

# Chapter 4

## Global convergence to a local minimum

We are considering the unconstrained minimization problem

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

and have seen a variety of methods for determining a descent direction: At iterate  $\mathbf{x}_k$  the descent direction is denoted  $\mathbf{p}_k$ .

If we use a Newton or a quasi-Newton (e.g. BFGS) method then the update

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$$

yields fast *local* convergence – quadratic or superlinear, respectively – provided  $\mathbf{x}_0$  is “close enough” to the local minimum solution  $\mathbf{x}^*$ .

But to obtain *global convergence*, which means essentially dropping the “close enough” clause, we must modify the basic update: We require a sufficient decrease in  $f(\mathbf{x}_{k+1})$  as compared to  $f(\mathbf{x}_k)$ . If this is not achieved by  $\mathbf{x}_k + \mathbf{p}_k$  then either set  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ , where  $\alpha_k$  is found by line searching, or modify the direction  $\mathbf{p}_k$  altogether using a trust region approach.

### 4.1 Line search methods

The material of this section and more is covered in Chapter 3 of [23].

In these methods we hold the search direction

$$\mathbf{p}_k = -B_k^{-1} \nabla f_k$$

fixed and search for a *step length*  $\alpha_k$  to define the next iterate

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k. \tag{4.1}$$

Recall that if  $B_k$  is positive definite then  $\mathbf{p}_k^T \nabla f_k > 0$  (unless  $\nabla f_k = 0$ , when we're done, which we assume we're not), hence by Taylor's series, for  $\alpha > 0$  small enough

$$f(\mathbf{x}_k + \alpha \mathbf{p}_k) = f_k + \alpha \mathbf{p}_k^T \nabla f_k + \mathcal{O}(\alpha^2) < f_k.$$

We do not perform an exact line search, i.e., we do not solve the one-dimensional minimization problem  $\min_{\alpha} f(\mathbf{x}_k + \alpha \mathbf{p}_k)$ . Rather, we perform a *weak line search*, accepting as  $\alpha_k$  the first  $\alpha$  we find which provides sufficient decrease in the objective function  $f$ .

### What is a “sufficient decrease”?

Note that it's not enough to require  $f(\mathbf{x}_{k+1}) < f_k$ , because we may still obtain convergence to the wrong point: it may happen that  $f(\mathbf{x}_k) \rightarrow f(\hat{\mathbf{x}})$ ,  $f(\hat{\mathbf{x}}) > f(\mathbf{x}^*)$ , or that  $\mathbf{x}_k \rightarrow \hat{\mathbf{x}}$ ,  $f(\hat{\mathbf{x}}) > f(\mathbf{x}^*)$ . (See p. 37 in [23].)

To prevent the first of these pathologies from occurring, require that there be a constant  $\sigma$ ,  $0 < \sigma < 1$ , s.t.

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq f_k + \sigma \alpha_k \nabla f_k^T \mathbf{p}_k. \quad (4.2)$$

This requirement can also be written as

$$\frac{f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) - f_k}{-\alpha_k \nabla f_k^T \mathbf{p}_k} \leq -\sigma,$$

i.e.,  $-\sigma$  is a minimal acceptable relative decrease in the objective function.

In practice, general implementations typically choose  $\sigma = 10^{-4}$ . More aggressive (and less patient) choices are  $\sigma = 10^{-2}$ , or even  $\sigma = 10^{-1}$ . The larger  $\sigma$  the more frequent is the event of software failure, but also the quicker it takes to detect failure.

To prevent the second pathology from occurring we require a minimum step length: that there be a constant  $\rho$ ,  $\sigma < \rho < 1$ , such that

$$\nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)^T \mathbf{p}_k \geq \rho \nabla f_k^T \mathbf{p}_k. \quad (4.3)$$

The condition (4.3) is a *curvature condition*: Let

$$\phi(\alpha) = f(\mathbf{x}_k + \alpha \mathbf{p}_k). \quad (4.4)$$

Then

$$\phi'(\alpha) = \frac{df(\mathbf{x}_k + \alpha \mathbf{p}_k)}{d\alpha} = \nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^T \mathbf{p}_k.$$

So, (4.3) can be written as

$$-\phi'(\alpha) \leq -\rho \phi'(0).$$

In practice, we would choose, e.g.,  $\rho = 0.9$  for a Newton or quasi-Newton direction, or ignore this condition altogether by using backtracking as detailed later on.

The conditions (4.2)-(4.3) are called the *Wolfe conditions*. It can be shown that there always exists a step size satisfying these conditions for objective functions  $f \in C^1$  which are bounded below. For quasi-Newton methods these conditions are particularly important because they guarantee that the next iterate  $\mathbf{x}_{k+1}$  will also satisfy the curvature condition (3.8).

## Global convergence

Let us define an algorithm to be *globally convergent* if for any starting point  $\mathbf{x}_0$  in an appropriate (not necessarily small) set  $\mathcal{N}$

$$\nabla f_k \rightarrow \mathbf{0} \quad \text{as } k \rightarrow \infty.$$

Note that guaranteeing a (local) minimum is harder.

To guarantee global convergence not only the step length  $\alpha_k$  but also the direction  $\mathbf{p}_k$  has to be sufficiently good: the angle  $\theta_k$  defined by

$$\cos \theta_k = \frac{-\nabla f_k^T \mathbf{p}_k}{\|\nabla f_k\| \|\mathbf{p}_k\|}$$

has to be bounded away from  $\pi/2$  (i.e.,  $\mathbf{p}_k$  should be bounded away from orthogonality with respect to the steepest descent direction).

It can be shown (Zoutendijk; see Section 3.2 of [23]) that under reasonable smoothness and boundedness conditions, if the Wolfe conditions hold then

$$\cos^2 \theta_k \|\nabla f_k\|^2 \rightarrow 0.$$

So, if the method used to define the descent direction satisfies that there is a  $\delta > 0$  s.t.

$$\cos \theta_k \geq \delta, \quad \forall k$$

(this is occasionally called the Zoutendijk condition) then global convergence is obtained.

This automatically yields global convergence for the steepest descent algorithm. For Newton-like methods we must have that the matrices  $B_k$  are not only positive definite but also have uniformly bounded condition numbers,

$$\text{cond}(B_k) \leq M, \quad \forall k.$$

This normally happens for quasi-Newton methods. For Newton's method we may need to consider

$$B_k = \nabla^2 f_k + \hat{\delta} I$$

for some appropriate  $\hat{\delta} > 0$  to ensure positive definiteness which is bounded away from singularity.

From a practical point of view the theory just summarized appears to be a bit technical and not altogether impressive for what it actually achieves. The strategies for line search devised below are simple on one hand, while on the other hand if there are two local minima, each with their basin of attraction, then the sets  $\mathcal{N}$  for each are mutually exclusive and not explicitly mapped out by this theory.

## Line search strategies

Usual line search strategies assume that we know  $\hat{\alpha}$  such that the search can be restricted to  $0 < \alpha \leq \hat{\alpha}$ . For Newton and quasi-Newton set  $\hat{\alpha} = 1$ . (This is important for an eventual fast local convergence.)

Thus, we try  $\alpha_k = \hat{\alpha}$ . If (4.2) does not hold then decrease  $\alpha_k$  by the formula  $\alpha_k := \mu\alpha_k$  for some  $0 < \mu < 1$ , and repeat. This is called *backtracking*.

Because we are backtracking, if we do this carefully then we need not worry about the second Wolfe condition (4.3).

A simple backtracking algorithm is Armijo's: set

$$\mu = 1/2.$$

(It's amazing what some people get their names cited for.) This strategy is often sufficient in practice, unless  $\mathbf{p}_k$  is just too rotten a direction.

A more involved algorithm for  $\mu$  is derived as follows. Note that we have  $\phi(0) = f_k$ ,  $\phi'(0) = \nabla f_k^T \mathbf{p}_k$ , and  $\phi(\alpha_k) = f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$  for the current, rejected value of  $\alpha_k$ . The reason that the current  $\alpha_k$  is rejected is that (4.2) does not hold,

$$\phi(\alpha_k) > \phi(0) + \sigma \alpha_k \phi'(0).$$

Let us then model  $\phi(\alpha)$  by a quadratic  $\psi(\alpha)$  which interpolates  $\phi$  at the three known values:

$$\psi(\alpha) = \phi(0) + \phi'(0)\alpha + [\phi(\alpha_k) - \phi(0) - \alpha_k \phi'(0)](\alpha/\alpha_k)^2.$$

We take the new  $\alpha_k$  as the argument which yields the minimum of  $\psi$ , which is where  $\psi'(\alpha) = \phi'(0) + [\phi(\alpha_k) - \phi(0) - \alpha_k \phi'(0)](2\alpha/\alpha_k) = 0$ . This gives the update formula

$$\alpha_k := \frac{-\phi'(0)\alpha_k^2/2}{\phi(\alpha_k) - \phi(0) - \alpha_k \phi'(0)},$$

or,

$$\mu = \frac{-(\nabla f_k^T \mathbf{p}_k)\alpha_k}{2[f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) - f_k - \alpha_k \nabla f_k^T \mathbf{p}_k]}. \quad (4.5)$$

Please verify that for this choice,

$$\mu \leq \frac{1}{2(1-\sigma)} < \frac{1}{2}.$$

If the condition (4.2) does not hold with this new  $\alpha_k$  then a cubic interpolation can be performed involving the two most recent, rejected values of  $\alpha_k$  in order to determine a new one. See [23] for further details on this.

The thorny question in these backtracking algorithms is when and how to admit defeat: after all, the assumptions that guarantee “global” convergence do not always hold; and if we are stuck with a relatively poor direction  $\mathbf{p}_k$ , how long should we take to admit this and do something else instead?

Precise answers to such soul-searching questions depend, as usual, on the application and the type of alternatives which are available.

Below is a simple MATLAB line search function implementing the ideas outlined above (with the exception of the cubic interpolation). Here  $\mathbf{f}$  is the objective function (defined in another MATLAB M-file) and  $\mathbf{f}_x$  and  $\mathbf{g}_x$  stand for  $f_k$  and  $\nabla f_k$ , respectively.

```
function [xn,alpha] = linesearch (f, x, p, fx, gx, sigma, alphamin)
%
% function [xn,alpha] = linesearch (f, x, p, fx, gx, sigma, alphamin)
%
% line search: given current iterate x and direction p,
% find alpha in [alphamin,1] such that xn = x + alpha*p
% gives sufficient descent in f, as specified by sigma.
% Upon return, if alpha <= alphamin then failure

    pgx = p' * gx;
    alpha = 1;
    xn = x + alpha * p;
    fxn = feval(f,xn);
    while (fxn > fx + sigma * alpha * pgx) & (alpha > alphamin)
        mu = -0.5 * pgx * alpha / (fxn - fx - alpha * pgx);
        if mu < .1
            mu = .5; % don't trust quadratic interpolation from far away
        end
        alpha = mu * alpha;
        xn = x + alpha * p;
        fxn = feval(f,xn);
    end
end
```

## 4.2 Trust region methods

The material in this section and more is covered in Chapter 4 of [23].

These methods are more elegant, and occasionally more powerful than line search methods. They are also less straightforward and occasionally more expensive.

Recall that for both Newton and quasi-Newton methods we started by looking at a quadratic model at iterate  $\mathbf{x}_k$ :

$$m_k(\mathbf{p}) = f_k + \nabla f_k^T \mathbf{p} + \frac{1}{2} \mathbf{p}^T B_k \mathbf{p},$$

where  $B_k$  is symmetric positive definite.<sup>1</sup> Then a minimization of  $m_k(\mathbf{p})$  gave

$$\mathbf{p} = \mathbf{p}_k = -B_k^{-1} \nabla f_k,$$

hence

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k.$$

Line search methods as in the previous subsection were introduced *because* this quadratic model did not yield descent, so we did not trust it. Our response was to reduce  $\|\alpha_k \mathbf{p}_k\|$ . From this point of view, then, there is something inherently unsatisfying in line search methods: If we do not trust the model which yields  $\mathbf{p}_k$  then why keep this direction and only play with its step length?!

In trust region methods we directly define or try to control the region in which we trust the quadratic model. So, we do not have a step size  $\alpha_k$ , but rather, we choose the entire search direction  $\mathbf{p}_k$  by controlling its maximum size.

Hence we consider the approximate solution of the special constrained problem

$$\begin{aligned} \min_{\mathbf{p}} \quad & m_k(\mathbf{p}) = f_k + \nabla f_k^T \mathbf{p} + \frac{1}{2} \mathbf{p}^T B_k \mathbf{p} \\ \text{s.t.} \quad & \|\mathbf{p}\| \leq \Delta \end{aligned} \tag{4.6}$$

where  $\Delta = \Delta_k$  represents the bound on (or, radius of) the region in which we trust the quadratic model.

---

<sup>1</sup>In case of Newton we may verify that  $H_k$  is positive definite by estimating its most negative eigenvalue. If  $\lambda_n < 0$  then set

$$B_k \leftarrow H_k - \lambda_n I + \delta I.$$

### Solving (4.6)

Assume for now that we know  $\Delta = \Delta_k$ . The problem (4.6) is a constrained optimization problem in a special form, having a quadratic objective function and only one constraint that can be written as a quadratic  $\mathbf{p}^T \mathbf{p} - \Delta^2 \leq 0$ . Theory, explored more fully in a later chapter, yields that

1. either

$$\mathbf{p}^B = -B_k^{-1} \nabla f_k$$

satisfies  $\|\mathbf{p}^B\| \leq \Delta$ , in which case  $\mathbf{p}_k = \mathbf{p}^B$  is the solution of (4.6) (i.e., the constraint is not binding), or

2.  $\|\mathbf{p}^B\| > \Delta$ , and the solution of (4.6) satisfies

$$\begin{aligned} \mathbf{p}_k &= -(B_k + \lambda I)^{-1} \nabla f_k, \\ \|\mathbf{p}_k\| &= \Delta, \end{aligned} \quad (4.7)$$

i.e., there is an additional unknown  $\lambda$  (the Lagrange multiplier) such that (4.7) is satisfied.

We can imagine a process in which we start with  $\lambda = 0$  and  $\|\mathbf{p}_k = \mathbf{p}^B\| > \Delta$ . We then increase  $\lambda$  gradually until  $\|\mathbf{p}_k\| = \Delta$ . Thus, we increase the “steepest descent component” in the mix that makes up  $\mathbf{p}_k$ .

Indeed, if  $\Delta$  is very small then  $m_k(\mathbf{p})$  can be approximated by  $f_k + \nabla f_k^T \mathbf{p}$ , for which the constrained minimizer is

$$\mathbf{p} = -\frac{\Delta}{\|\nabla f_k\|} \nabla f_k,$$

i.e., a steepest descent direction with a known step length. In fact, the optimal step length for the steepest descent direction is known for  $m_k$ :

$$\mathbf{p}^S = -\alpha_S \nabla f_k, \quad \alpha_S = \frac{\|\nabla f_k\|^2}{\nabla f_k^T B_k \nabla f_k}.$$

More generally, then, we have a mix of  $\mathbf{p}^B$  and  $\mathbf{p}^S$ .

### Dogleg method

To solve the nonlinear equations (4.7) just in order to find a direction  $\mathbf{p}_k$  for the current value of  $\Delta$  is not realistic. Instead, the *dogleg method* composes the direction  $\mathbf{p}_k$  directly from the two directions  $\mathbf{p}^B$  and  $\mathbf{p}^S$ .

1. If  $\Delta$  is so large that  $\|\mathbf{p}^B\| \leq \Delta$  then set  $\mathbf{p}_k = \mathbf{p}^B$ .

2. Otherwise, if  $\Delta$  is so small that  $\|\mathbf{p}^S\| \geq \Delta$  then take

$$\mathbf{p}_k = \frac{\Delta}{\|\mathbf{p}^S\|} \mathbf{p}^S. \quad (4.8a)$$

3. Otherwise, consider

$$\mathbf{p} = \mathbf{p}^S + \tau(\mathbf{p}^B - \mathbf{p}^S), \quad \|\mathbf{p}\|^2 = \Delta^2.$$

Expanding,

$$\Delta^2 = \|\mathbf{p}^S\|^2 + 2\tau(\mathbf{p}^S)^T(\mathbf{p}^B - \mathbf{p}^S) + \tau^2\|\mathbf{p}^B - \mathbf{p}^S\|^2.$$

This is a quadratic equation for  $\tau$  yielding

$$\tau_k = \frac{-(\mathbf{p}^S)^T(\mathbf{p}^B - \mathbf{p}^S) + \sqrt{[(\mathbf{p}^S)^T(\mathbf{p}^B - \mathbf{p}^S)]^2 + (\Delta^2 - \|\mathbf{p}^S\|^2)\|\mathbf{p}^B - \mathbf{p}^S\|^2}}{\|\mathbf{p}^B - \mathbf{p}^S\|^2} \quad (4.8b)$$

(the other root for  $\tau$  is negative), and

$$\mathbf{p}_k = \tau_k \mathbf{p}^B + (1 - \tau_k) \mathbf{p}^S. \quad (4.8c)$$

## A trust region algorithm

All that is left to be ready for coding is to answer the question, how do we determine  $\Delta = \Delta_k$ ?

Consider the ratio of the actual reduction in the objective function over the predicted reduction,

$$\rho_k = \frac{f_k - f(\mathbf{x}_k + \mathbf{p}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{p}_k)}.$$

The denominator is always nonnegative (recall  $B_k$  is positive definite). If  $\rho_k < 0$ , we reject the step: indeed, we demand that  $\rho_k$  be above some positive value. So, we decrease  $\Delta$ . On the other hand, if  $\rho_k$  is close to 1 then our trust in the model increases, so we increase  $\Delta$ .

Here is a MATLAB function that does it all. In addition to the notation of the trust region routine we have here `hx` standing for  $B_k$ .

```
function [x,p,Del] = trustreg (f, x, p, fx, gx, hx, eta, Del, Delb)
%
% function [x,p,Del] = trustreg (f, x, p, fx, gx, hx, eta, Del, Delb)
%
```

```

% trust region: given current iterate x and 'Newton-like' direction p,
% find a mix of p and steepest descent, renamed p,
% such that norm(p) is bounded by Del.
% The value of Del is updated as well.
% Upon return, if Del < alphamin * (1+norm(x)) then failure

% obtain direction p

gxp = gx' * p;
% if input p satisfies constraint, we're happy
if norm(p) > Del

    % optimal steepest descent:
    alpha = norm(gx)^2 / (gx'*hx*gx);
    ps = -alpha * gx;
    nps = norm(ps);

    % dogleg direction
    if nps >= Del
        p = (Del / nps) * ps;
    else
        p1 = ps' * (p - ps);
        p2 = norm(p-ps)^2;
        tau = (-p1 + sqrt (p1^2 + (Del^2-nps^2)*p2 )) / p2;
        p = tau * p + (1-tau) * ps;
    end

end

% evaluate improvement rho
xn = x + p;
fn = feval (f,xn);
rho = (fn - fx) / (gx'*p + 0.5*p'*hx*p);

np = norm (p);
if rho < 0.25
    Del = 0.25 * np; %shrink region where we trust model
elseif (rho > 0.75) * (np == Del)
    Del = min (2*Del, Delb); % increase region where we trust model
end

if rho > eta

```

```

    x = xn;
end

```

### 4.3 Numerical examples

Below is my MATLAB implementation of a modified Newton's method which calls the line search or the trust region procedures. Thus, at each iteration  $k$  the Hessian is evaluated, and a full Newton step is considered first. If no sufficient decrease in the objective function is obtained, or the iterate is already close to a critical point, then the Hessian's smallest eigenvalue is calculated: If it's not sufficiently above 0 then an appropriate multiple of the identity is added to form a positive definite  $B_k$ . A line search or a trust region step is then launched.

```

function [x,k,fvals] = newtgl (f,x,tol,nmax,gl)
%
% function [x,k,fvals] = newtgl (f,x,tol,nmax,gl)
%
% This function returns in x a column vector x_k such that
%     || x_k - x_{k-1} || < tol (1 + ||x_k||)
% and in k the number of iterations required (+1).
% On entry, x contains an initial guess.
% If k exceeds nmax then no convergence has been reached.
% Also, the values of f at subsequent iterations are
% in fvals
%
% The method used is Newton with either
%     a (quadratic-based) line search for gl = 1   or
%     a trust region (dogleg) strategy for gl = 2.
%
% constants
n = size(x,1);
n2 = sqrt(n);
delta = 1.e-4; % minimum descent
sigma = 1.e-4; % minimum decrease in f
alphamin = 1.e-4; % minimum step length
eta = 1.e-4; % minimum reduction for trust region step
Delb = 10*(n2 + norm(x)); % maximum trust region radius

% ensure x is column vector
if size(x,1) < size(x,2)

```

```

    x = x';
end
n = size(x,1);
I = eye(n);
k = 0;
Del = Delb;

% Iterate newton
while k < nmax
    [fx,gx,hx] = feval(f,x);
    k = k+1;
    fvals(k) = fx;
    ngx = norm(gx);

    % Give pure Newton a chance first
    p = -hx \ gx;
    np = norm(p);
    naccept = 1;
    if (np < Del) & (np > tol*(n2 + norm(x)))    % try a step
        xn = x + p;
        [fxn,gxn] = feval(f,xn);
        if fxn < fx + sigma * p'* gx
            naccept = 0;
        end
    end
end

if naccept % make Newton's direction positive definite
    ln = min (eig (hx)); % this is good only for small problems
    lnp = max(0, -ln);
    if ln > delta
        mu = 0;
    else
        mu = lnp + delta;
    end
    hx = hx + mu*I;
    p = -hx \ gx;
    np = norm(p);
end

% check for successful termination
if (np < tol*(n2 + norm(x))) & (ngx < tol)
    x = x + p;
end

```

```

    return
end

% find next iterate
if gl == 1
    %line search
    [x,alpha] = linesearch (f, x, p, fx, gx, sigma, alphamin);
    if alpha < .99
        fprintf('after iteration %d, alpha = %e\n',k,alpha) % for debugging
    end
    % check for failure
    if alpha < alphamin
        fprintf(' Procedure failed to locate a minimum; alpha = %e\n',alpha);
        return
    end
else
    %trust region
    [x,p,Del] = trustreg (f, x, p, fx, gx, hx, eta, Del, Delb);
%   fprintf('after iteration %d, Del = %e\n',k,Del) % for debugging
    % check for failure
    if Del < (alphamin * (n2+norm(x)))
        fprintf(' Procedure failed to locate a minimum; Del = %e\n',Del);
        return
    end
end
end
end
end

```

Using this program we next consider the operation of these two global strategies as compared to using the unadulterated Newton's method.

#### Example 4.1

Let  $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T H \mathbf{x}$ , where  $\mathbf{c} = (5.04, -59.4, 146.4, -96.6)^T$ ,

$$H = \begin{pmatrix} .16 & -1.2 & 2.4 & -1.4 \\ -1.2 & 12.0 & -27.0 & 16.8 \\ 2.4 & -27.0 & 64.8 & -42.0 \\ -1.4 & 16.8 & -42.0 & 28.0 \end{pmatrix}.$$

Starting the pure Newton method from  $\mathbf{x}_0 = (-1, 3, 3, 0)^T$ , or from any other point for that matter, yields the minimum  $\mathbf{x}^* = (1, 0, -1, 2)^T$ , at which

$f(\mathbf{x}^*) = -167.28$ , in one iteration. (It takes another iteration, though, to verify that indeed  $\|\mathbf{x}_k - \mathbf{x}_{k-1}\| \leq \text{tol}(1 + \|\mathbf{x}_k\|)$  for  $k = 2$ .) The reason that Newton's method finds a critical point in one iteration is that this function is quadratic. The reason that the critical point is a minimum is that the Hessian  $H$  is positive definite: its eigenvalues are approximately 103.4, 1.484,  $5.912 \times 10^{-2}$  and  $6.666 \times 10^{-3}$ .

The matrix  $H$  is not very well conditioned, though:  $\text{cond}(H) = \frac{\lambda_1}{\lambda_4} \approx 15,513$ . Thus, we expect methods which do not take advantage of precise second order information to take significantly longer to converge. For instance, the BFGS quasi-Newton method (in pure form, i.e. without line search or trust region modifications) converges after 18 iterations for  $\text{tol} = 1.e - 6$ .

Both our global strategies, the line search and the trust region methods, converge here also in one iteration. However, if we insist that the iteration matrix  $B$  have all eigenvalues above  $\delta = 10^{-2}$ , say, then  $H$  does not qualify because the smallest eigenvalue  $\lambda_4$  of  $H$  is below  $\delta$ . The corresponding modification  $B = H + \mu I$  then slows down the process significantly: Convergence for  $\text{tol} = 1.e - 6$  requires 22 iterations.

◆

### Example 4.2

Consider  $f(\mathbf{x}) = [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2$ .

We try two starting points:  $\mathbf{x}_0^{(1)} = (8, .2)^T$ , and  $\mathbf{x}_0^{(2)} = (8, .8)^T$ .

There is a unique minimum:  $\mathbf{x}^* = (3, .5)^T$ ,  $f(\mathbf{x}^*) = 0$ .

There is also a saddle point, though: At  $\hat{\mathbf{x}} = (0, 1)^T$  the gradient vanishes,

but the Hessian equals  $H = \begin{pmatrix} 0 & 27.75 \\ 27.75 & 0 \end{pmatrix}$ . The eigenvalues are  $\pm 27.75$ ,

hence the Hessian is not positive semi-definite and the necessary conditions for a minimum are not satisfied.

Starting from  $\mathbf{x}_0 = (8, .2)^T$  we obtain the iterations recorded in Table 4.1. Rapid convergence to the minimum is observed. Note that the values of  $f$  decrease monotonically here, even though the Hessian at the starting iterate,  $\nabla^2 f(\mathbf{x}_0)$ , is not positive definite: luck is on our side with this one.

Starting from  $\mathbf{x}_0 = (8, .8)^T$  we obtain the iterations recorded in Table 4.2. Rapid convergence is again observed but, unfortunately, to the saddle point  $(1, 0)^T$  rather than to the minimum! Note that the directions encountered are all descent directions: This can happen if the critical point is a saddle point (and not a maximum), although the Hessian matrices encountered are not all positive definite, of course.

| $k$ | $\ \mathbf{x}_k - \mathbf{x}^*\ $ | $f_k - f(\mathbf{x}^*)$ | $-\nabla f_k^T \mathbf{p}_k$ |
|-----|-----------------------------------|-------------------------|------------------------------|
| 0   | 5.008992e+00                      | 8.170162e+01            | -1.454963e+02                |
| 1   | 8.659729e-01                      | 2.423044e+00            | -4.526698e+00                |
| 2   | 6.493645e-02                      | 2.407468e-02            | -4.643199e-02                |
| 3   | 1.392602e-01                      | 3.449554e-03            | -6.319942e-03                |
| 4   | 2.102606e-02                      | 1.382774e-04            | -2.704124e-04                |
| 5   | 1.376857e-03                      | 2.862970e-07            | -5.717405e-07                |
| 6   | 3.033257e-06                      | 2.185891e-12            | -4.371763e-12                |
| 7   | 2.835847e-11                      | 1.232782e-22            | -2.465564e-22                |

Table 4.1: Example 4.2,  $\mathbf{x}_0 = \mathbf{x}_0^{(1)}$ : The pure Newton’s method yields the minimum quickly.

Thus, this numerical example illustrates also a shortcoming of the “global convergence” theory: although the Wolfe and the Zoutendijk conditions hold convergence is obtained only in the sense that  $\nabla f_k \rightarrow 0$ . Convergence to a minimum is not guaranteed, indeed here it does not occur!

Note that for this problem specifying  $\nabla^2 f$  is a bit of a pain. There are general procedures for either computing  $\nabla^2 f$  exactly yet automatically (this is called **automatic differentiation**) or approximating these derivatives by taking finite differences of the gradient. The latter is simple, but it requires quite a few gradient evaluations (between  $n$  and  $2n$ ) and there are scaling problems as well.

It is important to understand that evaluating the gradient only approximately is much more of a “sin” than evaluating the Hessian only approximately.

Let us turn now to the performance of the global strategies. For the first starting point,  $\mathbf{x}_0 = (8, .2)^T$ , our special implementation of the basic search direction provides acceptable pure Newton steps (even when the Hessians are not positive definite), and the resulting sequence of iterates is precisely as in Table 4.1. But for the second starting point,  $\mathbf{x}_0 = (8, .8)^T$ , the program eventually checks the Hessian for sufficient positive definiteness and corrects the Newton direction, thus avoiding convergence to the saddle point. The results are recorded in Table 4.3. The trust region method does not update

| $k$ | $\ \mathbf{x}_k - \mathbf{x}^*\ $ | $f_k - f(\mathbf{x}^*)$ | $-\nabla f_k^T \mathbf{p}_k$ |
|-----|-----------------------------------|-------------------------|------------------------------|
| 0   | 5.008992e+00                      | 2.042741e+00            | -3.291419e+00                |
| 1   | 3.962964e+00                      | 2.552922e-01            | -5.885204e-02                |
| 2   | 4.218276e+00                      | 2.328013e-01            | -6.421302e-01                |
| 3   | 1.661422e+01                      | 5.743155e+02            | -9.290908e+02                |
| 4   | 6.137264e+00                      | 5.225736e+01            | -6.327071e+01                |
| 5   | 3.426011e+00                      | 1.595973e+01            | -3.708800e+00                |
| 6   | 2.973779e+00                      | 1.420736e+01            | -8.444914e-03                |
| 7   | 3.041143e+00                      | 1.420313e+01            | -5.688234e-08                |
| 8   | 3.041381e+00                      | 1.420312e+01            | -1.083034e-18                |

Table 4.2: Example 4.2,  $\mathbf{x}_0 = \mathbf{x}_0^{(2)}$ : The pure Newton's method finds a critical point, but it's not the minimum.

the iterate when  $\rho_k \leq \eta$ , hence the seemingly larger number of total iterations need not require more work than for the line search procedure. ◆

### Example 4.3

Consider the Rosenbrock function  $f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$ .

Starting point:  $\mathbf{x}_0 = (0, 0)^T$

Solution:  $\mathbf{x}^* = (1, 1)^T$ ,  $f(\mathbf{x}^*) = 0$ .

For this problem the pure Newton's method finds the minimum in two iterations; see Table 4.4. This is so even though the value of  $f_1$  jumps up by a factor of 100 before finding the solution in the next iterate. This example is very special indeed – we may not expect such luck in general!

Turning to the performance of the global strategies, we now require a monotonic descent of the objective function. This is sure to necessitate more than two iterations. Indeed, it turns out that using the line search procedure requires 14 iterations, with step sizes 0.25, 0.5, 0.14, 0.28 required at iterates 1, 3, 5 and 6, respectively. The trust region procedure requires 16 iterations, with  $\tau_k = 0.31, 0.37, 0.47, 0.5$  at iterates 3, 6, 7 and 8, respectively. The line search procedure is slightly better here.

**Example 4.4**

Consider  $f(\mathbf{x}) = x_1^4 + x_1x_2 + (1 + x_2)^2$ .

Try two starting points:  $\mathbf{x}_0 = (.75, -1.25)^T$ , and  $\mathbf{x}_0 = (0, 0)^T$ .

Solution:  $\mathbf{x}^* \approx (.695884386, -1.34794219)^T$ ,  $f(\mathbf{x}^*) \approx -.582445174$ .

Starting from  $\mathbf{x}_0 = (.75, -1.25)^T$  and using the pure Newton's method we obtain the iterations recorded in Table 4.5. Rapid convergence to the minimum is observed.

However, starting the pure Newton iteration from  $\mathbf{x}_0 = (0, 0)^T$  yields no convergence.

To understand why this happens, note that the Hessian  $\nabla^2 f(x) = \begin{pmatrix} 12x_1^2 & 1 \\ 1 & 2 \end{pmatrix}$

is positive definite only when  $24x_1^2 > 1$ . In particular, the Hessian is singular for  $x_1 = \pm \frac{1}{\sqrt{24}}$ . The initial iterate  $\mathbf{x}_0 = (0, 0)^T$  and subsequent Newton iterates are trapped in the “bad region” of the Hessian.

Turning to the global strategies, the “good” starting point yields a sequence of pure Newton iterates, as recorded in Table 4.5. Starting from  $\mathbf{x}_0 = (0, 0)^T$  and using the line search strategy we obtain convergence in 5 iterations. The first step length is successively cut down (6 times) until  $\alpha_0 = 1.22e - 04$  is found acceptable. After this first step, pure Newton iterations yield rapid convergence. The trust region method performs similarly.



Trying to summarize the experience gained from the above four examples we see that the pure Newton's method often works well, but there are exceptions. When it does not work well those global strategies may come in handy. But be sure to realize that, although here the global strategies have always worked, this is not true in general. The performance of the line search and the trust region strategies is comparable for these examples, which is a blow to the more involved trust region method.

General purpose software for non-large problems is usually based on a quasi-Newton method, usually BFGS, rather than on the slightly modified Newton method that I have implemented. One reason is simply requiring much less input from the user (by not requiring the Hessian), another is that no eigenvalue estimation is needed. Indeed, in the 1970's everyone seemed to feel that quasi-Newton was the only way to go, while Newton's method itself was deemed too clutzy. But the trend since has somewhat reversed. Not

only are there ways, as we mentioned earlier during Example 4.2, to avoid specifying the Hessian manually, also for large and sparse problems there are ways to modify Newton's method to handle the positive definiteness issue, as we will see in the next chapter.

## Problems in more than two unknowns?

Considering peculiar problems in two unknowns can be addictive...! Yet most practical problems that one encounters look very different. They are typically large and they often have some special structure. This is an essential reason for taking this course – not for solving toy problems which canned routines excel in. It may even be a challenge just to set the problem up, occasionally more challenging even than subsequently solving it. For something quite different, then, let us consider an instance of distributed parameter estimation and concentrate on just setting the problem up. This can be viewed in part as motivation for the next chapter.

### Example 4.5

Recall Example 2.5. Here we consider in detail an instance in 1D. Moreover, assume that the forward problem is given as a boundary value ODE,

$$\begin{aligned} -(\sigma(t)u')' &= q(t), & 0 \leq t < 1 \\ u'(0) &= u(1) = 0. \end{aligned}$$

The function  $\sigma(t)$  to be recovered must be positive (and bounded away from 0) throughout the interval. We will, however, assume below that this constraint is non-binding and seek the unconstrained optimum as in Example 2.5.

The solution of the forward problem can be written as

$$u = \mathcal{G}(\sigma)q$$

where  $\mathcal{G}$  is the Green's function of this differential problem.

Let us next discretize this forward problem on a uniform grid with step size  $h = 1/N$ . We write it as a first order system,

$$\begin{aligned} u' &= \sigma^{-1}j \\ j' &= -q \\ j(0) &= u(1) = 0 \end{aligned}$$

and consider the unknowns  $u_{i-1/2}$  as placed at cell (or subinterval) centers,  $(i - 1/2)h$ ,  $i = 1, 2, \dots, N$ , and the unknowns  $j_i$  as placed at mesh points





Note how forming  $\nabla f$  can be accomplished using sparse matrix operations only.

The Hessian can likewise be written as

$$\begin{aligned}\nabla^2 f(\boldsymbol{\sigma}) &= S^T S + \beta W^T W - T - T^T, \quad \text{where} \\ T &= V A^{-1} U, \quad \text{where} \\ V &= \frac{\partial(A(\boldsymbol{\sigma})^T \boldsymbol{\lambda})}{\partial \boldsymbol{\sigma}}, \quad \text{where} \\ \boldsymbol{\lambda} &= -A(\boldsymbol{\sigma})^{-T} [A^{-1}(\boldsymbol{\sigma}) \mathbf{q} - \mathbf{b}].\end{aligned}$$

The *Gauss-Newton* approximation for the Hessian is obtained by dropping the terms involving  $T$ , yielding a positive definite matrix

$$B(\boldsymbol{\sigma}) = S^T S + \beta W^T W.$$

We'll discuss the resulting method further in the chapter on nonlinear least squares problems.



| $k$ | $\alpha_k$ | $f_k - f(\mathbf{x}^*)$ | $\tau_k$ | $f_k - f(\mathbf{x}^*)$ |
|-----|------------|-------------------------|----------|-------------------------|
| 0   | 1.0        | 2.042741                | 1.0      | 2.042741                |
| 1   | 1.0        | 2.552922e-1             | 1.0      | 2.552922e-1             |
| 2   | 1.70e-2    | 2.328013e-1             | 1.0      | 2.328013e-1             |
| 3   | 0.25       | 2.220907e-1             | 0.25     | 2.328013e-1             |
| 4   | 1.0        | 2.137310e-1             | 6.25e-2  | 2.328013e-1             |
| 5   | 7.92e-2    | 1.479007e-1             | 1.0      | 2.060980e-1             |
| 6   | 0.31       | 1.241418e-1             | 0.89     | 1.609696e-1             |
| 7   | 1.0        | 9.947654e-2             | 0.22     | 1.609696e-1             |
| 8   | 1.0        | 6.679143e-2             | 5.27e-2  | 1.365913e-1             |
| 9   | 1.0        | 2.087717e-2             | 6.86e-1  | 9.984920e-2             |
| 10  | 1.0        | 1.582909e-2             | 1.0      | 6.091570e-2             |
| 11  | 1.0        | 1.025067e-5             | 1.0      | 2.331891e-2             |
| 12  | 1.0        | 1.689930e-9             | 1.0      | 5.977851e-3             |
| 13  | 1.0        | 4.445213e-17            | 1.0      | 2.562167e-4             |
| 14  |            |                         | 1.0      | 1.105912e-6             |
| 15  |            |                         | 1.0      | 2.884519e-13            |
| 16  |            |                         | 1.0      | 2.148511e-24            |

Table 4.3: Example 4.2,  $\mathbf{x}_0 = \mathbf{x}_0^{(2)}$ : The line search procedure (columns 2 & 3) and the trust region procedure (columns 4 & 5) both yield the minimum.

| $k$ | $\ \mathbf{x}_k - \mathbf{x}^*\ $ | $f_k - f(\mathbf{x}^*)$ | $-\nabla f_k^T \mathbf{p}_k$ |
|-----|-----------------------------------|-------------------------|------------------------------|
| 0   | 1.414214e+00                      | 1.000000e+00            | -2.000000e+00                |
| 1   | 1.000000e+00                      | 1.000000e+02            | -2.000000e+02                |
| 2   | 0.000000e+00                      | 0.000000e+00            | 0.000000e+00                 |

Table 4.4: Example 4.3,  $\mathbf{x}_0 = (0, 0)^T$ : The pure Newton's method finds the minimum, even though the sequence of values of  $f(\mathbf{x}_k)$  is far from being monotonically decreasing.

| $k$ | $\ \mathbf{x}_k - \mathbf{x}^*\ $ | $f_k - f(\mathbf{x}^*)$ | $-\nabla f_k^T \mathbf{p}_k$ |
|-----|-----------------------------------|-------------------------|------------------------------|
| 0   | 1.118980e-01                      | 2.385142e-02            | -4.687500e-02                |
| 1   | 4.601398e-03                      | 4.517400e-05            | -8.996283e-05                |
| 2   | 2.951148e-05                      | 1.406451e-09            | -3.700069e-09                |
| 3   | 3.805090e-09                      | -4.436351e-10           | -6.371804e-18                |

Table 4.5: Example 4.4,  $\mathbf{x}_0 = \mathbf{x}_0^{(1)}$ : The pure Newton's method yields the minimum quickly.