Chapter 5
Newton's Method

5.1 ▪ Pure Newton's Method

In the previous chapter we considered the unconstrained minimization problem

$$\min\{f(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^n\},$$

where we assumed that f is continuously differentiable. We studied the gradient method which only uses first order information, namely information on the function values and gradients. In this chapter we assume that f is twice continuously differentiable, and we will present a *second order* method, namely a method that uses, in addition to the information on function values and gradients, evaluations of the Hessian matrices. We will concentrate on Newton's method. The main idea of Newton's method is the following. Given an iterate $\mathbf{x}_k$, the next iterate $\mathbf{x}_{k+1}$ is chosen to minimize the quadratic approximation of the function around $\mathbf{x}_k$:

$$\mathbf{x}_{k+1} = \operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^n} \left\{ f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{x}-\mathbf{x}_k) + \frac{1}{2}(\mathbf{x}-\mathbf{x}_k)^T \nabla^2 f(\mathbf{x}_k)(\mathbf{x}-\mathbf{x}_k) \right\}. \quad (5.1)$$

The above update formula is not well-defined unless we further assume that $\nabla^2 f(\mathbf{x}_k)$ is positive definite. In that case, the unique minimizer of the minimization problem (5.1) is the unique stationary point:

$$\nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = \mathbf{0},$$

which is the same as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k). \quad (5.2)$$

The vector $-(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k)$ is called the *Newton direction*, and the algorithm induced by the update formula (5.2) is called *the pure Newton's method*. Note that when $\nabla^2 f(\mathbf{x}_k)$ is positive definite for any k, pure Newton's method is essentially a scaled gradient method, and Newton's directions are descent directions.

Pure Newton's Method

Input: $\varepsilon > 0$ - tolerance parameter.

Initialization: Pick $\mathbf{x}_0 \in \mathbb{R}^n$ arbitrarily.
General step: For any $k = 0, 1, 2, \ldots$ execute the following steps:

(a) Compute the Newton direction $\mathbf{d}_k$, which is the solution to the linear system $\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$.

(b) Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$.

(c) If $\|\nabla f(\mathbf{x}_{k+1})\| \le \varepsilon$, then STOP, and $\mathbf{x}_{k+1}$ is the output.

At the very least, Newton's method requires that $\nabla^2 f(\mathbf{x})$ is positive definite for every $\mathbf{x} \in \mathbb{R}^n$, which in particular implies that there exists a unique optimal solution $\mathbf{x}^*$. However, this is not enough to guarantee convergence, as the following example illustrates.

Example 5.1. Consider the function $f(x) = \sqrt{1+x^2}$ defined over the real line. The minimizer of f over $\mathbb{R}$ is of course $x = 0$. The first and second derivatives of f are

$$f'(x) = \frac{x}{\sqrt{1+x^2}}, \qquad f''(x) = \frac{1}{(1+x^2)^{3/2}}.$$

Therefore, (pure) Newton's method has the form

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} = x_k - x_k(1+x_k^2) = -x_k^3.$$

We therefore see that for $|x_0| \ge 1$ the method diverges and that for $|x_0| < 1$ the method converges very rapidly to the correct solution $x^* = 0$. ∎

Despite the fact that a lot of assumptions are required to be made in order to guarantee the convergence of the method, Newton's method does have one very attractive feature: under certain assumptions one can prove local *quadratic* rate of convergence, which means that near the optimal solution the errors $e_k = \|\mathbf{x}_k - \mathbf{x}^*\|$ (where $\mathbf{x}^*$ is the unique optimal solution) satisfy the inequality $e_{k+1} \le M e_k^2$ for some positive $M > 0$. This property essentially means that the number of accuracy digits is doubled at each iteration.

Theorem 5.2 (quadratic local convergence of Newton's method). *Let f be a twice continuously differentiable function defined over $\mathbb{R}^n$. Assume that*

- *there exists $m > 0$ for which $\nabla^2 f(\mathbf{x}) \succeq m\mathbf{I}$ for any $\mathbf{x} \in \mathbb{R}^n$,*
- *there exists $L > 0$ for which $\|\nabla^2 f(\mathbf{x}) - \nabla^2 f(\mathbf{y})\| \le L\|\mathbf{x} - \mathbf{y}\|$ for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$.*

Let $\{\mathbf{x}_k\}_{k\ge 0}$ be the sequence generated by Newton's method, and let $\mathbf{x}^*$ be the unique minimizer of f over $\mathbb{R}^n$. Then for any $k = 0, 1, \ldots$ the inequality

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \le \frac{L}{2m}\|\mathbf{x}_k - \mathbf{x}^*\|^2 \tag{5.3}$$

holds. In addition, if $\|\mathbf{x}_0 - \mathbf{x}^*\| \le \frac{m}{L}$, then

$$\|\mathbf{x}_k - \mathbf{x}^*\| \le \frac{2m}{L}\left(\frac{1}{2}\right)^{2^k}, \quad k = 0, 1, 2, \ldots. \tag{5.4}$$

Proof. Let k be a nonnegative integer. Then

$$\begin{aligned}
\mathbf{x}_{k+1}-\mathbf{x}^* &= \mathbf{x}_k-(\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k)-\mathbf{x}^* \\
&\overset{\nabla f(\mathbf{x}^*)=0}{=} \mathbf{x}_k-\mathbf{x}^*+(\nabla^2 f(\mathbf{x}_k))^{-1}(\nabla f(\mathbf{x}^*)-\nabla f(\mathbf{x}_k)) \\
&= \mathbf{x}_k-\mathbf{x}^*+(\nabla^2 f(\mathbf{x}_k))^{-1}\int_0^1 [\nabla^2 f(\mathbf{x}_k+t(\mathbf{x}^*-\mathbf{x}_k))](\mathbf{x}^*-\mathbf{x}_k)dt \\
&= (\nabla^2 f(\mathbf{x}_k))^{-1}\int_0^1 \left[\nabla^2 f(\mathbf{x}_k+t(\mathbf{x}^*-\mathbf{x}_k))-\nabla^2 f(\mathbf{x}_k)\right](\mathbf{x}^*-\mathbf{x}_k)dt.
\end{aligned}$$

Since $\nabla^2 f(\mathbf{x}_k)\succeq m\mathbf{I}$, it follows that $\|(\nabla^2 f(\mathbf{x}_k))^{-1}\|\le\frac{1}{m}$. Hence,

$$\begin{aligned}
\|\mathbf{x}_{k+1}-\mathbf{x}^*\| &\le \|(\nabla^2 f(\mathbf{x}_k))^{-1}\|\left\|\int_0^1 \left[\nabla^2 f(\mathbf{x}_k+t(\mathbf{x}^*-\mathbf{x}_k))-\nabla^2 f(\mathbf{x}_k)\right](\mathbf{x}^*-\mathbf{x}_k)dt\right\| \\
&\le \|(\nabla^2 f(\mathbf{x}_k))^{-1}\|\int_0^1 \left\|\left[\nabla^2 f(\mathbf{x}_k+t(\mathbf{x}^*-\mathbf{x}_k))-\nabla^2 f(\mathbf{x}_k)\right](\mathbf{x}^*-\mathbf{x}_k)\right\|dt \\
&\le \|(\nabla^2 f(\mathbf{x}_k))^{-1}\|\int_0^1 \left\|\nabla^2 f(\mathbf{x}_k+t(\mathbf{x}^*-\mathbf{x}_k))-\nabla^2 f(\mathbf{x}_k)\right\|\cdot\|\mathbf{x}^*-\mathbf{x}_k\|dt \\
&\le \frac{L}{m}\int_0^1 t\|\mathbf{x}_k-\mathbf{x}^*\|^2 dt=\frac{L}{2m}\|\mathbf{x}_k-\mathbf{x}^*\|^2.
\end{aligned}$$

We will prove inequality (5.4) by induction on k. Note that for $k=0$, we assumed that

$$\|\mathbf{x}_0-\mathbf{x}^*\|\le\frac{m}{L},$$

so in particular

$$\|\mathbf{x}_0-\mathbf{x}^*\|\le\frac{2m}{L}\left(\frac{1}{2}\right)^{2^0},$$

establishing the basis of the induction. Assume that (5.4) holds for an integer k, that is, $\|\mathbf{x}_k-\mathbf{x}^*\|\le\frac{2m}{L}(\frac{1}{2})^{2^k}$; we will show it holds for $k+1$. Indeed, by (5.3) we have

$$\|\mathbf{x}_{k+1}-\mathbf{x}^*\|\le\frac{L}{2m}\|\mathbf{x}_k-\mathbf{x}^*\|^2\le\frac{L}{2m}\left(\frac{2m}{L}\left(\frac{1}{2}\right)^{2^k}\right)^2=\frac{2m}{L}\left(\frac{1}{2}\right)^{2^{k+1}},$$

proving the desired result. □

A very naive implementation of Newton's method in MATLAB is given below.

```
function x=pure_newton(f,g,h,x0,epsilon)
% Pure Newton's method
%
% INPUT
% ==============
% f ..........  objective function
% g ..........  gradient of the objective function
% h ..........  Hessian of the objective function
% x0........... initial point
% epsilon ..... tolerance parameter
% OUTPUT
% ==============
```

```
% x - solution obtained by Newton's method (up to some tolerance)

if (nargin<5)
    epsilon=1e-5;
end
x=x0;
gval=g(x);
hval=h(x);
iter=0;
while ((norm(gval)>epsilon)&&(iter<10000))
    iter=iter+1;
    x=x-hval\gval;
    fprintf('iter= %2d f(x)=%10.10f\n',iter,f(x))
    gval=g(x);
    hval=h(x);
end
if (iter==10000)
    fprintf('did not converge')
end
```

Note that the above implementation does not check the positive definiteness of the Hessian, and it essentially assumes implicitly that this property holds. As already mentioned, Newton's method requires quite a lot of assumptions in order to guarantee convergence, and hence the described implementation includes a divergence criteria (10000 iterations).

Example 5.3. Consider the minimization problem

$$\min_{x,y} 100x^4 + 0.01y^4,$$

whose optimal solution is obviously $(x,y) = (0,0)$. This is a rather poorly scaled problem, and indeed the gradient method with initial vector $\mathbf{x}_0 = (1,1)^T$ and parameters $(s,\alpha,\beta,\varepsilon) = (1,0.5,0.5,10^{-6})$ converges after the huge amount of 14612 iterations:

```
>> f=@(x)100*x(1)^4+0.01*x(2)^4;
>> g=@(x)[400*x(1)^3;0.04*x(2)^3];
>> [x,fun_val]=gradient_method_backtracking(f,g,[1;1],1,0.5,0.5,1e-6)
iter_number =   1 norm_grad = 90.513620 fun_val = 13.799181
iter_number =   2 norm_grad = 32.381098 fun_val = 3.511932
iter_number =   3 norm_grad = 11.472585 fun_val = 0.887929
      :                    :                     :
iter_number = 14611 norm_grad = 0.000001 fun_val = 0.000000
iter_number = 14612 norm_grad = 0.000001 fun_val = 0.000000
```

Invoking pure Newton's method, we obtain convergence after only 17 iterations:

```
>> h=@(x)[1200*x(1)^2,0;0,0.12*x(2)^2];
>> pure_newton(f,g,h,[1;1],1e-6)
iter=  1 f(x)=19.7550617284
iter=  2 f(x)=3.9022344155
iter=  3 f(x)=0.7708117364
       :            :
iter= 15 f(x)=0.0000000027
iter= 16 f(x)=0.0000000005
iter= 17 f(x)=0.0000000001
```

Note that the basic assumptions required for the convergence of Newton's method as described in Theorem 5.2 are not satisfied. The Hessian is always positive semidefinite, but it is not always positive definite and does not satisfy a Lipschitz property. ∎

The previous example exhibited convergence even when the basic underlying assumptions of Theorem 5.2 are not satisfied. However, in general, convergence is unfortunately not guaranteed in the absence of these very restrictive assumptions.

Example 5.4. Consider the minimization problem

$$\min_{x_1,x_2} \sqrt{x_1^2+1}+\sqrt{x_2^2+1},$$

whose optimal solution is $\mathbf{x}=0$. The Hessian of the function is

$$\nabla^2 f(\mathbf{x})=\begin{pmatrix} \frac{1}{(x_1^2+1)^{3/2}} & 0 \\ 0 & \frac{1}{(x_2^2+1)^{3/2}} \end{pmatrix} \succ 0.$$

Note that despite the fact that the Hessian is positive definite, there does not exist an $m>0$ for which $\nabla^2 f(\mathbf{x}) \succeq m\mathbf{I}$. This violation of the basic assumptions can be seen practically. Indeed, if we employ Newton's method with initial vector $\mathbf{x}_0=(1,1)^T$ and tolerance parameter $\varepsilon=10^{-8}$ we obtain convergence after 37 iterations:

```
>> f=@(x)sqrt(1+x(1)^2)+sqrt(1+x(2)^2);
>> g=@(x)[x(1)/sqrt(x(1)^2+1);x(2)/sqrt(x(2)^2+1)];
>> h=@(x)diag([1/(x(1)^2+1)^1.5,1/(x(2)^2+1)^1.5]);
>> pure_newton(f,g,h,[1;1],1e-8)
iter=  1 f(x)=2.8284271247
iter=  2 f(x)=2.8284271247
   :                :
iter= 30 f(x)=2.8105247315
iter= 31 f(x)=2.7757389625
iter= 32 f(x)=2.6791717153
iter= 33 f(x)=2.4507092918
iter= 34 f(x)=2.1223796622
iter= 35 f(x)=2.0020052756
iter= 36 f(x)=2.0000000081
iter= 37 f(x)=2.0000000000
```

Note that in the first 30 iterations the method is almost stuck. On the other hand, the gradient method with backtracking and parameters $(s,\alpha,\beta)=(1,0.5,0.5)$ converges after only 7 iterations:

```
>> [x,fun_val]=gradient_method_backtracking(f,g,[1;1],1,0.5,0.5,1e-8);
iter_number =   1 norm_grad = 0.397514 fun_val = 2.084022
iter_number =   2 norm_grad = 0.016699 fun_val = 2.000139
iter_number =   3 norm_grad = 0.000001 fun_val = 2.000000
iter_number =   4 norm_grad = 0.000001 fun_val = 2.000000
iter_number =   5 norm_grad = 0.000000 fun_val = 2.000000
iter_number =   6 norm_grad = 0.000000 fun_val = 2.000000
iter_number =   7 norm_grad = 0.000000 fun_val = 2.000000
```

If we start from the more distant point $(10,10)^T$. The situation is much more severe. The gradient method with backtracking converges after 13 iterations:

```
>> [x,fun_val]=gradient_method_backtracking(f,g,[10;10],1,0.5,0.5,1e-8);
iter_number =   1 norm_grad = 1.405573 fun_val = 18.120635
iter_number =   2 norm_grad = 1.403323 fun_val = 16.146490
     :                :                      :
iter_number =  12 norm_grad = 0.000049 fun_val = 2.000000
iter_number =  13 norm_grad = 0.000000 fun_val = 2.000000
```

Newton's method, on the other hand, diverges:

```
>> pure_newton(f,g,h,[10;10],1e-8);
iter=  1 f(x)=2000.0009999997
iter=  2 f(x)=1999999999.9999990000
iter=  3 f(x)=199999999999999730000000000.0000000
iter=  4 f(x)=199999999999999230000000000000000000....
iter=  5 f(x)=       Inf
```

∎

As can be seen in the last example, pure Newton's method does not guarantee descent of the generated sequence of function values even when the Hessian is positive definite. This drawback can be rectified by introducing a stepsize chosen by a certain line search procedure, leading to the so-called *damped Newton's method*.

5.2 ▪ Damped Newton's Method

Below we describe the damped Newton's method with a backtracking stepsize strategy. Of course, other stepsize strategies may be used.

Damped Newton's Method

Input: $\alpha, \beta \in (0,1)$ - parameters for the backtracking procedure.
$\varepsilon > 0$ - tolerance parameter.

Initialization: Pick $\mathbf{x}_0 \in \mathbb{R}^n$ arbitrarily.
General step: For any $k = 0, 1, 2, \ldots$ execute the following steps:

(a) Compute the Newton direction $\mathbf{d}_k$, which is the solution to the linear system $\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$.

(b) Set $t_k = 1$. While

$$f(\mathbf{x}_k) - f(\mathbf{x}_k + t_k \mathbf{d}_k) < -\alpha t_k \nabla f(\mathbf{x}_k)^T \mathbf{d}_k$$

set $t_k := \beta t_k$.

(c) $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \mathbf{d}_k$.

(c) If $\|\nabla f(\mathbf{x}_{k+1})\| \le \varepsilon$, then STOP, and $\mathbf{x}_{k+1}$ is the output.

A MATLAB implementation of the method is given below.

```
function x=newton_backtracking(f,g,h,x0,alpha,beta,epsilon)
% Newton's method with backtracking
%
% INPUT
%=========================================
% f ......... objective function
% g ......... gradient of the objective function
% h ......... hessian of the objective function
% x0......... initial point
% alpha ..... tolerance parameter for the stepsize selection strategy
% beta ...... the proportion in which the stepsize is multiplied
%             at each backtracking step (0<beta<1)
% epsilon ... tolerance parameter for stopping rule
% OUTPUT
%=========================================
% x ......... optimal solution (up to a tolerance)
%             of min f(x)
% fun_val ... optimal function value

x=x0;
gval=g(x);
hval=h(x);
d=hval\gval;
iter=0;
while ((norm(gval)>epsilon)&&(iter<10000))
    iter=iter+1;
    t=1;
    while(f(x-t*d)>f(x)-alpha*t*gval'*d)
      t=beta*t;
    end
    x=x-t*d;
    fprintf('iter= %2d f(x)=%10.10f\n',iter,f(x))
    gval=g(x);
    hval=h(x);
    d=hval\gval;
end

if (iter==10000)
    fprintf('did not converge\n')
end
```

Example 5.5. Continuing Example 5.4, invoking Newton's method with the same initial vector $\mathbf{x}_0 = (10, 10)^T$ that caused the divergence of pure Newton's method, results in convergence after 17 iterations.

```
>> newton_backtracking(f,g,h,[10;10],0.5,0.5,1e-8);
iter=  1 f(x)=4.6688169339
iter=  2 f(x)=2.4101973721
iter=  3 f(x)=2.0336386321
   :              :
iter= 16 f(x)=2.0000000005
iter= 17 f(x)=2.0000000000
```

■

5.3 ▪ The Cholesky Factorization

An important issue that naturally arises when employing Newton's method is the one of validating whether the Hessian matrix is positive definite, and if it is, then another issue

is how to solve the linear system $\nabla^2 f(\mathbf{x}_k)\mathbf{d} = -\nabla f(\mathbf{x}_k)$. These two issues are resolved by using the Cholesky factorization, which we briefly recall in this section.

Given an $n \times n$ positive definite matrix $\mathbf{A}$, a *Cholesky factorization* is a factorization of the form

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T,$$

where $\mathbf{L}$ is a lower triangular $n \times n$ matrix whose diagonal is positive. Given a Cholesky factorization, the task of solving a linear system of equations of the form $\mathbf{A}\mathbf{x} = \mathbf{b}$ can be easily done by the following two steps:

A. Find the solution $\mathbf{u}$ of $\mathbf{L}\mathbf{u} = \mathbf{b}$.

B. Find the solution $\mathbf{x}$ of $\mathbf{L}^T\mathbf{x} = \mathbf{u}$.

Since $\mathbf{L}$ is a triangular matrix with a positive diagonal, steps A and B can be carried out by backward or forward substitution, which requires only an order of n^2 arithmetic operations. The computation of the Cholesky factorization requires an order of n^3 operations.

The computation of the Cholesky factor $\mathbf{L}$ is done via a simple recursive formula. Consider the following block matrix partition of the matrices $\mathbf{A}$ and $\mathbf{L}$:

$$\mathbf{A} = \begin{pmatrix} A_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{12}^T & \mathbf{A}_{22} \end{pmatrix}, \qquad \mathbf{L} = \begin{pmatrix} L_{11} & 0 \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{pmatrix},$$

where $A_{11} \in \mathbb{R}, \mathbf{A}_{12} \in \mathbb{R}^{1\times(n-1)}, \mathbf{A}_{22} \in \mathbb{R}^{(n-1)\times(n-1)}, L_{11} \in \mathbb{R}, \mathbf{L}_{21} \in \mathbb{R}^{n-1}, \mathbf{L}_{22} \in \mathbb{R}^{(n-1)\times(n-1)}$. Since $\mathbf{A} = \mathbf{L}\mathbf{L}^T$ we have

$$\begin{pmatrix} A_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{12}^T & \mathbf{A}_{22} \end{pmatrix} = \begin{pmatrix} L_{11}^2 & L_{11}\mathbf{L}_{21}^T \\ L_{11}\mathbf{L}_{21} & \mathbf{L}_{21}\mathbf{L}_{21}^T + \mathbf{L}_{22}\mathbf{L}_{22}^T \end{pmatrix}.$$

Therefore, in particular,

$$L_{11} = \sqrt{A_{11}}, \qquad \mathbf{L}_{21} = \frac{1}{\sqrt{A_{11}}}\mathbf{A}_{12}^T,$$

and we can thus also write

$$\mathbf{L}_{22}\mathbf{L}_{22}^T = \mathbf{A}_{22} - \mathbf{L}_{21}\mathbf{L}_{21}^T = \mathbf{A}_{22} - \frac{1}{A_{11}}\mathbf{A}_{12}^T\mathbf{A}_{12}.$$

We are left with the task of finding a Cholesky factorization of the $(n-1)\times(n-1)$ matrix $\mathbf{A}_{22} - \frac{1}{A_{11}}\mathbf{A}_{12}^T\mathbf{A}_{12}$. Continuing in this way, we can compute the complete Cholesky factorization of the matrix. The process is illustrated in the following simple example.

Example 5.6. Let

$$\mathbf{A} = \begin{pmatrix} 9 & 3 & 3 \\ 3 & 17 & 21 \\ 3 & 21 & 107 \end{pmatrix}.$$

We will denote the Cholesky factor by

$$\mathbf{L} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix}.$$

Then

$$l_{11} = \sqrt{9} = 3$$

and

$$\begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} = \frac{1}{\sqrt{9}} \begin{pmatrix} 3 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

We now need to find the Cholesky factorization of

$$\mathbf{L}_{22}\mathbf{L}_{22}^T = \mathbf{A}_{22} - \frac{1}{A_{11}}\mathbf{A}_{12}^T\mathbf{A}_{12} = \begin{pmatrix} 17 & 21 \\ 21 & 107 \end{pmatrix} - \frac{1}{9}\begin{pmatrix} 3 \\ 3 \end{pmatrix}\begin{pmatrix} 3 & 3 \end{pmatrix} = \begin{pmatrix} 16 & 20 \\ 20 & 106 \end{pmatrix}.$$

To do so, let us write

$$\mathbf{L}_{22} = \begin{pmatrix} l_{22} & 0 \\ l_{32} & l_{33} \end{pmatrix}.$$

Consequently, $l_{22} = \sqrt{16} = 4$ and $l_{32} = \frac{1}{\sqrt{16}} \cdot 20 = 5$. We are thus left with the task of finding the Cholesky factorization of

$$106 - \frac{1}{16} \cdot (20 \cdot 20) = 81,$$

which is of course $l_{33} = \sqrt{81} = 9$. To conclude, the Cholesky factor is given by

$$\mathbf{L} = \begin{pmatrix} 3 & 0 & 0 \\ 1 & 4 & 0 \\ 1 & 5 & 9 \end{pmatrix}. \qquad \blacksquare$$

The process of computing the Cholesky factorization is well-defined as long as all the diagonal elements l_{ii} that are computed during the process are positive, so that computing their square root is possible. The positiveness of these elements is equivalent to the property that the matrix to be factored is positive definite. Therefore, the Cholesky factorization process can be viewed as a criteria for positive definiteness, and it is actually the test that is used in many algorithms.

Example 5.7. Let us check whether the matrix

$$\mathbf{A} = \begin{pmatrix} 2 & 4 & 7 \\ 4 & 6 & 7 \\ 7 & 7 & 4 \end{pmatrix}$$

is positive definite. We will invoke the Cholesky factorization process. We have

$$l_{11} = \sqrt{2}, \qquad \begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 4 \\ 7 \end{pmatrix}.$$

Now we need to find the Cholesky factorization of

$$\begin{pmatrix} 6 & 7 \\ 7 & 4 \end{pmatrix} - \frac{1}{2}\begin{pmatrix} 4 \\ 7 \end{pmatrix}\begin{pmatrix} 4 & 7 \end{pmatrix} = \begin{pmatrix} -2 & -7 \\ -7 & -20.5 \end{pmatrix}.$$

At this point the process fails since we are required to find the square root of -2, which is of course not a real number. The conclusion is that $\mathbf{A}$ is not positive definite. $\blacksquare$

In MATLAB, the Cholesky factorization if performed via the function `chol`. Thus, for example, the factorization of the matrix from Example 5.6 can be done by the following MATLAB commands:

```
>> A=[9,3,3;3,17,21;3,21,107];
>> L=chol(A,'lower')
L =
     3     0     0
     1     4     0
     1     5     9
```

The function `chol` can also output a second argument which is zero if the matrix is positive definite, or positive when the matrix is not positive definite, and in the latter case a Cholesky factorization cannot be computed. For the nonpositive definite matrix of Example 5.7 we obtain

```
>> A=[2,4,7;4,6,7;7,7,4];
>> [L,p]=chol(A,'lower');
>> p
p =
     2
```

As was already mentioned several times, Newton's method (pure or not) assumes that the Hessian matrix is positive definite and we are thus left with the question of how to employ Newton's method when the Hessian is not always positive definite. There are several ways to deal with this situation, but perhaps the simplest one is to construct a hybrid method that employs either a Newton step at iterations in which the Hessian is positive definite or a gradient step when the Hessian is not positive definite. The algorithm is written in detail below and also incorporates a backtracking procedure.

Hybrid Gradient-Newton Method

Input: $\alpha, \beta \in (0,1)$ - parameters for the backtracking procedure.
$\varepsilon > 0$ - tolerance parameter.

Initialization: Pick $\mathbf{x}_0 \in \mathbb{R}^n$ arbitrarily.
General step: For any $k = 0, 1, 2, \ldots$ execute the following steps:

(a) If $\nabla^2 f(\mathbf{x}_k) \succ 0$, then take $\mathbf{d}_k$ as the Newton direction $\mathbf{d}_k$, which is the solution to the linear system $\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$. Otherwise, set $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$.

(b) Set $t_k = 1$. While

$$f(\mathbf{x}_k) - f(\mathbf{x}_k + t_k \mathbf{d}_k) < -\alpha t_k \nabla f(\mathbf{x}_k)^T \mathbf{d}_k.$$

set $t_k := \beta t_k$

(c) $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \mathbf{d}_k$.

(c) If $\|\nabla f(\mathbf{x}_{k+1})\| \le \varepsilon$, then STOP, and $\mathbf{x}_{k+1}$ is the output.

Following is a MATLAB implementation of the method that also incorporates the Cholesky factorization.

```
function x=newton_hybrid(f,g,h,x0,alpha,beta,epsilon)
% Hybrid Newton's method
%
% INPUT
%=======================================
% f ......... objective function
% g ......... gradient of the objective function
% h ......... hessian of the objective function
% x0......... initial point
% alpha ..... tolerance parameter for the stepsize selection strategy
% beta ...... the proportion in which the stepsize is multiplied
%             at each backtracking step (0<beta<1)
% epsilon ... tolerance parameter for stopping rule
% OUTPUT
%=======================================
% x ......... optimal solution (up to a tolerance)
%             of min f(x)
% fun_val ... optimal function value

x=x0;
gval=g(x);
hval=h(x);
[L,p]=chol(hval,'lower');
if (p==0)
    d=L'\(L\gval);
else
    d=gval;
end
iter=0;
while ((norm(gval)>epsilon)&&(iter<10000))
    iter=iter+1;
    t=1;
    while(f(x-t*d)>f(x)-alpha*t*gval'*d)
        t=beta*t;
    end
    x=x-t*d;
    fprintf('iter= %2d f(x)=%10.10f\n',iter,f(x))
    gval=g(x);
    hval=h(x);
    [L,p]=chol(hval,'lower');
    if (p==0)
        d=L'\(L\gval);
    else
        d=gval;
    end
end

if (iter==10000)
    fprintf('did not converge\n')
end
```

Example 5.8 (Rosenbrock function). Recall that the Rosenbrock function introduced in Example 4.13 is given by $f(x_1,x_2) = 100(x_2 - x_1^2)^2 + (1-x_1)^2$ and is severely ill-conditioned near the minimizer $(1,1)$ (which is the unique stationary point). In Example 4.13 we employed the gradient method with a backtracking stepsize selection strategy, and it took approximately 6900 iterations to converge from the starting point $(2,5)^T$ to

a point satisfying $\|\nabla f(\mathbf{x})\| \le 10^{-5}$. Employing hybrid Newton's method with the same starting point and stopping criteria results in a fast convergence after only 18 iterations:

```
>> f=@(x)100*(x(2)-x(1)^2)^2+(1-x(1))^2;
>> g=@(x)[-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));200*(x(2)-x(1)^2)];
>> h=@(x)[-400*x(2)+1200*x(1)^2+2,-400*x(1);-400*x(1),200];
>> x=newton_hybrid(f,g,h,[2;5],0.5,0.5,1e-5);
iter=  1 f(x)=3.2210220151
iter=  2 f(x)=1.4965858368
    :                :
iter= 16 f(x)=0.0000000000
iter= 17 f(x)=0.0000000000
```

The contour plots of the Rosenbrock function along with the 17 iterates are illustrated in Figure 5.1. ∎

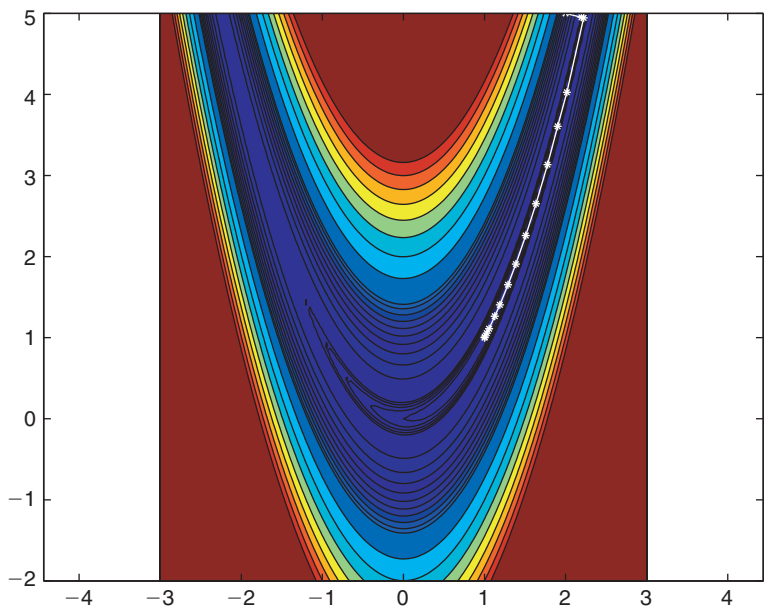

Figure 5.1. *Contour lines of the Rosenbrock function along with the 17 iterates of the hybrid Newton's method.*

Exercises

5.1. Find without MATLAB the Cholesky factorization of the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 4 & 7 \\ 2 & 13 & 23 & 38 \\ 4 & 23 & 77 & 122 \\ 7 & 38 & 122 & 294 \end{pmatrix}.$$

5.2. Consider the Freudenstein and Roth test function

$$f(\mathbf{x}) = f_1(\mathbf{x})^2 + f_2(\mathbf{x})^2, \qquad \mathbf{x} \in \mathbb{R}^2,$$

where

$$f_1(\mathbf{x}) = -13 + x_1 + ((5 - x_2)x_2 - 2)x_2,$$
$$f_2(\mathbf{x}) = -29 + x_1 + ((x_2 + 1)x_2 - 14)x_2.$$

(i) Show that the function f has three stationary points. Find them and prove that one is a global minimizer, one is a strict local minimum and the third is a saddle point.

(ii) Use MATLAB to employ the following three methods on the problem of minimizing f:

1. the gradient method with backtracking and parameters $(s, \alpha, \beta) = (1, 0.5, 0.5)$.
2. the hybrid Gradient-Newton Method with parameters $(s, \alpha, \beta) = (0.5, 0.5)$.
3. damped Gauss–Newton's method with a backtracking line search strategy with parameters $(s, \alpha, \beta) = (1, 0.5, 0.5)$.

All the algorithms should use the stopping criteria $\|\nabla f(\mathbf{x})\| \leq 10^{-5}$. Each algorithm should be employed four times on the following four starting points: $(-50, 7)^T, (20, 7)^T, (20, -18)^T, (5, -10)^T$. For each of the four starting points, compare the number of iterations and the point to which each method converged. If a method did not converge, explain why.

5.3. Let f be a twice continuously differentiable function satisfying $L\mathbf{I} \succeq \nabla^2 f(\mathbf{x}) \succeq m\mathbf{I}$ for some $L > m > 0$ and let $\mathbf{x}^*$ be the unique minimizer of f over $\mathbb{R}^n$.

(i) Show that

$$f(\mathbf{x}) - f(\mathbf{x}^*) \geq \frac{m}{2}\|\mathbf{x} - \mathbf{x}^*\|^2$$

for any $\mathbf{x} \in \mathbb{R}^n$.

(ii) Let $\{\mathbf{x}_k\}_{k \geq 0}$ be the sequence generated by damped Newton's method with constant stepsize $t_k = \frac{m}{L}$. Show that

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \frac{m}{2L}\nabla f(\mathbf{x}_k)^T (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k).$$

(iii) Show that $\mathbf{x}_k \to \mathbf{x}^*$ as $k \to \infty$.