq 0$ for all i. See [Bom10, NW06] for a presentation of Quadratic Programming.

In the special case where constraints are all expressed by equalities, the matrix form of the Langrange conditions (7.75) reads (with obvious choice of notations)

$$\begin{bmatrix} \mathrm{A} & -\mathrm{C}^T \\ \mathrm{C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\mathbf{b} \\ \mathbf{d} \end{bmatrix}. \tag{7.77}$$

If A is symmetric and positive definite on the kernel of C, that is

$$\mathbf{y}^T \mathrm{A}\mathbf{y} > 0 \quad \forall \mathbf{y} \in \mathit{ker}(\mathrm{C}) = \{\mathbf{z} : \mathrm{C}\mathbf{z} = \mathbf{0}\},\ \mathbf{y} \neq \mathbf{0},$$

and assuming that C has full rank, system (7.77) admits a unique solution, thus there exists a unique global minimizer for the cost function defined in (7.76).

A quadratic programming problem can therefore be tackled by solving the linear system (7.77) using one of the methods of Chapter 5.

In general, the matrix M $= [\mathrm{A},\ -\mathrm{C}^T; \mathrm{C},\ 0]$ of (7.77) is not definite, that is it features both positive and negative eigenvalues. Suitable iterative methods for its treatment are Krylov methods like GMRES or Bi-CGStab. See, e.g., [Qua13] and [BGL05].

Example 7.14 To solve Problem 7.4 we note that the cost function defined in (7.7) (the risk) is quadratic, while the constraints read

$$h_1(\mathbf{x}) = 0.6x_1 + x_2 + 1.2x_3 = 1.04, \quad h_2(\mathbf{x}) = x_1 + x_2 + x_3 = 1. \tag{7.78}$$

The former states that the expected return be equal to 10.4%, while the latter establishes that the sum of the fractions invested into the 3 funds be equal to the entire capital. This is a quadratic programming problem that we can rewrite under the form (7.77), where

$$\mathrm{A} = \begin{bmatrix} 0.08 & 0.1 & 0 \\ 0.1 & 0.5 & 0.208 \\ 0 & 0.208 & 1.28 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mathrm{C} = \begin{bmatrix} 0.6 & 1 & 1.2 \\ 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} 1.04 \\ 1 \end{bmatrix}.$$

Matrix C has (maximum) rank equal to 2, its kernel $\mathit{ker}(\mathrm{C}) = \{\mathbf{z} = \alpha[1,-3,2]^T, \alpha \in \mathbb{R}\}$ has dimension 1.

As A is symmetric, we need to verify that it is positive definite on $\mathit{ker}(\mathrm{C})$, that is $\mathbf{z}^T\mathrm{A}\mathbf{z} > 0$ for all $\mathbf{z} = \alpha[1,-3,2]^T$, $\alpha \neq 0$. As a matter of fact, $\mathbf{z}^T\mathrm{A}\mathbf{z} = \alpha^2[1,-3,2]^T\mathrm{A}[1,-3,2] = 6.6040\alpha^2 > 0$. Upon building the matrix M $= [\mathrm{A},\ -\mathrm{C}^T; \mathrm{C},\ 0]$ and the right hand side $\mathbf{f} = [-\mathbf{b}, \mathbf{d}]^T$, we solve (7.77) using the following instructions:

```
A=[0.08 0.1 0; 0.1 0.5 0.208; 0 0.208 1.28]; b=[0;0;0];
C=[0.6 1 1.2;1,1,1]; d=[1.04;1];
M=[A -C'; C, zeros(2)]; f=[-b;d];
xl=M\f
```

and obtain the solution

```
xl =
    0.0606
    0.6183
    0.3211
    0.7883
   -0.4063
```

The first 3 components of `xl` correspond to the 3 fractions of the capital to invest in the 3 funds, whereas the last two components provide the values of the Lagrangian multipliers associated with the constraints. The risk corresponding to this capital splitting is given by the value of the cost function at the point `xl(1:3)` and is approximately equal to 21%. ■

7.8.1 The penalty method

A strategy for solving problem (7.69) consists of turning it into a non-constrained optimization problem for a modified *penalty function*

$$\mathcal{P}_\alpha(\mathbf{x}) = f(\mathbf{x}) + \frac{\alpha}{2}\sum_{i\in\mathcal{I}_h} h_i^2(\mathbf{x}) + \frac{\alpha}{2}\sum_{j\in\mathcal{I}_g}(\max\{-g_j(\mathbf{x}),0\})^2 \qquad (7.79)$$

where $\alpha > 0$ is a parameter to be chosen.

If the given constraints are not fulfilled at the point $\mathbf{x}$, the sums appearing in (7.79) provide a measure of how far $\mathbf{x}$ is from the admissible set Ω. Since in this case $\mathbf{x}$ violates the constraints, choosing large values of α would severely penalize such a violation. Every solution $\mathbf{x}^*$ of (7.69) clearly provides a minimizer of $\mathcal{P}$. Conversely, assuming f, h_i, and g_j regular enough, and denoting with $\mathbf{x}^*(\alpha)$ a minimizer of $\mathcal{P}_\alpha(\mathbf{x})$, it holds ([Ber82])

$$\lim_{\alpha\to\infty} \mathbf{x}^*(\alpha) = \mathbf{x}^*.$$

For $\alpha \gg 1$, $\mathbf{x}^*(\alpha)$ can therefore be regarded as a convenient approximation of $\mathbf{x}^*$. However, since numerical instabilities arising from the minimization of $\mathcal{P}_\alpha(\mathbf{x})$ increase with α, a better strategy consists of solving a sequence of unconstrained minimization problems

$$\mathbf{x}^{(k)} = \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}}\, \mathcal{P}_{\alpha_k}(\mathbf{x}) \qquad (7.80)$$

where $\{\alpha_k\}$ is a monotonically increasing unbounded sequence of parameters (with, e.g., $\alpha_0 = 1$). For every k, α_{k+1} is chosen as a function of α_k and $\mathbf{x}^{(k)}$ provides the initial value for problem (7.80) at the new step $k+1$.

A heuristic approach consists of choosing $\alpha_{k+1} = \delta\alpha_k$ where δ is small (say $\delta \in [1.5, 2]$) if many iterations have been necessary to solve (7.80) at the step k, otherwise one could afford a larger value for δ, say $\delta \simeq 10$.

As a matter of fact, in the course of the first iterations, when using a moderate (not too high) α_k, there is no reason why the solution of (7.80) should resemble that of (7.69). This legitimates the search for an inexact solution of (7.80), differing from the exact one $\mathbf{x}^{(k)}$ by a small enough tolerance ε_k.

The algorithm above is formulated as follows (note that a further tolerance $\bar{\varepsilon} > 0$ is requested to assess the behaviour of the gradient of $\mathcal{P}$ at $\mathbf{x}^{(k)}$).

For given α_0 (tipically, $\alpha_0 = 1$), ε_0 (tipically, $\varepsilon_0 = 1/10$), $\bar{\varepsilon} > 0$ and $\mathbf{x}_0^{(0)} \in \mathbb{R}^n$, for $k = 0, 1, \ldots$ until convergence

```
compute an approximation $\mathbf{x}^{(k)}$ to (7.80) using an initial
data $\mathbf{x}_0^{(k)}$ and a tolerance $\varepsilon_k$ on the stopping criterium;
if $\|\nabla_{\mathbf{x}}\mathcal{P}_{\alpha_k}(\mathbf{x}^{(k)})\| \le \bar{\varepsilon}$
    set $\mathbf{x}^* = \mathbf{x}^{(k)}$ (convergence achieved)
else                                                              (7.81)
    choose $\alpha_{k+1}$ s.t. $\alpha_{k+1} > \alpha_k$
    choose $\varepsilon_{k+1}$ s.t. $\varepsilon_{k+1} < \varepsilon_k$
    set $\mathbf{x}_0^{(k+1)} = \mathbf{x}^{(k)}$
endif
```

This alogorithm is implemented in Program 7.6. `fun` and `grad_fun` are function handles associated with the cost function and its gradient, respectively; `h` and `grad_h` are those associated with the equality constraint functions, while `g` and `grad_g` those associated with inequality constraint functions. When $\mathcal{I}_h$ (resp., $\mathcal{I}_g$) is an empty set, `h` and `grad_h` (resp. `g` and `grad_g`) are empty variables. The output of the functions `grad_fun`, `grad_h` and `grad_g` respectively contain: an n dimensional column vector $\mathbf{y}$ with components $y_i = \partial f/\partial x_i$, an $n \times p$ matrix C whose coefficients are $\mathrm{C}_{ji} = \partial h_i/\partial x_j$, an $n \times q$ matrix G whose entries are $\mathrm{G}_{j\ell} = \partial g_\ell/\partial x_j$. The vector `x0` contains $\mathbf{x}_0^{(0)}$, `tol` and `kmax` the tolerance and the maximum number of iterations for the penalty loop, while `kmaxd` is the maximum number of iterations for the descent method, when the latter is called at every step to solve the unconstrained minimization problem. In this program the tolerance ε_k for the descent method is chosen equal to $1/10$ for $k = 0$ and then reduced at every iteration by a factor 10 until the tolerance $\bar{\varepsilon}$ is reached. The variable `meth` is used to select the unconstrained minimization method: if `meth=0` the MATLAB `fminsearch` function implementing the Nelder and Mead method is chosen, while `meth>1` has the same role played in Program `descent`

7.3 to select the descent method. Finally, if `meth=1`, the Hessian matrix necessary to implement the descent method with Newton's directions is provided as input variable, while it provides H_0 for the BFGS method (7.49) if `meth=2`.

Program 7.6. penalty: penalty method

```
function [x,err,k]=penalty(fun,grad_fun,h,grad_h,...
g,grad_g,x0,tol,kmax,kmaxd,meth,varargin)
% PENALTY Constrained  optimization with penalty
%  [X,ERR,K]=PENALTY(FUN,GRAD_FUN,H,GRAD_H,...
%  G,GRAD_G,X0,TOL,KMAX,KMAXD,METH)
%  computes a local minimizer of the cost function
%  FUN under the constraints H=0 and G>=0, by the
%  penalty method. X0 is the initial point, TOL is
%  the tolerance for the stopping test, KMAX is the
%  maximum number of allowed iterations.
%  GRAD_FUN, GRAD_H, and GRAD_G contain the gradient
%  of FUN, H, and G, respectively. The variables
%  H, G, GRAD_H, and GRAD_G can be set to [], if they
%  are not present. The solution of the corresponding
%  unconstrained minimization problem is performed
%  by calling either Matlab FMINSEARCH function
%  (if METH=0) or DESCENT function (if METH>0).
%  When METH>0, KMAXD and METH contain respectively
%  the maximum number of allowed iterations for the
%  function DESCENT and the choice of the descent
%  directions. When METH>1
%  [X,ERR,K]=PENALTY(FUN,GRAD_FUN,H,GRAD_H,...
%  G,GRAD_G,X0,TOL,KMAX,KMAXD,METH, HESS)
%  is the correct calling instruction.
%  If METH=1 HESS is the function handle associated
%  with the Hessian is required, if METH=2 HESS is a
%  suitable approximation of the Hessian at the step 0.
xk=x0(:); alpha0=1;
if meth==1, hess=varargin{1};
elseif meth==2, hess=varargin{1};
else  hess=[]; end
if ~isempty(h), [nh,mh]=size(h(xk)); end
if ~isempty(g), [ng,mg]=size(g(xk)); else, ng=[]; end
err=tol+1; k=0;
alphak=alpha0; alphak2=alphak/2; told=.1;
while err>tol && k< kmax
P=@(x)Pf(x,fun,g,h,alphak2,ng);
grad_P=@(x)grad_Pf(x,grad_fun,h,g,...
                   grad_h,grad_g,alphak,ng);
if meth==0
options=optimset('TolX',told);
[x,err,kd]=fminsearch(P,xk,options);
err=norm(x-xk);
else
[x,err,kd]=descent(P,grad_P,xk,told,kmaxd,meth,hess);
err=norm(grad_P(x));
end
if kd<kmaxd, alphak=alphak*10; alphak2=alphak/2;
else alphak=alphak*1.5; alphak2=alphak/2; end
k=k+1; xk=x; told=max([tol,told/10]);
end
```

```
end % end of the function penalty
function y=Pf(x,fun,g,h,alphak2,ng)
y=fun(x);
if ~isempty(h), y=y+alphak2*sum((h(x)).^2); end
if ~isempty(g), G=g(x);
for j=1:ng, y=y+alphak2*max([-G(j),0])^2; end
end
end % end of function Pf
function y=grad_Pf(x,grad_fun,h,g,...
                  grad_h,grad_g,alphak,ng)
y=grad_fun(x);
if ~isempty(h), y=y+alphak*grad_h(x)*h(x); end
if ~isempty(g), G=g(x); Gg=grad_g(x);
for j=1:ng
if G(j)<0, y=y+alphak*Gg(:,j)*G(j); end
end, end
end % end of function grad_Pf
```

Example 7.15 Let us solve Problem 2 of the Example 7.13 using Program 7.6. Setting $\mathbf{x}^{(0)} = (1.2, 0.2)$ and tolerance $\overline{\varepsilon} = 10^{-5}$, by the following instructions

```
fun=@(x) 100*(x(2)-x(1).^2).^2+(1-x(1)).^2;
grad_fun=@(x) [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
            200*(x(2)-x(1)^2)];
g=@(x)[-34*x(1)-30*x(2)+19; 10*x(1)-5*x(2)+11;
    3*x(1)+22*x(2)+8];
grad_g=@(x)[-34,10,3;-30,-5,22];
x0=[1.2,.2]; tol=1.e-5; kmax=100; kmaxd=100;
meth=2; hess=eye(2);
[x,err,k]=penalty(fun,grad_fun,[],[],g,grad_g,...
    x0,tol,kmax,kmaxd,meth,hess)
```

after 3 iterations we achieve convergence to the point $(0.41183, 0.16660)$ with a residual on the gradient $\|\nabla_{\mathbf{x}}\mathcal{P}_{\alpha_3}(\mathbf{x}^{(3)})\| \simeq 2.6379 \cdot 10^{-7}$. For the solution of the unconstrained minimization problem we have used the program 7.3 `descent`, more precisely the BFGS method described in Section 7.5.4. The constraints at the minimizers are equal to $g_1(\mathbf{x}) = 2.0036e-04$, $g_2(\mathbf{x}) = 1.4285e+01$ and $g_3(\mathbf{x}) = 1.2901e+01$. ■

Example 7.16 To solve Problem 7.3 with the penalty method, let Ω be the circle centered at the origin with radius 2. Let us triangulate Ω with a grid featuring 49 nodes (the vertices) and $Ne = 72$ triangles, as shown in Figure 7.2, left. The 24 boundary nodes are kept fixed, whereas the coordinates of the 25 internal nodes are collected in a vector $\mathbf{x}$ and represent the problem independent variables. The cost function is

$$f(\mathbf{x}) = \sum_{k=1}^{Ne} \frac{1}{\mu_k(\mathbf{x})} = \sum_{k=1}^{Ne} \frac{\sqrt{3}\|\mathrm{A}_k(\mathbf{x})\mathrm{W}^{-1}\|_F^2}{4\det(\mathrm{A}_k(\mathbf{x}))},$$

where we have used definition (7.6), while the inequality constraints are

$$g_k(\mathbf{x}) = \det(\mathrm{A}_k(\mathbf{x})) - \tau \geq 0, \quad k = 1, \ldots, Ne,$$

with $\tau = 0.10876$ being twice the value of the area of the smallest triangle of the initial grid. The result shown in Figure 7.2, right, has been obtained after

21 iterations of the penalty algorithm, having set $\bar{\epsilon} = 10^{-8}$ for the stopping test. The maximum number of iterations for the Nelder and Mead method has been fixed to 100. ■

Example 7.17 We solve Problem 7.5 by considering the road network of Figure 7.3. There are $11(= n)$ streets s_j and $7(= p)$ cross roads. We assume that at every minute $M = 20$ cars enter and leave the network, that the length of the streets s_j are collected in the vector $\mathbf{L} = (1, 1, 1.5, 1.5, 1.5, 2.2, 1.5, 1.5, 2.2, 1.5, 2.2)$ km (the jth component refers to the length of s_j street, see Figure 7.3), that the maximum speed allowed on every street is 1 km/min and that the maximum car densities on every street are (the ordered components of the vector) $\boldsymbol{\rho}_m = (60, 40, 20, 60, 60, 40, 60, 20, 40, 20, 60)$. Since we are dealing with a constrained minimization problem with both equality and inequality constraints, we can use the penalty method. The associated unconstrained minimization problem will be solved by the descent method with quasi-Newton directions, for which we need to provide the expression of the gradient of the cost function as well as the constraints. By expressing the cost function f and the functions associated with the constraints in terms of the independent variables ρ_j, we have

$$\begin{aligned}
&f(\boldsymbol{\rho}) = \left(\sum_{j=1}^{11} \frac{L_j}{v_{j,m}} \frac{\rho_j}{1 - \rho_j/\rho_{j,m}}\right) / \sum_{j=1}^{11} \rho_j, \\
&h_1(\boldsymbol{\rho}) = M - \sum_{j=1}^{2} v_{j,m}(1 - \rho_j/\rho_{j,m}) \\
&h_2(\boldsymbol{\rho}) = v_{1,m}(1 - \rho_1/\rho_{1,m}) - \sum_{j=3}^{4} v_{j,m}(1 - \rho_j/\rho_{j,m}) \\
&\dots \\
&h_7(\boldsymbol{\rho}) = \sum_{j=9}^{11} v_{j,m}(1 - \rho_j/\rho_{j,m}) - M \\
&g_j(\boldsymbol{\rho}) = \rho_j \qquad\qquad\qquad\qquad j = 1, \dots, 11 \\
&g_{11+j}(\boldsymbol{\rho}) = \rho_{j,m} - \rho_j \qquad\qquad j = 1, \dots, 11.
\end{aligned} \tag{7.82}$$

For a given vector $\boldsymbol{\rho}$, the gradient $\nabla f(\boldsymbol{\rho})$ and the matrices $[\nabla h_1(\boldsymbol{\rho}), \dots, \nabla h_p(\boldsymbol{\rho})]$ and $[\nabla g_1(\boldsymbol{\rho}), \dots, \nabla g_n(\boldsymbol{\rho}), \nabla g_{n+1}(\boldsymbol{\rho}), \dots, \nabla g_{2n}(\boldsymbol{\rho})]$ can be built up through 3 distinct MATLAB functions `grad_fun.m`, `grad_h.m`, and `grad_g.m`. By calling the Program `penalty.m`, using an initial vector with unitary components and the tolerance $\bar{\varepsilon} = 10^{-5}$ for the stopping test, after 5 iterations the method converges to the vector

```
rho_opt =
   15.0246     12.8942      4.6535      9.0594      5.5847      9.4996
    0.5278     -0.0000     11.5494      6.9723      8.4272.
```

Its components, ordered by rows and represented in Figure 7.16, provide the densities ρ_j of the cars on the streets s_j that minimize the cost function. The minimum average time founded is $f(\boldsymbol{\rho}_{opt}) = 2.0782$ min. ■

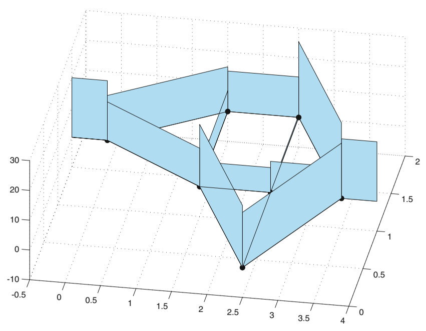

Figure 7.16. The densities ρ_j of the road network Problem 7.5

7.8.2 The augmented Lagrangian method

In this section we address minimization problems with equality constraints only, whence $\mathcal{I}_g = \emptyset$ in (7.69). The function

$$\mathcal{L}_\alpha(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x}) + \frac{\alpha}{2} \sum_{i \in \mathcal{I}_h} h_i^2(\mathbf{x}) \tag{7.83}$$

obtained from (7.74) is called *augmented Lagrangian*; $\alpha > 0$ is a suitable large coefficient to be assigned.

The augmented Lagrangian method is an iterative method that, at the kth iteration, given α_k and $\boldsymbol{\lambda}^{(k)}$, computes

$$\mathbf{x}^{(k)} = \underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}}\, \mathcal{L}_{\alpha_k}(\mathbf{x}, \boldsymbol{\lambda}^{(k)}) \tag{7.84}$$

in such a way that the sequence $\mathbf{x}^{(k)}$ converges to a KKT point (see (7.75)) for the Lagrangian $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x})$.

The initial values α_0 and $\boldsymbol{\lambda}^{(0)}$ are set arbitrarily. The values for the new iterations are generated as follows. The coefficient α_{k+1} is obtained from α_k proceeding as in the penalty method discussed in Section 7.8.1. On its hand, $\boldsymbol{\lambda}^{(k+1)}$ is computed as follows. We compute $\nabla_{\mathbf{x}} \mathcal{L}_{\alpha_k}(\mathbf{x}, \boldsymbol{\lambda}^{(k)})$ and set it to zero, yielding:

$$\nabla_{\mathbf{x}} \mathcal{L}_{\alpha_k}(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}) = \nabla f(\mathbf{x}^{(k)}) - \sum_{i \in \mathcal{I}_h} (\lambda_i^{(k)} - \alpha_k h_i(\mathbf{x}^{(k)})) \nabla h_i(\mathbf{x}^{(k)}) = 0.$$

By comparison with the optimality condition (7.75), we identify the new value of $\lambda_i^{(k+1)}$ as

$$\boxed{\lambda_i^{(k+1)} = \lambda_i^{(k)} - \mu_k h_i(\mathbf{x}^{(k)}).} \tag{7.85}$$

We now obtain $\mathbf{x}^{(k+1)}$ by solving (7.84) with k replaced by $k+1$.

We summarize the algorithm as follows: given α_0 (tipically, $\alpha_0 = 1$), ε_0 (tipically, $\varepsilon_0 = 1/10$), $\overline{\varepsilon} > 0$, $\mathbf{x}_0^{(0)} \in \mathbb{R}^n$ and $\boldsymbol{\lambda}_0^{(0)} \in \mathbb{R}^p$ for $k = 0, 1, \ldots$ until convergence

```
compute an approximation $\mathbf{x}^{(k)}$ to (7.84) using an initial
data $\mathbf{x}_0^{(k)}$ and a tolerance $\varepsilon_k$ on the stopping criterium;
if $\|\nabla_{\mathbf{x}}\mathcal{L}_{\alpha_k}(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)})\| \leq \overline{\varepsilon}$
    set $\mathbf{x}^* = \mathbf{x}^{(k)}$ (convergence achieved)
else
    compute $\lambda_i^{(k+1)}$ by (7.85)
    choose $\alpha_{k+1}$ s.t. $\alpha_{k+1} > \alpha_k$
    choose $\varepsilon_{k+1}$ s.t. $\varepsilon_{k+1} < \varepsilon_k$
    set $\mathbf{x}_0^{(k+1)} = \mathbf{x}^{(k)}$
endif                                                          (7.86)
```

This algorithm is implemented in Program 7.7. Apart from `lambda0` that contains the initial vector $\boldsymbol{\lambda}^{(0)}$ of the Lagrange multipliers, all the other input and output parameters coincide with those of Program 7.6.

Program 7.7. auglagrange: augmented Lagrangian method

```
function [x,err,k]=auglagrange(fun,grad_fun,h,grad_h,...
    x0,lambda0,tol,kmax,kmaxd,meth,varargin)
% AUGLAGRANGE Constrained optimization
%  [X,ERR,K]=AUGLAGRANGE(FUN,GRAD_FUN,H,GRAD_H,...
%  X0,LAMBDA0,TOL,KMAX,KMAXD,METH)
%  computes a local minimizer of the cost function
%  FUN under the constraints H=0, by the augmented
%  Lagrangian method. X0 is the initial point, TOL
%  the tolerance for the stopping test, KMAX the
%  maximum number of allowed iterations.
%  GRAD_FUN and GRAD_H contain the gradient of FUN
%  and H respectively. The solution of the associated
%  unconstrained minimization problem is performed
%  by calling either the Matlab FMINSEARCH function
%  (if METH=0) or the DESCENT function (if METH>0).
%  When METH>0, KMAXD and METH contain respectively
%  the maximum number of allowed iterations for the
%  function DESCENT and the choice of the descent
%  directions. When METH>1
%  [X,ERR,K]=AUGLAGRANGE(FUN,GRAD_FUN,H,GRAD_H,...
%  X0,LAMBDA0,TOL,KMAX,KMAXD,METHi, HESS)
%  is the correct calling instruction.
%  If METH=1 HESS is the function handle associated
%  with the Hessian is required, if METH=2 HESS is a
%  suitable approximation of the Hessian at the step 0.
alpha0=1;
if meth==1, hess=varargin{1};
elseif meth==2, hess=varargin{1};
else, hess=[]; end
err=tol+1; k=0; xk=x0(:); lambdak=lambda0(:);
if ~isempty(h), [nh,mh]=size(h(xk)); end
alphak=alpha0; alphak2=alphak/2; told=0.1;
while err>tol && k< kmax
L=@(x)Lf(x,fun,lambdak,alphak2,h);
grad_L=@(x)grad_Lf(x,grad_fun,lambdak,alphak,h,grad_h);
if meth==0
options=optimset('TolX',told);
[x,err,kd]=fminsearch(L,xk,options);
err=norm(x-xk);
else
[x,err,kd]=descent(L,grad_L,xk,told,kmaxd,meth,hess);
err=norm(grad_L(x));
end
lambdak=lambdak-alphak*h(x);
if kd<kmaxd, alphak=alphak*10; alphak2=alphak/2;
else alphak=alphak*1.5; alphak2=alphak/2; end
k=k+1; xk=x; told=max([tol,told/10]);
end
end % end auglagrange
function y=Lf(x,fun,lambdak,alphak2,h)
y=fun(x);
if ~isempty(h)
y=y-sum(lambdak'*h(x))+alphak2*sum((h(x)).^2); end
end % end function Lf
function y=grad_Lf(x,grad_fun,lambdak,alphak,h,grad_h)
y=grad_fun(x);
if ~isempty(h)
   y=y+grad_h(x)*(alphak*h(x)-lambdak); end
end % end function grad_Lf
```

Example 7.18 To solve Problem 1 of Example 7.13 we use the augmented Lagrangian method by calling Program 7.7 as follows:

```
fun=@(x)0.6*x(1).^2+0.5*x(2).*x(1)-x(2)+3*x(1);
grad_fun=@(x) [1.2*x(1)+0.5*x(2)+3; 0.5*x(1)-1];
h=@(x)x(1).^2+x(2).^2-1;
grad_h=@(x)[2*x(1); 2*x(2)];
x0=[1.2,.2]; tol=1.e-5; kmax=500; kmaxd=100;
p=1; % number of equality constraints
lambda0=rand(p,1); meth=2; hess=eye(2);
[xmin,err,k]=auglagrange(fun,grad_fun,h,grad_h,...
     x0,lambda0,tol,kmax,kmax,meth,hess)
```

We have set the tolerance equal to 10^{-5} for the stopping test, and solved the associated unconstrained minimization problem by quasi-Newton descent directions (therefore setting `meth=2` and `hess=eye(2)`).
After 5 iterations we reach convergence to the point

```
xmin =
    -8.454667252699469e-01
     5.340281045624525e-01
```

The constraint function h at this point is equal to `resh=5.6046-10`. The solution to this problem is reported in Figure 7.15, left.

Should we use the penalty method instead, leaving unchanged all the other settings, we would obtain convergence after 6 iterations to the point

```
xmin =
    -8.454715822058602e-01
     5.340328869427682e-01
```

with the value of h therein equal to `resh=1.3320e-04`. The latter value is larger by 6 orders of magnitude than the one obtained using the augmented Lagrangian method. Since this behaviour occurs quite often, the augmented Lagrangian method is in general preferable in case of minimization problems featuring only equality constraints. ■

See Exercises 7.12-7.14.

Let us summarize

1. For a constrained minimization problem, the minimizers should be sought for among the KKT points associated with the Lagrangian function, or among the points where the LICQ condition fails to be satisfied;
2. a quadratic programming problem is one for which the cost function is quadratic and the constraints are linear. Under suitable assumptions on the matrix associated with the quadratic terms and on the constraint functions, it admits a unique minimizer that can be obtained by solving a linear system;
3. a constrained minimization problem can be turned into an unconstrained one using a suitable penalty function. The corresponding penalized problem can be severely ill-conditioned because of the large value that is tipically assigned to the penalty parameter;
4. the augmented Lagrangian method is a penalty method suitable for the search of KKT points.

7.9 What we haven't told you

Large scale optimization problems are especially demanding in terms of computational time and storage requirements. Both line search and trust region methods require the factorization of the Hessian matrix or the construction of suitable approximations that might be dense even when the Hessian is sparse. Special variants featuring limited memory of the methods illustrated above have been developed, based on Conjugate Gradient and Lanczos iterations. See for instance [Ste83, NW06, GOT05].

A classical and efficient method for the solution of constrained minimization problems is the *Sequential Quadratic Programming* (SQP), which transforms a minimization problem with cost function f and arbitrary constraints into the successive solution of quadratic programming problems. At every iteration, f is approximated by a quadratic function like (7.76), then one looks for the KKT points of the associated Lagrangian function (see for instance [Fle10], [NW06]).

In case of inequality constraints solely, the *barrier methods* represent an alternative to penalty methods: the cost function is modified by adding a function depending on the inequality constraints which inhibits an admissible point $\mathbf{x} \in \Omega$ to generate a successive point which is not admissible. This barrier function is defined only at the interior of the admissible set and is unbounded on the boundary of Ω. These methods require the initial point to be admissible, a condition hard to be fulfilled. For a more in depth presentation we refer to [Ter10].

7.10 Exercises

Exercise 7.1 Compute the minimum of $f(x) = (x-1)e^{-x^2}$ using the golden section method with or without quadratic interpolation.

Exercise 7.2 Two ships leave the harbour at the same time and move along trajectories respectively described by the parametric curves

$$\gamma_1(t) = \begin{cases} 7\cos\left(\frac{t}{3}+\frac{\pi}{2}\right)+5 \\ -4\sin\left(\frac{t}{3}+\frac{\pi}{2}\right)-3 \end{cases}, \qquad \gamma_2(t) = \begin{cases} 6\cos\left(\frac{t}{6}-\frac{\pi}{3}\right)-4 \\ -6\sin\left(\frac{t}{3}-\frac{\pi}{3}\right)+5 \end{cases}.$$

The parameter $t > 0$ represents the time (in hours), whereas the positions are expressed in miles with respect to the origin of the reference framework. Find the minimum distance between the two ships along all their motion.

Exercise 7.3 Compute the global minima of $f(\mathbf{x}) = x_1^4 + x_2^4 + x_1^3 + 3x_1x_2^2 - 3x_1^2 - 3x_2^2 + 10$ using the Nelder and Mead method.

Exercise 7.4 By setting $x^{(0)} = 3/2$, $d^{(k)} = (-1)^{k+1}$, and $\alpha_k = 2 + 2/3^{k+1}$, show that the descent method generates a sequence that does not converge to the minimizer of $f(x) = x^4$ even though $\{f(x^{(k)})\}$ is monotonically decreasing. Show moreover that the steplengths α_k do not fulfill the Wolfe conditions (7.43).

Exercise 7.5 Show that the same conclusions drawn for the previous exercise hold by taking $x^{(0)} = -2$, $d^{(k)} = 1$, and $\alpha_k = 3^{-(k+1)}$.

Exercise 7.6 Approximate the minimizer of the Rosenbrock function defined in Example 7.3 using the descent method with different choices of the descent directions (7.35)–(7.38). Set $\mathbf{x}^{(0)} = (-1.2, 1)$ and $\varepsilon = 10^{-8}$ as tolerance for the stopping test, plot the convergence histories for the different choices of the descent directions and comment on the efficiency of the different methods.

Exercise 7.7 Compute the minimum of $f(\mathbf{x}) = (x_1^2 - x_1^3 x_2 - 2x_2 + 2x_1 x_2^2)^2 + (3 - x_1 x_2)^2$ using the BFGS method and the trust region method with quasi-Newton directions to solve problem (7.54). As initial guess try $\mathbf{x}^{(0)} = (2, -1)$, or $\mathbf{x}^{(0)} = (2, 1)$, or else $\mathbf{x}^{(0)} = (-1, -1)$.

Exercise 7.8 Show that the Gauss-Newton method (7.63) can be reformulated as follows: for $k = 0, 1, \dots$ until convergence, solve

$$\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{2}\|\widetilde{\mathbf{R}}_k(\mathbf{x})\|^2 \text{ with } \widetilde{\mathbf{R}}_k(\mathbf{x}) \text{ defined in (7.64).} \tag{7.87}$$

Exercise 7.9 Consider the Gauss-Newton method of Section 7.7.1. Show that if $\mathrm{J}_{\mathbf{R}}(\mathbf{x}^{(k)})$ has full rank, then the solution $\boldsymbol{\delta}\mathbf{x}^{(k)}$ of $(7.63)_1$ is a descent direction for the function f defined in (7.61).

Exercise 7.10 Consider the table

t_i	0.055	0.181	0.245	0.342	0.419	0.465	0.593	0.752
y_i	2.80	1.76	1.61	1.21	1.25	1.13	0.52	0.28

and find the least squares approximation $\phi(t) = x_1 + x_2 t + x_3 t^2 + x_4 e^{-x_5 t}$ (with unknown coefficients $x_1, x_2, \dots, x_5$) of the data set (t_i, y_i).

Exercise 7.11 Prove that the function $\tilde{\varPhi}_k(\mathbf{x})$ defined in (7.65) is a quadratic approximation of $\varPhi$ obtained by approximating $\mathbf{R}(\mathbf{x})$ with its linear model (7.64).

Exercise 7.12 We look for the optimal positioning of the warehouse that has to provide goods to three selling points whose coordinates are reported in the table below:

Selling point	coordinates (x_i, y_i) (km)	annual deliveries (units)
1	(6,3)	140
2	(-9,9)	134
3	(-8,-5)	88

The warehouse must be allocated within the region $\varOmega = \{(x, y) \in \mathbb{R}^2 : \; y \leq x - 10\}$.

Exercise 7.13 Compute the minimum of the Quadratic Programming problem (7.76) featuring only equality constraints, with

$$\mathrm{A} = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}, \qquad \mathrm{C} = \begin{bmatrix} 2 & -2 & 0 \\ 2 & 1 & -3 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Exercise 7.14 A material point moves with speed $v(x, y) = (\sin(\pi x y) + 1)(2x + 3y + 4)$ along an elliptic trajectory whose equation is $x^2/4 + y^2 = 1$. Find the maximum value of the velocity reached by the point as well as the corresponding position.