Chapter 4

Miscellaneous topics

4.1 Nonlinear least-squares and nonlinear equations

Many optimization problems encountered in applications are nonlinear least-squares problems. Such problems have the form

$$\min_x \frac{1}{2} ||F(x)||^2,$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ and (usually) $m > n$. Typically $F$ has the form $F(x) = m(x) - d$, where $m$ describes a mathematical model depending on the parameters $x$ and $d$ is observed data. The model is known to have the form $m(x)$ for theoretical reasons, but the actual values of the parameters must be determined by performing experiments, measuring the data, and choosing the value of $x$ that gives the best fit to the data. The nonlinear least-squares problem can be regarded as a general unconstrained minimization problem by defining

$$f(x) = \frac{1}{2} ||F(x)||^2$$

and minimizing $f$. However, as I explain below, this problem can (at least under certain circumstances) be solved more efficiently by taking into account the special form of the objective function.

The problem of solving a system of nonlinear equations, represented by $F(x) = 0$, where $F : \mathbb{R}^n \to \mathbb{R}^n$ (that is, $m = n$), is closely related to the nonlinear least-squares problem. I have already explained Newton’s method and Broyden’s method for solving $F(x) = 0$, given a good estimate of the solution. To globalize these methods, one can use the objective function (4.1) and a line search to ensure progress when far from the solution.

4.1.1 Nonlinear least-squares

When $f$ is defined by (4.1), then

$$\nabla f(x) = F'(x)^T F(x)$$
and
\[ \nabla^2 f(x) = F'(x)^T F'(x) + \sum_{i=1}^{m} F_i(x) \nabla^2 F_i(x), \]

where \( F'(x) \) is the Jacobian of \( F \) at \( x \). The key observation is the following: By computing \( F'(x) \), one knows not only the gradient of \( f \) but also part of the Hessian. One can use \( F'(x)^T F'(x) \) as a “free” approximation to \( \nabla^2 f(x) \), and this approximation is useful for two reasons:

1. The matrix \( F'(x)^T F'(x) \) is always at least positive semidefinite, and it is positive definite if \( F'(x) \) has full rank. Therefore, choosing \( H_k = F'(x^{(k)})^T F'(x^{(k)}) \) leads to descent directions in many cases. (If \( F'(x^{(k)}) \) is rank-deficient, so that \( H_k \) is singular, then \( H_k \) can be modified directly to make it positive definite, say by adding a multiple of the identity matrix.)

2. If the problem at hand is a zero-residual problem (\( F(x^*) = 0 \)) or a small-residual problem (\( \|F(x^*)\| \) small), then \( H_k \) is an increasingly good approximation to \( \nabla^2 f(x^{(k)}) \) as \( x^{(k)} \to x^* \). This implies that a quasi-Newton algorithm using \( H_k = F'(x^{(k)})^T F'(x^{(k)}) \) can achieve fast local convergence.

The approximation \( F'(x)^T F'(x) \) to \( \nabla^2 f(x) \) is often referred to as the Gauss-Newton Hessian and the quasi-Newton algorithm based on it as the Gauss-Newton method. The Gauss-Newton step is
\[ x^{(k+1)} = x^{(k)} - \left( F'(x^{(k)})^T F'(x^{(k)}) \right)^{-1} F'(x^{(k)})^T F(x^{(k)}). \]

As the above discussion suggests, the Gauss-Newton method may converge slowly in a large-residual problem, since in such a problem the Gauss-Newton Hessian is a poor approximation to the true Hessian at the solution. A large residual problem can be solved using general unconstrained minimization algorithms.

Since one often does not know in advance if a least-squares problem is large- or small-residual, various algorithms have been proposed that use the Gauss-Newton Hessian, modifying it only if the step it produces fails to yield good progress. The reader can consult Chapter 10 of Nocedal and Wright [10] for details.

### 4.1.2 Nonlinear equations

A system of nonlinear equations represented by \( F(x) = 0, F : \mathbb{R}^n \to \mathbb{R}^n \), can be posed as the nonlinear least-squares problem
\[ \min_x \frac{1}{2} \|F(x)\|^2. \]

Assuming that the system of equations has a solution, the problem is known to be a zero-residual problem, and it makes sense to apply the Gauss-Newton algorithm. However, in this case \( F'(x^{(k)}) \) is square and (hopefully) nonsingular, so the Gauss-Newton step is
\[ x^{(k+1)} = x^{(k)} - \left( F'(x^{(k)})^T F'(x^{(k)}) \right)^{-1} F'(x^{(k)})^T F(x^{(k)}). \]
4.2 Trust region methods

\[ x^{(k)} - F'(x^{(k)})^{-1} F'(x^{(k)})^T F(x^{(k)}) = x^{(k)} - F'(x^{(k)})^{-1} F(x^{(k)}) \]

This calculation shows that the Gauss-Newton step for minimizing \( f(x) = (1/2)||F(x)||^2 \) is the same as the Newton step for solving \( F(x) = 0 \). (On the other hand, the Newton step for minimizing \( f \) is generally different.) Therefore, the analysis presented earlier guarantees fast local convergence, and a line search can be used to ensure global convergence.

There is, however, one problem with the global convergence guarantee: numerical optimization techniques only guarantee global convergence to a local minimizer of the objective function. The function \( f(x) = (1/2)||F(x)||^2 \) can have local, non-global minimizers. Such a “solution” \( x^* \) is useless since \( F(x^*) \neq 0 \). There is really no way around this difficulty in general when using the Gauss-Newton approach. If the algorithm converges to a local, non-global minimizer of \( f \), it must be restarted with a different \( x^{(0)} \).

An alternative approach to solving \( F(x) = 0 \) that is generally more robust is the homotopy method. To apply this method, one must define a smooth function \( H: \mathbb{R}^n \times [0, 1] \to \mathbb{R}^n \) with the properties that a solution \( x_0 \) of \( H(x, 0) = 0 \) is known and

\[ H(x, 1) = F(x) \text{ for all } x \in \mathbb{R}^n. \]

The goal is then to follow a path \( x_t \) with the property that

\[ H(x_t, t) = 0 \text{ for all } t \in [0, 1]. \]

Following the path or trajectory \( x_t \) is similar to solving a system of ordinary differential equations. For details, the interested reader can consult Nocedal and Wright [10], Section 11.3, and the references contained therein.

4.2 Trust region methods

I have presented a single strategy for globalizing Newton’s method and quasi-Newton methods, namely, the use of a line search. The reader will recall that, in the line search approach, the local model

\[ f(x + p) \approx q_k(x) = f(x^{(k)}) + \nabla f(x^{(k)}) \cdot p + \frac{1}{2} p \cdot H_k p \quad (4.2) \]

is used to generate a descent direction \( p^{(k)} \). A line search is then used to choose the step length \( \alpha_k \) and then \( x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \).

Trust region methods use the quadratic model \( q_k \) in an essentially different fashion. Since (4.2) is a local model, it is expected to accurately represent \( f \) only in a region near \( x^{(k)} \). Using heuristics, it is possible to maintain an estimate \( \Delta \) of the radius of a ball on which the quadratic model is an accurate representation of \( f \). The step from \( x^{(k)} \) is then determined by minimizing \( q_k \) over this ball, which is called the trust region (it is the region in which the quadratic model is trusted).
The trust region subproblem is

$$\min f(x^{(k)}) + \nabla f(x^{(k)}) \cdot p + \frac{1}{2} p \cdot H_k p$$

s.t. $\|p\| \leq \Delta$. 

This should be compared with the line search subproblem, which is the one-dimensional minimization

$$\min_{\alpha > 0} f(x^{(k)} + \alpha p^{(k)}).$$

The advantage of the trust region approach is partially due to the fact that $H_k$ is not required to be positive definite (so, in particular, $H_k = \nabla^2 f(x^{(k)})$ can be used even if it is singular or indefinite). If $H_k$ has directions of negative curvature, then the trust region algorithm can take advantage of them directly. 

As a result of this better use of the curvature information, it is possible to prove stronger convergence theorems. In particular, when $H_k = \nabla^2 f(x^{(k)})$, then it is possible (under some assumptions) to prove convergence to a stationary point $x^*$ that also satisfies the second-order necessary condition that $\nabla^2 f(x^*)$ be positive semidefinite.

### 4.2.1 The trust region subproblem

The trust region subproblem is interesting in its own right. It is an inequality-constrained minimization problem, and therefore falls into a class of problems that I will discuss later in the course. However, the trust region subproblem is special in that it is possible to write down necessary and sufficient conditions for $p$ to be a global minimizer. For general inequality-constrained optimization problems, as for general unconstrained optimization problems, only necessary conditions are available for global solutions.\(^\text{14}\)

**Theorem 4.1.** Suppose $f \in \mathbb{R}$, $g \in \mathbb{R}^n$, and $H \in \mathbb{R}^{n \times n}$. The vector $p^*$ is a global solution of

$$\min f + g \cdot p + \frac{1}{2} p \cdot Hp$$

s.t. $\|p\| \leq \Delta$ 

if and only if $\|p^*\| \leq \Delta$ and there is a scalar $\lambda \geq 0$ such that:

1. $(H + \lambda I)p^T = -g$;

2. $\lambda(\Delta - \|p^*\|) = 0$;

3. and $H + \lambda I$ is positive semidefinite.

\(^\text{14}\)As I will discuss later, there is a class of inequality-constrained nonlinear programs (called convex programs) for which the necessary conditions are necessary and sufficient. The trust region subproblem has (almost) the same property even though it is not a convex problem.
4.3 Scaling of functions and variables: an example

The scalar \( \lambda \) is the Lagrange multiplier for the trust region subproblem (I will define Lagrange multipliers later), and it can be shown that \( \lambda \) increase as \( \Delta \) decreases, with \( \lambda \to \infty \) as \( \Delta \to 0^+ \). If \( H \) is nonsingular and the quasi-Newton step \( p^N = -H^{-1}g \) lies inside the trust region, then \( p^N \) is the solution of the trust region subproblem and \( \lambda = 0 \). On the other hand, as \( \Delta \) decreases to zero, \( \lambda \) increases to infinity, which means that \( H + \lambda I \) is increasingly dominated the \( \lambda I \) term. The effect is that \( p^* = -(H + \lambda I)^{-1}g \) lies more and more in the steepest descent direction as \( \Delta \to 0 \). Thus the solution to the trust region subproblem follows a path from the quasi-Newton direction to the steepest descent direction as \( \Delta \) decreases.

I will not discuss trust region methods further. There are a variety of algorithms for solving the trust region subproblem and these have been incorporated into effective optimization codes. The reader is referred to Nocedal and Wright [10], Chapter 4, for more details.

4.3 Scaling of functions and variables: an example

The discussion of stopping test in the previous section pointed out the importance of taking into account the scaling of the variable and function values. In this section, I will show by an explicit example how a poorly scaled problem causes difficulty for an optimization code. The issues raised by this example are typical of those arising in many practical problems.

The application discussed here involves human population growth, specifically the growth in the population of the United States since its founding. A discrete model will be adopted, in which \( P_n \) denotes the population \( n \) years after the first U.S. census (so \( n = 0 \) corresponds to the year 1790).

The simplest model of population growth is based on a constant growth rate:

\[
P_{n+1} = (1+r)P_n, \quad n = 0, 1, 2, \ldots \tag{4.3}
\]

Each year a fraction \( b \) of the population gives birth and a fraction \( d \) dies, with the result that the population changes by \( rP_n \) individuals, where \( r = b - d \). (For simplicity of exposition, I will ignore immigration.) The difference equation (4.3) can be easily solved to yield

\[
P_n = P_0(1+r)^n, \quad n = 0, 1, 2, \ldots
\]

The growth implied by (4.3) is therefore exponential, which is not realistic for human population growth.

A more realistic model is the logistic difference equation,

\[
P_{n+1} = P_n + r \left( 1 - \frac{P_n}{L} \right) P_n, \quad n = 0, 1, 2, \ldots \tag{4.4}
\]

The constant \( L \) is called the carrying capacity, it defines the intrinsic capacity of the environment.
4.4 Stopping tests for optimization and nonlinear equations

Since algorithms for nonlinear optimization are iterative in nature, with computed solutions converging to the true solution only in the limit, stopping criteria form an important part of an implementable algorithm. A good optimization code will stop the iteration when one of several conditions has been met. The following material is adapted from Dennis and Schnabel [5], Chapter 7.

4.4.1 Stopping tests for optimization

The gradient tolerance

In a quasi-Newton (descent) algorithm, it is (implicitly) assumed that approximating a stationary point is equivalent to solving a minimization problem. Therefore, the most important stopping test should be something like

$$||\nabla f(x^{(k)})|| \leq \epsilon_1.$$  (4.5)

However, a little thought shows that such a simplistic test is inadequate; the reason is that it fails to take into account the scale of the problem. It should be obvious that a given $x^{(k)}$ is an equally good (or bad) approximate solution to

$$\min f(x)$$

and

$$\min \tilde{f}(x)$$

if $\tilde{f}(x) = 10^8 f(x)$. However, for a given $\epsilon_1$, $x^{(k)}$ is likely to fail to satisfy $||\nabla \tilde{f}(x^{(k)})|| \leq \epsilon_1$ even if $||\nabla f(x^{(k)})|| \leq \epsilon_1$ holds. For this reason, the stopping test should take into account the scale of $f$.

Somewhat less obvious is that the scale of $x$ also must be taken into account. The quantity $-||\nabla f(x)||$ is the predicted decrease in $f$ if a step of length one is taken in the steepest descent direction. Whether $||\nabla f(x)||$ is considered large depends on whether a step of length one is large or small. If, for instance, $||x^{(k)}|| = 10^6$, then a step of length one is a very small step, and $||\nabla f(x^{(k)})|| = 10^{-2}$ might be considered large. On the other hand, if $||x^{(k)}|| = 10^{-6}$, then $||\nabla f(x^{(k)})|| = 10^{-2}$ should probably be regarded as rather small.

Because of the above arguments, it is natural to measure the relative gradient $g_{rel}$ of $f$, defined by

$$g_{rel}(x)_i = \lim_{h \to 0} \frac{f(x+he_i) - f(x)}{f(x)} = \frac{\nabla f(x)_i x_i}{f(x)}.$$ 

The iteration can then be stopped if

$$||g_{rel}(x^{(k)})||_\infty = \max \left\{ |g_{rel}(x^{(k)})_i| : i = 1, 2, \ldots, n \right\} \leq \epsilon_1.$$  (4.6)
4.4. Stopping tests for optimization and nonlinear equations

There is still one difficulty with (4.6), though: relative measures are not well-defined when the base quantity (in this case $f(x^{(k)})$ or $x_i$) approaches zero. To protect against this, the user can be asked to provide typical values for $f(x)$ and $|x_i|$, and these are used to protect against division by zero (or a very small quantity). The test then becomes

$$\max \left\{ \frac{\nabla f(x^{(k)})}{\max\{|f(x^{(k)})|, \text{typ}_f\}} \right\} \leq \epsilon_1. \quad (4.7)$$

A simpler version of (4.7) is

$$\frac{||\nabla f(x^{(k)})||}{\max\{|f(x^{(k)})|, \text{typ}_\text{norm}\}} \leq \epsilon_1. \quad (4.8)$$

Condition (4.8) is usually satisfactory, although (4.7) is preferable if the components of $x$ tend to differ by orders of magnitude.

The quantity $\epsilon_1$ in (4.7) or (4.8) is called the gradient tolerance.

The step tolerance

In some problems, it is difficult to satisfy the gradient tolerance, particularly if the gradient is computed only approximately. For this reason, a secondary stopping test is usually implemented that halts the iteration if the step taken becomes small. The reader will recall that, for a superlinearly convergent algorithm, this condition suggests that a good approximation of $x^*$ has been obtained. Since it is important to take the scale of the problem into account in this stopping test as well, the relevant conditions are

$$\max \left\{ \frac{|x_i^{(k+1)} - x_i^{(k)}|}{\max\{|x_i^{(k)}|, \text{typ}_x\}} : i = 1, 2, \ldots, n \right\} \leq \epsilon_2 \quad (4.9)$$

or

$$\frac{||x^{(k+1)} - x^{(k)}||}{\max\{|x^{(k)}|, \text{typ}_\text{norm}\}} \leq \epsilon_2. \quad (4.10)$$

The quantity $\epsilon_2$ is referred to as the step tolerance.

Safeguards

In spite of the fact that there are global convergence theorems for many optimization algorithms, some problems are so nonlinear that convergence is impossible in a reasonable number of iterations. Moreover, users sometimes try to solve incorrectly-posed problems (such as trying to minimize a function that is unbounded below). For this reason, every good optimization code allows the user to specify a maximum number of iterations. If neither the gradient tolerance nor the step tolerance has been satisfied when this limit is reached, the algorithm simply halts and reports failure.
Sometimes another stopping test is implemented specifically to detect divergence (as may occur, for instance, with a unbounded objective function). The algorithm can keep track of the lengths of the steps, and if several consecutive steps are taken that exceed a pre-determined limit, the algorithm assumes that the sequence is diverging.

### 4.4.2 Stopping tests for nonlinear equations

The step tolerance test is equally valid for the problem of solving $F(x) = 0$, as are tests for divergence or an iteration limit. The gradient test should be modified to a properly scaled version of

$$
\|F(x^{(k)})\| \leq \varepsilon_3.
$$

Here there is the difficulty that $F(x^{(k)})$ is supposed to converge to zero, so the obvious tests for the relative size of $F(x^{(k)})$ are meaningless. It is therefore essential that the user provide a typical size for $F(x)$ when $x$ is not too close to the solution (failing input from the user, the value of $F(x^{(0)})$ could be used to determine this size).

The primary stopping test for nonlinear equations therefore takes the form

$$
\max \left\{ \frac{|F(x^{(k)})|}{\text{typ} F_i} : i = 1, 2, \ldots, n \right\} \leq \varepsilon_3 \tag{4.11}
$$

or

$$
\frac{\|F(x^{(k)})\|}{\text{typ} F_{\text{norm}}} \leq \varepsilon_3. \tag{4.12}
$$

In (4.12), $\text{typ} F_{\text{norm}}$ is a positive scalar, while in (4.11) $\text{typ} F$ is a vector of positive scalars. The quantity $\varepsilon_3$ is called the function tolerance.

Finally, an algorithm for nonlinear equations should check explicitly whether the iteration is converging to a local, non-global minimizer of $f(x) = (1/2)\|F(x)\|^2$. A relative gradient check similar to (4.7) or (4.8) should be employed.

All codes, whether for optimization or nonlinear equations, should report which stopping test caused the algorithm to halt.

### 4.5 Computing derivatives by finite differences

In order to use an optimization code to solve a problem, the user must code the objective function (and constraint functions, if any). Coding the derivatives as well can be an imposing task for the user. For this reason, many optimization codes provide the option of approximating the necessary derivatives by finite differences.
4.5.1 Finite differences in one variable

**The forward difference**

To introduce the idea of a finite difference approximation, I begin with Taylor’s theorem for a function $f : \mathbb{R} \rightarrow \mathbb{R}$:

$$f(x + h) = f(x) + f'(x)h + \frac{f''(x + \theta h)}{2}h^2, \, \theta \in (0, 1).$$

Solving the above equation for $f'(x)$ yields

$$f'(x) = \frac{f(x + h) - f(x)}{h} - \frac{f''(x + \theta h)}{2}h.$$  \hfill (4.13)

It would appear possible to approximate $f'(x)$ to any desired accuracy by using (4.13) and just taking $h$ to be very small. However, the effect of finite precision arithmetic cannot be ignored. First of all, since $x$ is represented using a finite number of digits, there exists a threshold $h_0 > 0$ such that, for all $h$ with $|h| < h_0$, $x + h$ and $x$ are indistinguishable in finite precision (this threshold, of course, depends on the particular floating point system being used). In other words, for $h$ smaller than $h_0$, the floating point operation $x_1 = x + h$ would result in $x_1 = x$. The threshold can be expressed in terms of the unit round, which was introduced in Section 3.1.2. In terms of $u$, $h_0 = u|x|$. This suggests that there is an absolute limit to the accuracy with which a forward difference can approximate a derivative, namely

$$O(u|x|).$$

In fact, however, there is a more stringent restriction on $h$. The function $f$ cannot be computed exactly in finite precision arithmetic and so the computed value of $f(x)$ generally differs from the exact value. I will call the computed value $\hat{f}(x)$ and write

$$\hat{f}(x) = f(x) + e(x).$$

If the error $e(x)$ is due only to round-off error, then it would be reasonable to assume that

$$|e(x)| \leq Cu|f(x)|,$$

where $C$ is a constant. It is important to realize, though, that $e$ is likely to be a very nonsmooth function.

The computed forward difference has the form

$$\frac{\hat{f}(x + h) - \hat{f}(x)}{h},$$
and it differs from \( f'(x) \) because of two types of errors: the \textit{truncation error}, that is, the \( O(h) \) error intrinsic to the forward difference approximation, and the error \( e \) in computing \( f \).\(^{15}\) The total error can be written as follows:

\[
f'(x) - \frac{\hat{f}(x + h) - \hat{f}(x)}{h} = f'(x) - \frac{f(x + h) - f(x)}{h} + \frac{f(x + h) - f(x) - \hat{f}(x + h) + \hat{f}(x)}{h} \\
= -\frac{f''(x + \theta h)}{2} h + \frac{f(x + h) - f(x + h) - (f(x) - f(x))}{h} \\
= -\frac{f''(x + \theta h)}{2} h - \frac{e(x + h) - e(x)}{h}.
\]

Since \( e \) is unlikely to be differentiable, the only bound available for the second term above is

\[
\left| \frac{e(x + h) - e(x)}{h} \right| \leq \frac{Cu(|f(x + h)| + |f(x)|)}{h} = \frac{2Cu[f(x + \lambda h)]}{h}, \quad \lambda \in (0, 1).
\]

This suggests that the error in the forward difference due to the error in the computed \( f \) might blow up if \( h \) is taken too small, and in fact this occurs in practice (the reader can verify this with a simple example).

If \(|f| \leq M_1\) and \(|f''| \leq M_2\), then the error in the computed estimate of \( f'(x) \) is then bounded by

\[
\frac{M_2 h^2}{2} + \frac{2CuM_1}{h}.
\]

Ideally, the step size \( h \) would be chosen to minimize (4.14). A simple calculation shows that the optimal value of \( h \) is

\[
h = 2\sqrt{\frac{CM_1}{M_2}} \sqrt{u}.
\]

Since the bounds \( M_1, M_2, \) and \( C \) (particularly the last two) are unlikely to be known, formula (4.15) contains only one piece of useful information: the step size \( h \) should be chosen to be proportional to \( \sqrt{u} \). In the absence of further information, the usual practice is to choose \( h \) to be \(|x|\sqrt{u} \), suitably safeguarded in case \( x \) is close to zero. The following formula is recommended:\(^{16}\)

\[
h = \sqrt{u} \max\{|x|, \text{typx}\} \text{sign}(x)
\]

(sign\( (x) \) is 1 if \( x \geq 0 \) and \(-1 \) otherwise). Therefore, in practice, the forward difference formula can produce estimates with errors on the order of \( \sqrt{u}|x| \).

In many realistic applications, there are significant errors in the computation of \( f(x) \) that do not result simply from round-off. In this case, the unit round \( u \) can be replaced by an estimate \( \eta \) of the error in computing \( f(x) \).

\(^{15}\) There is actually a third type of error, namely, the additional round-off error in computing the difference and the quotient in the forward difference; however, this error is negligible compared to the other errors.

\(^{16}\) By Dennis and Schnabel [5].
4.5. Computing derivatives by finite differences

The central difference

In an optimization problem, the fundamental optimality condition is $f'(x) = 0$, and algorithms are designed to satisfy this condition, at least approximately. For this reason, a highly accurate solution requires an accurate computation of the derivative. Sometimes a more accurate finite difference formula is preferred over the forward difference; the most common is the central difference

$$f'(x) = \frac{f(x + h) - f(x - h)}{2h} + O(h^2). \quad (4.17)$$

This formula and its error can be derived by expanding $f(x \pm h)$ in Taylor series, adding the two series, and solving for $f'(x)$. An analysis of the error in the computed central difference,

$$\frac{\hat{f}(x + h) - \hat{f}(x - h)}{2h}$$

shows that $h$ should be proportional to $u^{1/3}$. This larger value of $h$ is the reason that the central difference formula is more accurate in practice—a larger $h$ reduces the errors propagated from errors in computing $f$. On the other hand, (4.17) has the disadvantage that it is twice as expensive as the forward difference formula, requiring two function values instead of one. (Of course, (4.13) also needs two function values, but in the context of an optimization algorithm, $f(x)$ must be computed anyway, so it is available for “free.”) Most optimization codes use the forward difference because of efficiency concerns, but some have an option to use central differences in case the user wants a more accurate final solution.

4.5.2 Finite differences for the gradient

If $f : \mathbb{R}^n \to \mathbb{R}$ is smooth, then $\nabla f(x)$ can be estimated using finite differences to approximate each partial derivative in turn. For example, using forward differences,

$$\nabla f(x)_i = \frac{\partial f}{\partial x_i}(x) = \frac{f(x + h_i e_i) - f(x)}{h_i},$$

where $e_i$ is the $i$th standard basis vector. The vector $x + h_i e_i$ results from perturbing only the $i$th component of $x$; it is a good idea to compute a different step size $h_i$ for each partial derivative in case the components of $x$ vary in size. Each $h_i$ should be computed via (4.16).

It is important to notice that estimating the gradient using finite differences costs $n$ times as much as evaluating the function (2$n$ times as much if central differences are used). This is a significant expense that may be unbearable if function evaluations are expensive.

4.5.3 Finite differences for the Jacobian

In an algorithm for solving nonlinear equations, the Jacobian of a vector-valued function is needed. The Jacobian is most conveniently estimated one column at a
time:
\[
\frac{\partial F}{\partial x_j}(x) = \frac{F(x + h_j e_j) - F(x)}{h_j},
\]
where \( \frac{\partial F}{\partial x_j} \) represents the \( j \)th column of the Jacobian matrix. As in the case of the gradient, each component \( x_j \) of \( x \) should have its own step size \( h_j \).

### 4.5.4 Finite differences for the Hessian

If the user provides a routine to compute the gradient of \( f : \mathbb{R}^n \to \mathbb{R} \), then the Hessian can be viewed as the Jacobian of the gradient and estimated as above. The resulting matrix is unlikely to be exactly symmetric, so it is important to replace the \( H \) obtained by finite differences by

\[
\frac{1}{2} (H + H^T),
\]

which is necessarily symmetric.

In case the analytic gradient is not available, then the Hessian can be estimated using second-order finite differences, although doing so requires an additional

\[
\frac{n^2 + 3n}{2}
\]

function evaluations. The appropriate formula, which can be derived from Taylor’s theorem, is

\[
\frac{\partial^2 f}{\partial x_i \partial x_j}(x) = \frac{f(x + h_i e_i + h_j e_j) - f(x + h_i e_i) - f(x + h_j e_j) + f(x)}{h_i h_j} + O(\max\{||h_i||, ||h_j||\}).
\]

The main drawback in using finite differences to approximate the Hessian is the expense; in contrast to the case with the gradient, accuracy is not of primary importance.