5. Numerical Optimization Algorithms

5.1. Introduction

Although the optimality criteria in Section 4.4 are powerful tools to find an optimum design, they are limited to apply for solving general engineering optimization problems. They can only be applicable when both the objective and constraints are available as an explicit function of design variables. They require to solve a system of equations, which are composed of the gradients of objective and constraint functions. Therefore, solving optimality criteria is only possible when the analytical expressions of the objective and constraint functions are available. This is a significant limitation because most advanced engineering applications use numerical methods to solve for system responses, from which the quantity of interest (QoI) can be calculated. The finite element method, computational fluid dynamics, and rigid-body dynamics are a short list of examples of numerical methods. For a given design, it is possible to evaluate the values of objective and constraint functions, and possibly their gradients, but their analytical expressions are mostly not available. Due to this critical limitation of optimality criteria, the optimality criteria are used only for limited applications, such as structural optimization [45, 46] and topology optimization [47, 48].

Most engineering optimization problems are solved using numerical methods. Still, the optimality criteria can be used to check the validity of the optimum design. In general, numerical optimization algorithms can be categorized into two groups. The first group of algorithms starts from a design and moves to a new design that can reduce the objective function while satisfying all constraints. This process is repeated until the algorithm cannot find a new design that is better than the previous one. These algorithms are the main focus of this chapter. In order to find a better design, not only the value of functions but also the gradients are used to find the direction to change the design. Because of this reason, the first group of algorithms is referred to as gradient-based algorithms.

The second group of optimization algorithms does not require the gradient information, which is why they are referred to as gradient-free algorithms. These algorithms explore the design space until they cannot find a design that can improve the objective function while satisfying constraints. Sometimes these algorithms are referred to as a global optimization algorithm, but this name can mislead as if they can guarantee to find a global optimum design. As we discussed in Section 4.4, it is possible to guarantee the existence of a global optimum design, but it does not mean that we can always find it. Unfortunately, there is no numerical optimization algorithm that can guarantee to find a global optimum for general problems. A proper name would be global search algorithms, as they tend to explore the entire design space rather than moving from the current design to a better one. We will discuss global search algorithms in Chapter 6.

In numerical optimization algorithms, simulation provides the values of the objective and constraint functions for given design variable. Gradient information is also supplied to the optimization algorithms, which can be obtained using sensitivity analysis [49] or finite difference. Then, the optimization algorithms, discussed in this chapter, calculate the best possible design of the problem. Each algorithm has its own advantages and disadvantages. The performance of an optimization algorithm critically depends on the characteristics of the design problem and the types of objective and constraint functions.

As mentioned earlier, since the entire graphs of the objective and constraint functions are not available, optimization algorithms can make a decision based on the current function value and its sensitivity. Referring to Figure 5-1, let us assume that the current design is located in Point A. Since the gradient of the function is negative at this point, the function will decrease if the design increases. Therefore, the numerical algorithm gradually increases the design until it reaches Point B, where the

gradient becomes zero. At Point B, the objective function will increase if the design is either increase or decrease. Therefore, Point B is claimed as an optimum design.

It is interesting to note that the optimum design at Point B is obtained because the optimization algorithm starts from Point A. It was a shear luck that Point B happens to be the global optimum in this case. If the optimization algorithm starts from Point C, it would end up Point D as an optimum design. Therefore, there is no guarantee to find the global optimum design. A real challenge is that even if the algorithm find the global optimum design at Point B, there is no way to tell if that is the global optimum. In order to increase the chance of finding the global optimum, it would be necessary to repeat the optimization algorithm with different starting points.

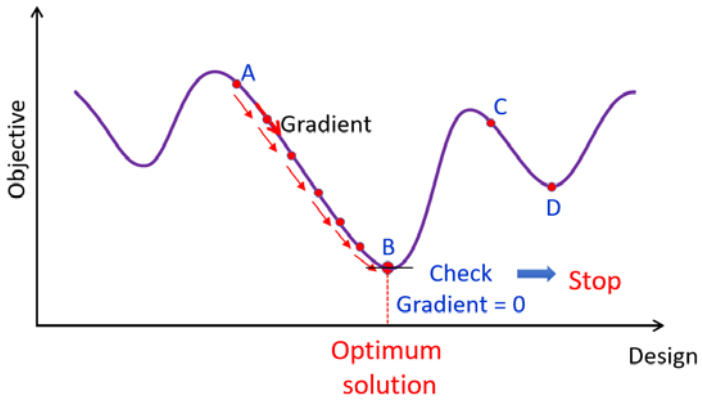


Figure 5-1: Finding an optimum design from a starting point.

5.2. Overview of the numerical optimization process

Table 5-1 shows the procedure of basic optimization algorithms. As mentioned before, gradient-based numerical optimization algorithms start from an initial design and move to a new design that is better than the previous one. Therefore, it is necessary to choose the initial design by the users (Step 1). The initial design may or may not belong to the feasible set, but it is always good to start with a design within the feasible set. For a given design, the optimization algorithm evaluates the values of the objective and constraint functions (Step 2). In most optimization problems, it is considered that the computational cost of the optimization algorithm itself is ignorable compared to that of the evaluation of objective and constraint functions. Therefore, this step takes the major part of computational cost. The algorithm also requires calculating gradient information. Some software has the capability of calculating the gradient information using an adjoint method [49], but most software calculates function values but not gradient information. In such a case, a finite difference method is used to approximate the gradient information, where each design variable is perturbed by a small amount and the simulation is repeated. Since each perturbation requires a full simulation, gradient calculation causes major computational costs. Once the function values and gradient information are available, optimization algorithms calculate the design change (Step 3). This is the key step for optimization algorithms. Different algorithms have different ways of calculating design change $\Delta\mathbf{x}^{(k)}$, which affects the convergence of the algorithm. Once the design is changed, optimization algorithms check if the design is converged (Step 4). It is natural to use optimality criteria to determine the convergence, but various other criteria can be used for this purpose. For example, if the objective and design variables do not change for several iterations, it means that the algorithm cannot improve the objective anymore and is converged. Also, if the algorithm failed to converge within the maximum number of function evaluations or the maximum number of iterations, it may stop without

convergence. If the algorithm is not converged, the design is updated using $\Delta\mathbf{x}^{(k)}$ (Step 5) and move to the next iteration (Steps 6).

Table 5-1: Procedure of basic optimization algorithms

Step	Procedure	Comment
1	Start with $\mathbf{x}^{(k)}$ with $k = 0$	Initial design must be given
2	Evaluate function values and their gradients:	$f^{(k)}, g_i^{(k)}, h_i^{(k)}, \nabla f^{(k)}, \nabla g_i^{(k)}, \nabla h_i^{(k)}$
3	Using information from Step 2, determine design change $\Delta\mathbf{x}^{(k)}$	
4	Check for the termination condition	Stop if converged
5	Update design: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}$	
6	Increase $k = k + 1$ and go to Step 2	Design iteration

Among the steps shown in Table 5-1, the key step in optimization is the determination of design change $\Delta\mathbf{x}^{(k)}$ in Step 3. In most algorithms, the design change is further decomposed into (a) the direction and (b) the magnitude of the design change. The former is often called the search direction and the latter is referred to as the step size. Therefore, the design change can be written as

$$\Delta\mathbf{x}^{(k)} = \alpha_k \mathbf{d}^{(k)} \tag{5.1}$$

where α_k is the step size and $\mathbf{d}^{(k)}$ is the search direction. The search direction is calculated in such a way that the objective function initially decreases along the direction. Due to nonlinearity, however, the function may increase if the design is changed too much. Therefore, the step size determines how much the design should change before the objective function increases again.

5.3. Determination of step size

Descent direction

In general, the search direction is not unique, but not any direction can be a search direction. Therefore, it would be necessary to determine the condition that a search direction must satisfy. It is obvious that the search direction must reduce the objective function. Let us consider an unconstrained optimization problem to minimize the objective function $f(\mathbf{x})$. The required condition is that the objective function should decrease after iteration: $f\left(\mathbf{x}^{(k+1)}\right) < f(\mathbf{x}^{(k)})$. This condition is used in the following Taylor series expansion of $f\left(\mathbf{x}^{(k+1)}\right)$ at $\mathbf{x}^{(k)}$ as

$$\begin{aligned} f\left(\mathbf{x}^{(k+1)}\right) &= f\left(\mathbf{x}^{(k)}\right) + \nabla f\left(\mathbf{x}^{(k)}\right)^T \Delta\mathbf{x}^{(k)} + \text{H. O. T.} \\ &\cong f\left(\mathbf{x}^{(k)}\right) + {\mathbf{c}^{(k)}}^T \Delta\mathbf{x}^{(k)} < f\left(\mathbf{x}^{(k)}\right) \end{aligned} \tag{5.2}$$

where $\mathbf{c}^{(k)} = \nabla f\left(\mathbf{x}^{(k+1)}\right)$ is the gradient of the objective function. In order to satisfy the inequality, the descent direction should satisfy the following condition:

$$\begin{aligned} {\mathbf{c}^{(k)}}^T \Delta\mathbf{x}^{(k)} &< 0 \\ \alpha_k {\mathbf{c}^{(k)}}^T \mathbf{d}^{(k)} &< 0 \end{aligned} \tag{5.3}$$

The search direction that satisfies Eq. (5.3) is called the descent direction. Knowing that the objective function increases in the direction of $\mathbf{c}^{(k)}$, the angle between $\mathbf{c}^{(k)}$ and $\mathbf{d}^{(k)}$ must be greater than 90°.

Example 5-1

An objective function is given as $f(x_1, x_2) = x_1^2 + 3x_1x_2 - 2x_2^2$. When $\mathbf{x} = \{2,0\}$ is the location of the current design, plot the descent directions.

Solution:
At the current design, the gradient of the objective function becomes

$$\mathbf{c}^{(k)} = \begin{Bmatrix} 2x_1 + 3x_2 \\ 3x_1 - 4x_2 \end{Bmatrix} = \begin{Bmatrix} 4 \\ 6 \end{Bmatrix}$$

Therefore, the descent direction is those vectors that have an angle greater than 90°. Figure 5-2 shows possible descent directions. At $\mathbf{x} = \{2,0\}$, the objective function has $f = 4$. The gradient $\mathbf{c}^{(k)}$ of the objective function is normal to the tangent plane. The object function increases along the direction of the gradient. Then, all directions below the tangent plane are descent directions.

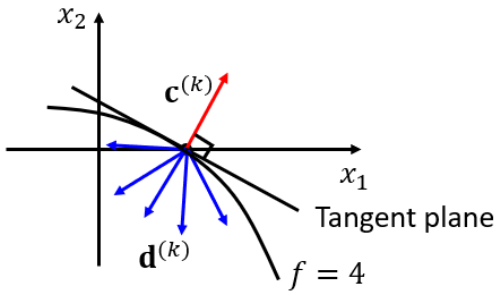


Figure 5-2: Descent directions.

Since there are infinitely many descent directions, various optimization algorithms were developed based on how to determine the search direction $\mathbf{d}^{(k)}$. It seems that $\mathbf{d}^{(k)} = -\mathbf{c}^{(k)}$ is the best search direction as the objective function creases fastest in this direction. However, we will be shown in the next section, it turns out that it is not an efficient direction. This is because the objective function is nonlinear and the best descent direction may change at different designs. Different numerical optimization algorithms will be discussed in Sections 5.4 and 5.5.

Step size termination criterion

The determination of step size α_k is independent of the search direction. For the moment, let us assume that the search direction $\mathbf{d}^{(k)}$ is given. The goal of step size determination is to find α_k that minimizes the objective function in the direction of $\mathbf{d}^{(k)}$. That is, the following 1D optimization problem needs to be solved for the step size:

$$\text{minimize} \quad \phi(\alpha_k) \equiv f(\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}) \tag{5.4}$$

As shown in Figure 5-3, the current design $\mathbf{x}^{(k)}$ is at the y-intercept $\alpha_k = 0$. The objective function has a negative slope at the current design because the search direction $\mathbf{d}^{(k)}$ is a descent direction. The goal is to find that step size α_k that minimizes $\phi(\alpha_k)$.

Similar to the optimality criteria, the minimum point must satisfy $\phi' = 0$ and $\phi'' > 0$. Therefore, the following condition can be obtained:

$$\phi' = \frac{\partial f}{\partial \mathbf{x}^{(k+1)}}^T \frac{\partial \mathbf{x}^{(k+1)}}{\partial \alpha} = \mathbf{c}^{(k+1)^T} \mathbf{d}^{(k)} = 0 \tag{5.5}$$

That is, the minimum point is when the search direction is perpendicular to the gradient of the objective function. The above condition is called the step size termination criterion. The optimum step size α_k should satisfy this condition.

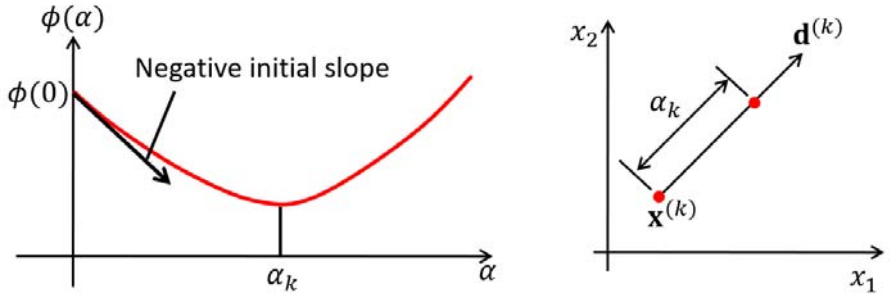


Figure 5-3: Determination of step size in 1D line search.

Example 5-2

In **Example 5-1**, let the descent direction is chosen as $\mathbf{d}^{(k)} = \{-4, -6\}^T$ at the current design $\mathbf{x}^{(k)} = \{2,0\}^T$. Find 1D line search to determine the step size α_k and new design $\mathbf{x}^{(k+1)}$

Solution:

The step size termination criterion in Eq. (5.5) can be written as

$$\mathbf{c}^{(k+1)^T}\mathbf{d}^{(k)} = -4(2x_1 + 3x_2) - 6(3x_1 - 4x_2) = 0$$

The new design point can be written as $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)} = \{2 - 4\alpha_k, -6\alpha_k\}^T$. After substituting this point into the step size termination criterion, we have

$$\mathbf{c}^{(k+1)^T}\mathbf{d}^{(k)} = -4\big(2(2 - 4\alpha_k) + 3(-6\alpha_k)\big) - 6(3(2 - 4\alpha_k) - 4(-6\alpha_k)) = 0$$

The above equation can be solved for $\alpha_k = 1.625$. Therefore, the new design point is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)} = \{2 - 4\alpha_k, -6\alpha_k\}^T = \{-4.5, -9.75\}^T$$

Although $\mathbf{x}^{(k+1)}$ minimizes the objective function in the direction of $\mathbf{d}^{(k)}$, it is not the minimum point of the objective function. This can be verified by calculating the gradient of the objective function in this design, as

$$\mathbf{c}^{(k+1)} = \begin{Bmatrix} 2x_1 + 3x_2 \\ 3x_1 - 4x_2 \end{Bmatrix} = \begin{Bmatrix} -38.25 \\ 25.5 \end{Bmatrix}$$

Therefore, $\mathbf{x}^{(k+1)}$ does not satisfy the KKT condition, and the objective function can further be reduced by moving to a new descent direction.

Although it is best to choose the step size that satisfies the termination criterion, it is unnecessary to find the exact minimum point. This is because it is the minimum only in the direction of $\mathbf{d}^{(k)}$. Once we move to $\mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)}$, the objective function can be reduced in other directions as well. Therefore, instead of wasting computational resources in finding an accurate minimum in 1D line search, it might be more efficient to approximate it.

In this section, we assume that the search direction $\mathbf{d}^{(k)}$ is given and would like to determine the step size α_k. There are many algorithms available to determine the step size. Some algorithms are based on reducing the intervals where the optimum step size exists, while other algorithms are approximating the

objective function along the search direction as a simple polynomial function and finding the optimum step size from it. In this section, we briefly introduce an example of these two types of algorithms. For detailed discussions of step-size determination methods, readers are referred to the book of Arora [39].

Interval reduction method

The first group of algorithms starts from a range of α where the optimum α_k is within the range. It may not be straightforward to determine an appropriate range. A too-large range may slow the process of finding the optimum, while a too-small range may miss it. An important assumption in determining the range is that the objective function shows unimodal behavior within the interval. That is, the objective function initially decreases, and after the optimum step size α_k, it increases again. Since different objective functions have different behaviors, it might be difficult to determine the range that works for general functions. As mentioned before, however, since finding the exact optimum α_k is not critical, the range is often determined heuristically.

Once the initial range is determined, the interval is gradually reduced by removing those sub-intervals that do not have optimum α_k. This is done by computing the objective function at different α's and comparing their values. For example, the equal interval search method is to divide the range into n intervals, to evaluate the function values starting from the first interval, and to stop when $\phi(\alpha_{j-1}) > \phi(\alpha_j)$ and $\phi(\alpha_{j+1}) > \phi(\alpha_j)$. Then, the range is reduced to $[\alpha_{j-1}, \alpha_{j+1}]$, and the process is repeated until the range becomes smaller than a threshold. This process is computationally expensive as it requires multiple simulations. The major disadvantage is that the samples in the previous interval cannot be used in the refined interval.

Among different interval reduction methods, the golden section search method is the most popular and efficient one. This method chooses the sample locations in such a way that the samples in the previous interval can be reused in the refined interval. It is a variable interval method, where the interval size is reduced in a constant ratio. With reference to Figure 5-4, let the interval length at kth iteration be $l^{(k)}$. The function values are evaluated at the two endpoints and two inside points. The two internal points α_3 and α_4 are located at the same distance from the two ends. The reduced interval is selected depending on function values at these sample locations. If $\phi(\alpha_3) < \phi(\alpha_4)$, then the minimum lies between α_1 and α_4, and thus, the new interval becomes $l^{(k+1)} = \alpha_4 - \alpha_1$. If $\phi(\alpha_3) > \phi(\alpha_4)$, then the minimum lies between α_3 and α_2, and thus, the new interval becomes $l^{(k+1)} = \alpha_2 - \alpha_3$. Figure 5-4 shows the case of $\phi(\alpha_3) < \phi(\alpha_4)$. This process is repeated until the interval size is smaller than a predetermined threshold.

In the golden section search, the internal two points (α_3 and α_4) are chosen in such a way that these points can be reused in the next iteration with a reduced interval. As shown in Figure 5-4, α_3 becomes the sample point in the next iteration. Therefore, only one additional point α_5 is required in each iteration, and the interval is reduced to $l^{(k+1)} = rl^{(k)}$. In order to reuse α_3, it is required that $rl^{(k+1)} = (1-r)l^{(k)}$. This requirement ends up being a quadratic equation $r^2 + r - 1 = 0$, whose positive root is $r = (-1 + \sqrt{5})/2 = 0.618$. Thus the two internal points are located at a distance of $0.618l$ or $0.382l$ from either end of the interval.

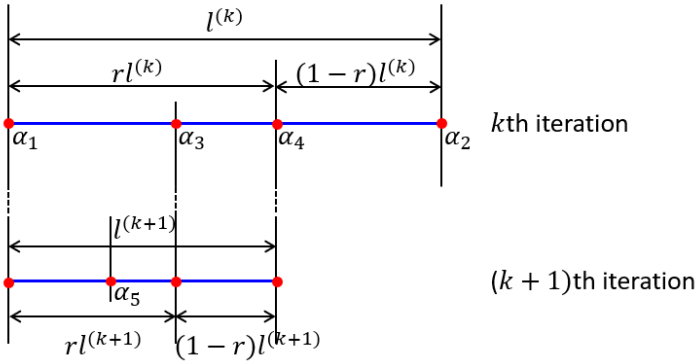


Figure 5-4: Interval reduction in the golden section search method.

Quadratic interpolation method

The second group of algorithms approximates the objective function as a simple polynomial. For example, the quadratic interpolation method evaluates the objective function at three step sizes, α_1, α_2, and α_3. Then, the function is approximated by a quadratic polynomial $\hat{\phi}(\alpha) = a\alpha^2 + b\alpha + c$. Since the polynomial has three coefficients, the three sample points are enough to determine the unknown coefficients. From the KKT condition in Chapter 4, the minimum point can be identified using the derivative information. That is, $\hat{\phi}'(\alpha) = 2a\alpha + b = 0$ and $\hat{\phi}''(\alpha) = 2a > 0$. From these two conditions, we can obtain the optimum step size $\alpha^* = -b/2a$ with $a > 0$ and $b < 0$. Figure 5-5 shows the quadratic approximation of the objective function and the approximate optimum step size.

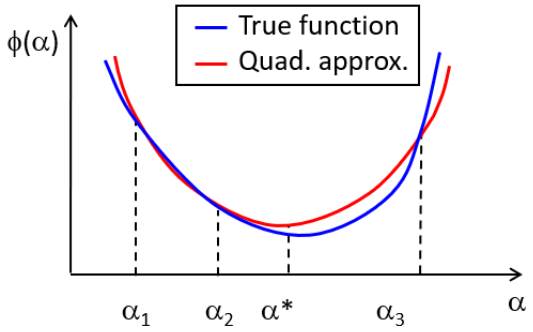


Figure 5-5: Quadratic approximation for step size determination.

Example 5-3

In **Example 5-1**, let the descent direction is chosen as $\mathbf{d}^{(k)} = \{-4, -6\}^T$ at the current design $\mathbf{x}^{(k)} = \{2,0\}^T$. Use the quadratic interpolation method to determine the step size α_k and new design $\mathbf{x}^{(k+1)}$. Use the following three points to sample the objective function: $\alpha = 0, 1$, and 2.

Solution:

The new design point can be written as $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)} = \{2 - 4\alpha_k, -6\alpha_k\}^T$. At the three sample points, the objective function has the following values: $\phi(0) = 4, \phi(1) = -32, \phi(2) = -36$. These three samples are used to approximate a quadratic function $\hat{\phi}(\alpha) = a\alpha^2 + b\alpha + c = 16\alpha^2 - 52\alpha + 4$. Therefore, the optimum step size is $\alpha^* = -b/2a = 1.625$, which happens to be the same as the optimum step size in **Example 5-2**. This is because the original objective function is a quadratic polynomial. Therefore, the new design point is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)} = \{2 - 4\alpha_k, -6\alpha_k\}^T = \{-4.5, -9.75\}^T$$

When the objective function is a quadratic function of design variables, the step size of the line search can be calculated analytically. Consider the following objective function is a quadratic form:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} - \mathbf{x}^T\mathbf{b} \tag{5.6}$$

where **H** is the Hessian matrix and **b** is a constant vector. The gradient of the objective function can be written as $\mathbf{c} = \mathbf{H}\mathbf{x} - \mathbf{b}$. From the KKT condition, the minimum of the objective function is equivalent to solving the linear system of equation $\mathbf{H}\mathbf{x} = \mathbf{b}$. When the search direction **d** is given, the design can be updated to $\mathbf{x} + \alpha\mathbf{d}$. By substituting this new design into the objective function, we have

$$f(\mathbf{x} + \alpha\mathbf{d}) = \frac{1}{2}(\mathbf{x} + \alpha\mathbf{d})^T\mathbf{H}(\mathbf{x} + \alpha\mathbf{d}) - (\mathbf{x} + \alpha\mathbf{d})^T\mathbf{b} \tag{5.7}$$

By differentiating this with respect to α, we can obtain the optimum step size as

$$\alpha = -\frac{\mathbf{d}^T\mathbf{c}}{\mathbf{d}^T\mathbf{H}\mathbf{d}} > 0 \tag{5.8}$$

Although this step size is only true for a quadratic function, it can be used for general nonlinear function as we can approximate them using a quadratic function when the design change is small.

5.4. Unconstrained optimization algorithms

When there are no constraints on the design problem, it is referred to as an unconstrained optimization problem. Even if most engineering problems have constraints, these problems can be transformed into unconstrained ones by using the penalty method, or the Lagrange multiplier method. The unconstrained optimization problem sometimes contains the lower and upper limits of a design variable, since this type of constraint can be treated easily. The standard form of an unconstrained optimization problem can be written as

$$\text{minimize} \quad f(\mathbf{x}) \tag{5.9}$$

In the standard numerical optimization algorithms, the optimum design is found iteratively. Let the superscript (k) be an iteration counter, the design at $k+1$th iteration is updated by $x^{(k+1)} = x^{(k)} + \alpha_k \mathbf{d}^{(k)}$. In the previous section, the methods of determining the optimum step size α_k was discussed assuming that the search direction $\mathbf{d}^{(k)}$ is given. In this section, various methods of determining the search direction $\mathbf{d}^{(k)}$ will be discussed. As mentioned before, there are infinitely many search directions. Therefore, the performance of optimization algorithms depends on the selection of the search direction.

Steepest descent method

Since the search direction needs to be a descent direction, it is intuitive to choose the direction that reduces the objective function the most. It turns out that the objective function increases fastest along the direction of its gradient. This is obvious from the definition of the gradient. It can also be shown that the gradient is normal to the hypersurface of $f(\mathbf{x}) = \text{constant}$. The surface shown in Figure 5-6 represents $f(\mathbf{x}) = \text{constant}$ in 2D design space. Although we cannot plot the surface in a higher-dimensional space, we can name it a hypersurface. On the surface, we can define a parametric curve $\mathbf{x}(s)$, where s is a parametric coordinate. Then, a tangent vector $\mathbf{t}(s)$ of the curve can be defined as $\mathbf{t}(s) = d\mathbf{x}/ds$. That is,

the derivative of the coordinate on the curve with respect to the parametric coordinate s. With the definition of the tangent vector, the derivative of the objective function with respect to s can be defined as

$$\frac{df}{ds} = \frac{df}{d\mathbf{x}}^T \frac{d\mathbf{x}}{ds} = \mathbf{c}^T\mathbf{t} = 0 \tag{5.10}$$

This result is expected as the objective function remains constant along the curve. An important observation from this equation is that the gradient vector $\mathbf{c}$ is perpendicular to the tangent vector $\mathbf{t}$. Therefore, the normal vector to the hypersurface of $f(\mathbf{x}) = \text{constant}$ is the direction that increases the objective function fastest.

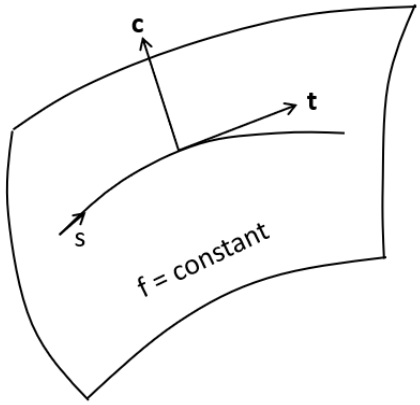


Figure 5-6: Gradient of objective function.

With this observation, the steepest descent method chooses the negative of the gradient as a search direction:

$$\mathbf{d}^{(k)} = -\mathbf{c}^{(k)} \tag{5.11}$$

First, this search direction satisfies the descent condition: $\mathbf{c}^{(k)^T}\mathbf{d}^{(k)} = -\left\|\mathbf{c}^{(k)}\right\|^2 < 0$. This seems the best direction for optimization, but it turns out that this is not a good strategy due to the nonlinearity of the objective function. Especially the convergence of this strategy becomes slow near the optimum point. This is because the search directions in the consecutive two iterations are perpendicular. In order to show this, consider the step-size termination criterion:

$$\mathbf{c}^{(k+1)^T}\mathbf{d}^{(k)} = -\mathbf{c}^{(k+1)^T}\mathbf{c}^{(k)} = 0 \tag{5.12}$$

Therefore, $\mathbf{c}^{(k+1)}$ and $\mathbf{c}^{(k)}$ are perpendicular. This makes the search directions move in a zigzag pattern and slows the convergence. The reason for a slow convergence is that the search direction is determined based on the current information, not using information from the previous iterations. In addition, only the first-order derivatives are used in determining the search direction.

Example 5-4
Consider an elliptical objective function $f(x_1, x_2) = x_1^2 + rx_2^2$. When the initial design is $\mathbf{x}^{(0)} = \{r, 1\}$, show the trajectory of design changes using the steepest descent method. The step size can be calculated using the step-size termination criterion in Eq. (5.5). Plot the contour of the objective function along with the trajectory of design change when $r = 10$.

Solution:
For the given objective function, the gradient vector can be calculated as

$$\mathbf{c}^{(k)} = \left\{2x_1^{(k)}, 2rx_2^{(k)}\right\}^T$$

Therefore, at the initial design, $\mathbf{x}^{(0)} = \{r, 1\}$, the gradient vector and the search direction become $\mathbf{c}^{(0)} = \{2r, 2r\}^T$ and $\mathbf{d}^{(0)} = -\mathbf{c}^{(0)}$. Then, at the next iteration, $k = 1$, the design becomes $\mathbf{x}^{(1)} = \{r - 2r\alpha, 1 - 2r\alpha\}$, and the gradient becomes $\mathbf{c}^{(1)} = \{2r - 4r\alpha, 2r - 4r^2\alpha\}^T$. Therefore, the step-size termination criterion can be used to determine the optimum step size:

$$\mathbf{c}^{(1)^T}\mathbf{d}^{(0)} = 2r(2r - 4r\alpha) + 2r(2r - 4r^2\alpha) = 0 \;\rightarrow\; \alpha_0 = \frac{1}{r+1}$$

Therefore, the design at $k = 1$ can be determined as

$$\mathbf{x}^{(1)} = \{r - 2r\alpha, 1 - 2r\alpha\} = \frac{r-1}{r+1}\{r, -1\}$$

This process is repeated for $k = 2, 3, \cdots$ to obtain the general formulation of the updated design as

$$\mathbf{x}^{(k)} = \left(\frac{r-1}{r+1}\right)^k \{r, (-1)^k\}$$

The following Matlab code plots the contour of the objective function and the trajectory of design changes as shown in Figure 5-7. For the quadratic objective function, it is possible to find the optimum design if the initial design is located along the axis of each design variable because then the initial descent direction is toward the optimum design $\mathbf{x}^* = \{0,0\}$. However, if the initial design is not on the axis, the descent direction is misaligned with the optimum design. Therefore, although the updated designs move toward the optimum design, the show zigzag pattern due to the orthogonal property of two consecutive descent directions.

```
r=10; x0=[r 1];
for k=1:10
 x(k,:)=((r-1)/(r+1))^k*[r (-1)^k];
end
x = [x0; x];
figure(1);hold on; axis equal;
line(x(:,1),x(:,2),'marker','o')
%
x = linspace(-12,12);
y = linspace(-5,5);
[X,Y] = meshgrid(x,y);
Z = X.^2 + r*Y.^2;
contour(X,Y,Z,[0.1 0.5 1 2 5 10 20 30 50 70 100])
```

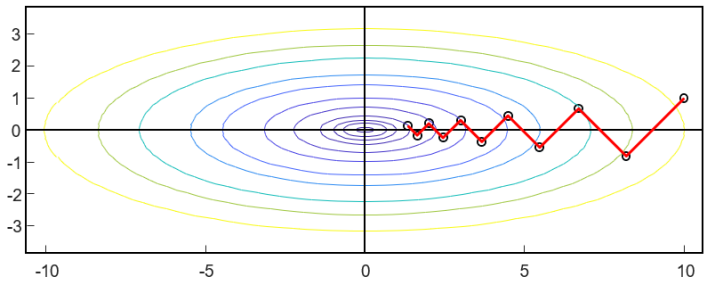


Figure 5-7: Zigzag motion of the steepest descent method.

Conjugate gradient method

The slow convergence of the steepest descent method is because the method only uses the current gradient information. Due to the perpendicular nature of consecutive two gradients, the search directions move in a zigzag pattern as shown in the blue lines in Figure 5-8. However, as shown in the red arrow, the convergence behavior can be improved if the search direction is in the diagonal direction of the two descent directions.

The conjugate gradient method developed by Fletcher and Reeves [50] improves the rate of slow convergence in the steepest descent method by selecting a direction which takes into account not only the gradient direction but also the search direction in the previous iteration. By doing that, it is possible to develop algorithms that are guaranteed to converge to the minimum of an n-dimensional quadratic function in no more than n iterations.

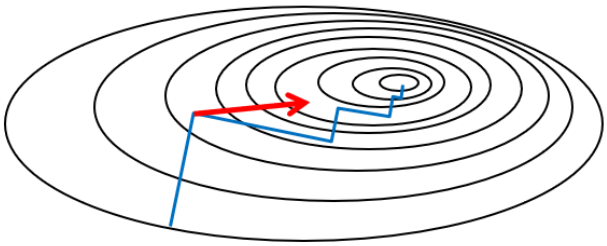


Figure 5-8: Conjugate gradient direction.

The difference in this method is the computation of search direction. The new descent direction is computed by

$$\mathbf{d}^{(k)} = -\mathbf{c}^{(k)} + \beta_k \mathbf{d}^{(k-1)} \tag{5.13}$$

and the first iteration is the same as the steepest descent method, $\mathbf{d}^{(0)} = -\mathbf{c}^{(0)}$. Because of the modification of the descent direction using the previous descent direction, the conjugate directions methods home on the minimum in the second move. The parameter β_k controls how much the previous descent direction contributes to the current one. It is determined based on the magnitude of the gradients as

$$\beta_k = \left(\frac{\left\| \mathbf{c}^{(k)} \right\|}{\left\| \mathbf{c}^{(k-1)} \right\|} \right)^2 > 0 \tag{5.14}$$

This method tends to select the descent direction as a diagonal of the two orthogonal steepest descent directions, such that a zigzagging pattern can be eliminated. This method always has better convergence than the steepest descent method. Since the optimum design will satisfy the KKT condition, the parameter β_k will be zero at the optimum design. It is noted that each move ends at the tangent to the function contour, because we minimize the function in the search direction. Most implementations approximate the function as a quadratic along the search direction, so that if the function is a quadratic the search will end at the minimum in that direction. However, if the function is not quadratic we are likely to stop at a distance from where the direction is tangent to the contour.

Example 5-5

Perform the first two iterations of unconstrained optimization problem in **Example 5-4** using the conjugate gradient method. Use the analytical step size shown in Eq. (5.8).

Solution:
The first iteration of the conjugate gradient method is identical to the steepest descent method. Therefore, $\mathbf{x}^{(0)} = \{10,1\}$, $\mathbf{d}^{(0)} = -\mathbf{c}^{(0)} = \{-20,-20\}$. From Eq. (5.8), $\alpha_0 = -(\mathbf{d}^T\mathbf{c})/(\mathbf{d}^T\mathbf{H}\mathbf{d}) = 0.0909$. The updated design becomes $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_0\mathbf{d}^{(0)} = \{8.1818,-0.8182\}$.

When $k = 1$: $\mathbf{c}^{(1)} = \{16.3636,-16.3636\}$, and

$$\beta_1 = \left(\frac{\|\mathbf{c}^{(1)}\|}{\|\mathbf{c}^{(0)}\|}\right)^2 = 0.6694$$

Therefore, the search direction in the conjugate gradient method becomes

$$\mathbf{d}^{(1)} = -\mathbf{c}^{(1)} + \beta_1\mathbf{d}^{(0)} = \{-29.7521,2.9752\}$$

The step size $\alpha_1 = \mathbf{d}^T(\mathbf{b} - \mathbf{H}\mathbf{x})/(\mathbf{d}^T\mathbf{H}\mathbf{d}) = 0.2750$. The updated design becomes $\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + \alpha_1\mathbf{d}^{(1)} = \{0,0\}$, which is the optimum design. At this point, $\mathbf{c}^{(2)} = \{0,0\}$. Since the objective function is a quadratic function, the conjugate gradient method converges in two iterations. Figure 5-9 shows the contour of the objective function and the trajectory of the design changes. The blue-colored lines are for the conjugate gradient method, while the red-colored lines are for the steepest descent method. By utilizing previous descent direction, the algorithm find the optimum design in two iterations.

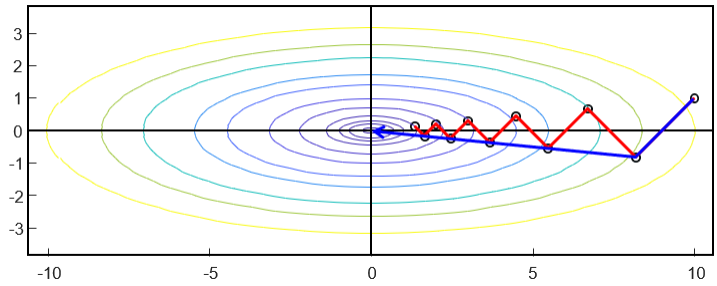


Figure 5-9: Convergence of conjugate gradient method.

Newton method
The previous two methods, the steepest descent and conjugate gradient methods, only use the first-order information (gradients) of the objective function to find the optimum design, which is called a first-order method. The convergence of these methods is generally slow and often require many iterations to find the optimum design. Although the conjugate gradient method in **Example 5-5** converged in two iterations, this happened because the objective function is a quadratic function. For a general nonlinear objective function, the conjugate gradient method may require many iterations as well.

In addition to the gradient information, the Newton's method uses second-order information (Hessian) to approximate the objective function as a quadratic function of the design. The advantage of this method is that the design change $\Delta\mathbf{x}$ that can satisfy the KKT condition can be found by solving a linear system of equations. Therefore, if the objective function is a quadratic function, this method can find the optimum design in one iteration. For a general nonlinear objective function, this method shows a second-order convergence, which is much faster than the first-order convergence.

Let us consider the unconstrained optimization problem in Eq. (5.9). The design variable is updated at $k+1$th iteration by $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}$. Since $\mathbf{x}^{(k)}$ is already determined from kth iteration, the only

variable is $\Delta\mathbf{x}^{(k)}$. Therefore, the objective function is approximated using the Taylor series expansion with respect to the previous design $\mathbf{x}^{(k)}$ as

$$\phi\left(\Delta\mathbf{x}^{(k)}\right) = f\left(\mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}\right) \approx f\left(\mathbf{x}^{(k)}\right) + \mathbf{c}^{(k)^T}\Delta\mathbf{x}^{(k)} + \frac{1}{2}\Delta\mathbf{x}^{(k)^T}\mathbf{H}^{(k)}\Delta\mathbf{x}^{(k)} \tag{5.15}$$

where $\mathbf{H}^{(k)}$ is the Hessian matrix $(n \times n)$, which is made of second-order derivatives. The above equation basically approximates the objective function at the new design as a quadratic function of $\Delta\mathbf{x}^{(k)}$. The goal is to find the design change $\Delta\mathbf{x}^{(k)}$ to minimize the objective function. One can use the KKT condition to find $\Delta\mathbf{x}^{(k)}$ that minimizes $\phi\left(\Delta\mathbf{x}^{(k)}\right)$ as

$$\frac{\partial\phi}{\partial\Delta\mathbf{x}^{(k)}} = \mathbf{c}^{(k)} + \mathbf{H}^{(k)}\Delta\mathbf{x}^{(k)} = 0 \tag{5.16}$$

which yields the following design change:

$$\Delta\mathbf{x}^{(k)} = -\mathbf{H}^{(k)^{-1}}\mathbf{c}^{(k)} \tag{5.17}$$

In practice, since calculating the inverse of the Hessian matrix is expensive, the linear system of equations $\mathbf{H}^{(k)}\Delta\mathbf{x}^{(k)} = -\mathbf{c}^{(k)}$ is solved for $\Delta\mathbf{x}^{(k)}$. If the current estimated design $\mathbf{x}^{(k)}$ is sufficiently close to the optimum design, then Newton's method will show a quadratic convergence.

Albeit its fast convergence, the major concern of Newton's method is how to calculate the Hessian matrix. The greater the number of design variables is, the greater the cost of computing $\mathbf{H}^{(k)}$ is. Although the Hessian information can be calculated using sensitivity analysis [49], it is not available in many application software. Therefore, Newton's method is often limited to cases when the functional form of the objective function is given in terms of design variables.

In addition, the Hessian matrix needs to be positive definite. Otherwise, the design change $\Delta\mathbf{x}^{(k)}$ from Eq. (5.17) can be unstable. The positive definite Hessian matrix means that the objective function should be convex (at least locally). From the descent direction, $\mathbf{c}^{(k)^T}\Delta\mathbf{x}^{(k)} < 0$, the Hessian matrix must satisfy

$$\mathbf{c}^{(k)}\mathbf{H}^{(k)}\mathbf{c}^{(k)} > 0 \tag{5.18}$$

which is the definition of positive definiteness. Therefore, $\Delta\mathbf{x}^{(k)}$is a descent direction when $\mathbf{H}^{(k)}$ is positive definite.

Lastly, Newton's method does not guarantee convergence. The method would find the optimum design if the starting design is close to the optimum design and the objective function is convex. Thus, several modifications are available. For example, the design update algorithm can be modified to include a step size by using a line search. In that case, the linear system of equations solves the search direction,

$$\Delta\mathbf{d}^{(k)} = -\mathbf{H}^{(k)^{-1}}\mathbf{c}^{(k)} \tag{5.19}$$

and the design can be updated as

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\Delta\mathbf{d}^{(k)} \tag{5.20}$$

Example 5-6

Find the optimum design of the unconstrained optimization problem in **Example 5-4** using the Newton's method.

Solution:

Since the objective function is quadratic, the Hessian matrix is constant. With the initial design $\mathbf{x}^{(0)} = \{10,1\}$, the gradient of the objective function is $\mathbf{c}^{(0)} = \{20,20\}$. Therefore, the design change can be found by solving the following linear system of equations:

$$\mathbf{H}\Delta\mathbf{x}^{(0)} = \begin{bmatrix} 2 & 0 \\ 0 & 20 \end{bmatrix} \begin{Bmatrix} \Delta x_1^{(0)} \\ \Delta x_2^{(0)} \end{Bmatrix} = \begin{Bmatrix} -20 \\ -20 \end{Bmatrix} = -\mathbf{c}^{(0)}$$

The above equations can be solved for $\Delta\mathbf{x}^{(0)} = \{-10, -1\}$. Therefore, the updated design becomes $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)} = \{0, 0\}$. With the new design $\mathbf{x}^{(1)}$, $\mathbf{c}^{(1)} = \{0,0\}$, and therefore, it is the optimum design. Since the objective function is a quadratic function of design variables, the Newton's method converges in one iteration.

Quasi-Newton method

Although Newton's method has a quadratic convergence, the cost of computing the Hessian matrix and the lack of a guaranteed convergence are drawbacks to this method. On the other hand, the steepest descent method only requires the gradient information but has a problem of a slow convergence. The main difference between these two methods is the Hessian information; i.e., the second-order derivatives. In a similar way that the gradient information can be calculated using finite difference between two function values, the Hessian information can also be approximated using the difference of two gradients. Quasi-Newton methods are based on this concept and approximate the Hessian information using the gradient information. These methods build an approximation to the Hessian matrix and update it after each gradient calculation. For a quadratic objective function, they are guaranteed to reach the optimum design in no more than n iterations and to have an exact Hessian after n iterations. Due to the approximate Hessian information, these methods show a slower convergence than that of the Newton's method, but faster than that of steepest descent method.

Quasi-Newton methods update the Hessian matrix starting from an identity matrix. An important property of Hessian update is that it needs to maintain the matrix positive definite when an exact line search is used. The calculation of search direction in Eq. (5.19) requires inverting the Hessian matrix or solving a linear system of matrix equations. Since the Hessian matrix is an approximate one, it would be more efficient if the inverse of the Hessian matrix is approximated directly. Therefore, The search direction in Quasi-Newton methods can be determined by

$$\begin{aligned} \mathbf{d}^{(k)} &= -{\mathbf{H}^{(k)}}^{-1}\mathbf{c}^{(k)} \\ \mathbf{d}^{(k)} &= -\mathbf{A}^{(k)}\mathbf{c}^{(k)} \end{aligned} \tag{5.21}$$

where $\mathbf{H}^{(k)}$ is an approximate Hessian matrix and $\mathbf{A}^{(k)}$ is an inverse of the approximate Hessian matrix. After the search direction is determined, the step size is determined using line search. Therefore, the main difference in quasi-Newton methods is how to update the Hessian or its inverse.

In order to show how the Hessian matrix is updated, the following vectors are defined first:

$$\begin{aligned} \mathbf{s}^{(k)} &= \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} = \alpha_k \mathbf{d}^{(k)} \\ \mathbf{y}^{(k)} &= \mathbf{c}^{(k+1)} - \mathbf{c}^{(k)} \end{aligned} \tag{5.22}$$

where $\mathbf{s}^{(k)}$ is the change of design variables, and $\mathbf{y}^{(k)}$ is the change of gradients. Then, the quasi-Newton condition means that the curvature remains constant between two consecutive iterations. Consider the Taylor series expansion of the gradient vector, $\mathbf{c}^{(k+1)} = \mathbf{c}\left(\mathbf{x}^{(k)} + \mathbf{s}^{(k)}\right) = \mathbf{c}^{(k)} + \mathbf{H}^{(k)}\mathbf{s}^{(k)} + H.O.T.$ Therefore, the change of gradient can be approximated by $\mathbf{y}^{(k)} = \mathbf{c}^{(k+1)} - \mathbf{c}^{(k)} \approx \mathbf{H}^{(k)}\mathbf{s}^{(k)}$. By multiplying $\mathbf{s}^{(k)}$ on both sides, we can obtain the curvature at the current design as

$${\mathbf{s}^{(k)}}^T \mathbf{y}^{(k)} = {\mathbf{s}^{(k)}}^T \mathbf{H}^{(k)}\mathbf{s}^{(k)} \tag{5.23}$$

Therefore, the quasi-Newton condition is to update the Hessian matrix while keeping the curvature constant. That is, $\mathbf{s}^{(k)^T}\mathbf{H}^{(k+1)}\mathbf{s}^{(k)} = \mathbf{s}^{(k)^T}\mathbf{H}^{(k)}\mathbf{s}^{(k)} = \mathbf{s}^{(k)^T}\mathbf{y}^{(k)}$. Therefore, the following conditions can be written:

$$\begin{aligned}\mathbf{H}^{(k+1)}\mathbf{s}^{(k)} &= \mathbf{y}^{(k)}\\ \mathbf{A}^{(k+1)}\mathbf{y}^{(k)} &= \mathbf{s}^{(k)}\end{aligned} \tag{5.24}$$

It is obvious that there are infinitely many ways of updating the Hessian matrix while satisfying the quasi-Newton condition in Eq. (5.24). Normally, the Hessian (or its inverse) matrix is updated using a vector at each iteration, starting from an identity matrix. For example, when a single vector $\mathbf{u}$ is used to update the Hessian matrix, it is called rank-1 update: $\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + a_k\mathbf{u}\mathbf{u}^T$. Of course, the magic is to choose vector $\mathbf{u}$ such that the quasi-Newton condition is satisfied while the Hessian matrix is positive definite.

One of the most popular quasi-Newton methods is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [51], which uses a rank-2 update. It means that two vectors, $\mathbf{u}$ and $\mathbf{v}$, are used to update the Hessian matrix as

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + a_k\mathbf{u}\mathbf{u}^T + b_k\mathbf{v}\mathbf{v}^T \tag{5.25}$$

Among infinitely many possibilities, the first vector is chosen as $\mathbf{u} = \mathbf{y}^{(k)}$. Then, the quasi-Newton condition requires the following relation:

$$\mathbf{H}^{(k+1)}\mathbf{s}^{(k)} = \mathbf{H}^{(k)}\mathbf{s}^{(k)} + a_k\mathbf{y}^{(k)}\left(\mathbf{y}^{(k)^T}\mathbf{s}^{(k)}\right) + b_k\mathbf{v}\left(\mathbf{v}^T\mathbf{s}^{(k)}\right) = \mathbf{y}^{(k)} \tag{5.26}$$

In addition, the coefficients a_k and b_k are chosen to normalize the dot-product terms as $a_k = 1/\mathbf{y}^{(k)^T}\mathbf{s}^{(k)}$ and $b_k = -1/\mathbf{v}^T\mathbf{s}^{(k)}$. Then, the quasi-Newton condition becomes

$$\mathbf{H}^{(k+1)}\mathbf{s}^{(k)} = \mathbf{H}^{(k)}\mathbf{s}^{(k)} + \mathbf{y}^{(k)} - \mathbf{v} = \mathbf{y}^{(k)} \tag{5.27}$$

This condition yields $\mathbf{v} = \mathbf{H}^{(k)}\mathbf{s}^{(k)}$. Therefore, the two vectors are determined. In summary, the Hessian matrix update of the BFGS method can be written as

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + \frac{\mathbf{y}^{(k)}\mathbf{y}^{(k)^T}}{\mathbf{y}^{(k)^T}\mathbf{s}^{(k)}} - \frac{\left(\mathbf{H}^{(k)}\mathbf{s}^{(k)}\right)\left(\mathbf{H}^{(k)}\mathbf{s}^{(k)}\right)^T}{\mathbf{s}^{(k)^T}\mathbf{H}^{(k)}\mathbf{s}^{(k)}} \tag{5.28}$$

As mentioned before, this update can keep the Hessian matrix positive definite. In addition, if the objective function is quadratic, $\mathbf{H}^{(n)}$ will be the exact Hessian matrix.

As mentioned before, it would be unnecessary to update the Hessian matrix and invert it to calculate the search direction. Instead, the inverse of the Hessian matrix can be updated directly. The DFP (Davidon-Fletcher-Powell [52]) method approximates the inverse of the Hessian matrix using first-order sensitivity information. Similar to the Hessian update, the inverse of the Hessian matrix is updated by

$$\mathbf{A}^{(k+1)} = \mathbf{A}^{(k)} + a_k\mathbf{u}\mathbf{u}^T + b_k\mathbf{v}\mathbf{v}^T \tag{5.29}$$

By selecting $\mathbf{u} = \mathbf{s}^{(k)}$, the other vector $\mathbf{v} = \mathbf{A}^{(k)}\mathbf{y}^{(k)}$ can be determined from the quasi-Newton condition. Therefore, the inverse of the Hessian matrix is updated as

$$\mathbf{A}^{(k+1)} = \mathbf{A}^{(k)} + \frac{\mathbf{s}^{(k)}\mathbf{s}^{(k)^T}}{\mathbf{y}^{(k)^T}\mathbf{s}^{(k)}} - \frac{\left(\mathbf{A}^{(k)}\mathbf{y}^{(k)}\right)\left(\mathbf{A}^{(k)}\mathbf{y}^{(k)}\right)^T}{\mathbf{y}^{(k)^T}\mathbf{A}^{(k)}\mathbf{y}^{(k)}} \tag{5.30}$$

The updated matrix $\mathbf{A}^{(k+1)}$ has similar properties to the updated Hessian matrix $\mathbf{H}^{(k+1)}$.

Example 5-7

Minimize $f(x_1, x_2) = 12x_1^2 + 4x_2^2 - 12x_1x_2 + 2x_1$ using the BFGS quasi-Newton method with the exact line search starting with the initial design $\mathbf{x}^{(0)} = \{-1, -2\}$.

Solution:
Although it is possible to calculate the exact Hessian matrix by differentiating the objective function twice, it is assumed that the Hessian matrix is unknown and needs to be updated. At the initial step, since $\mathbf{H}^{(0)} = \mathbf{I}$, the search direction is the same as that of the steepest descent method:

$$\nabla f = \begin{Bmatrix} 24x_1 - 12x_2 + 2 \\ 8x_2 - 12x_1 \end{Bmatrix}, \qquad \mathbf{d}^{(0)} = -\mathbf{c}^{(0)} = \begin{Bmatrix} -2 \\ 4 \end{Bmatrix}$$

Using the search direction and step size, the design can be updated as

$$\mathbf{x}^{(1)} = \begin{Bmatrix} -1 \\ -2 \end{Bmatrix} + \alpha_0 \begin{Bmatrix} -2 \\ 4 \end{Bmatrix}$$

The objective function at the updated design becomes $f(\alpha_0) = 12(-1 - 2\alpha_0)^2 + 4(-2 + 4\alpha_0)^2 - 12(-1 - 2\alpha_0)(-2 + 4\alpha_0) + 2(-1 - 2\alpha_0)$. The value of α_0 for which f is the minimum is obtained from the condition $df/d\alpha_0 = 0$, or $\alpha_0 = 0.048077$. Therefore, the updated design and the gradient at that point are

$$\mathbf{x}^{(1)} = \begin{Bmatrix} -1.0961 \\ -1.8077 \end{Bmatrix}, \qquad \mathbf{c}^{(1)} = \nabla f(\mathbf{x}^{(1)}) = \begin{Bmatrix} -2.6154 \\ -1.3077 \end{Bmatrix}$$

Then, the vectors $\mathbf{s}^{(0)}$ and $\mathbf{y}^{(0)}$ can be calculated as

$$\mathbf{s}^{(0)} = \mathbf{x}^{(1)} - \mathbf{x}^{(0)} = \begin{Bmatrix} -0.0961 \\ 0.1923 \end{Bmatrix}, \qquad \mathbf{y}^{(0)} = \mathbf{c}^{(1)} - \mathbf{c}^{(0)} = \begin{Bmatrix} -4.6154 \\ 2.6923 \end{Bmatrix}$$

Therefore, the Hessian matrix in Eq. (5.28) is updated as

$$\mathbf{H}^{(1)} = \mathbf{H}^{(0)} + \frac{\mathbf{y}^{(0)}\mathbf{y}^{(0)^T}}{\mathbf{y}^{(0)^T}\mathbf{s}^{(0)}} - \frac{\left(\mathbf{H}^{(0)}\mathbf{s}^{(0)}\right)\left(\mathbf{H}^{(0)}\mathbf{s}^{(0)}\right)^T}{\mathbf{s}^{(0)^T}\mathbf{H}^{(0)}\mathbf{s}^{(0)}} = \begin{bmatrix} 22.9538 & -12.5231 \\ -12.5231 & 7.7385 \end{bmatrix}$$

Then, the search direction is obtained from Eq. (5.21) as

$$\mathbf{d}^{(k)} = -\mathbf{H}^{(k)^{-1}}\mathbf{c}^{(k)} = \begin{Bmatrix} 1.7608 \\ 3.0186 \end{Bmatrix}$$

And, the design at the next iteration becomes

$$\mathbf{x}^{(2)} = \begin{Bmatrix} -1.0961 \\ -1.8077 \end{Bmatrix} + \alpha_1 \begin{Bmatrix} 1.7608 \\ 3.0186 \end{Bmatrix}$$

The step size α_1 is calculated from the condition $df/d\alpha_1 = 0$, to yield $\alpha_1 = 0.4332$, and

$$\mathbf{x}^{(2)} = \begin{Bmatrix} -0.3333 \\ -0.500 \end{Bmatrix}, \qquad \mathbf{c}^{(2)} = \nabla f(\mathbf{x}^{(2)}) \approx \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$

This implies convergence to the exact solution.

The quasi-Newton method is most popular in commercial optimization algorithms. The Matlab function `fminunc` uses a variant of Newton's method if the analytical gradient information is provided and BFGS if it has to calculate the gradient by finite differences. The variant of Newton's method used by `fminunc` is called a trust-region method. It constructs a quadratic approximation to the function and minimizes it in a box around the current point, with the size of the box adjusted depending on the success of the previous iteration.

Rate of convergence

We used the order of convergence of the steepest descent method and Newton's method without formally defining it. In numerical analysis, the order of convergence and the rate of convergence of a convergent sequence are quantities that represent how quickly the sequence approaches its limit. In the context of optimization, the sequence means the history of design variables. Let $\{x^{(k)}\}$ be the history of design variables ($k = 1,2,\cdots$) and let x^* be the optimum design after convergence. Then the order of convergence means how fast the sequence $\{x^{(k)}\}$ converges to x^*. If the order of convergence is higher, then typically fewer iterations are necessary to converge to the optimum design.

A sequence $\{x^{(k)}\}$ is said to converge to x^* with order p where p is the largest number such that

$$0 \le \lim_{k\to\infty} \frac{\left\|x^{(k+1)} - x^*\right\|}{\left\|x^{(k)} - x^*\right\|^p} < \infty \tag{5.31}$$

In addition, when a sequence has p order of convergence, the rate of convergence is defined as

$$\beta = \lim_{k\to\infty} \frac{\left\|x^{(k+1)} - x^*\right\|}{\left\|x^{(k)} - x^*\right\|^p} \tag{5.32}$$

If $p = 1$, the sequence displays linear convergence, and β must be less than 1 for the sequence to converge. In particular, if $\beta = 0$ when $p = 1$, then the sequence is called super-linear convergence. The steepest descent method shows linear convergence. A quadratic convergence is when $p = 2$. Newton's method shows a quadratic convergence when the initial design is close to the optimum design. The quasi-Newton methods show the order of convergence between 1 and 2.

Example 5-8

Calculate the order and rate of convergence of the sequence $x^{(k)} = a^k, a < 1$, and $x^* = 0$.

Solution:

First, assume that $p = 1$. Then, Eq. (5.31) becomes

$$0 \le \lim_{k\to\infty} \frac{\left\|x^{(k+1)} - x^*\right\|}{\left\|x^{(k)} - x^*\right\|^1} = \frac{a^{k+1}}{a^k} = a < \infty$$

On the other hand, if $p = 2$ is assumed, then

$$0 \le \lim_{k\to\infty} \frac{\left\|x^{(k+1)} - x^*\right\|}{\left\|x^{(k)} - x^*\right\|^2} = \frac{a^{k+1}}{a^{2k}} = \frac{1}{a^{k-1}} \to \infty$$

Therefore, the sequence shows linear convergence. With $p = 1, \beta = a$. Therefore, the rate of convergence is $a < 1$.

5.5. Constrained optimization using unconstrained algorithms

In the previous section, numerical algorithms for solving unconstrained optimization problems were discussed. Most engineering applications, however, have constraints that must be satisfied during the design optimization process. As shown in Figure 5-10, optimums design normally exits on the boundary of constraints. These constraints play a critical role in determining the optimum design. In this section, numerical algorithms that can find the optimum design for constrained optimization problems will be discussed.

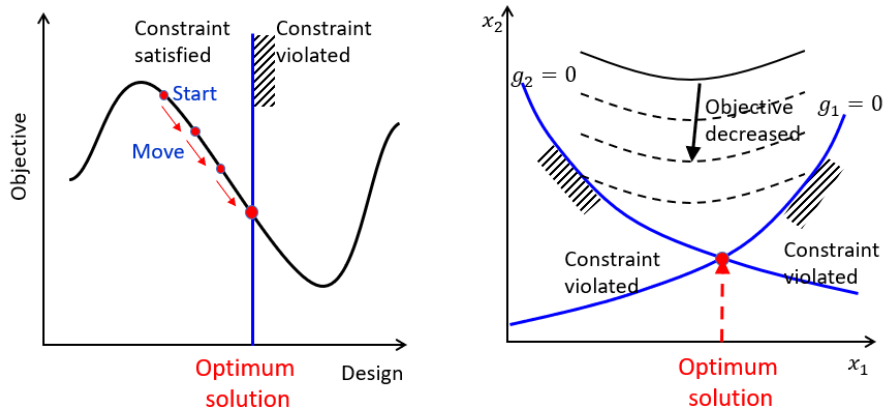


Figure 5-10: The role of constraints on optimum design.

As we discussed in Section 4.4, not all constraints are active at the optimum design. All equality constraints should be active, while some inequality constraints may be inactive. Only those active constraints contribute to the optimum design. When all constraints are inactive, the constraint optimization problem can be considered as an unconstrained problem, it is assumed that at least one constraint should be active at the optimum design.

Numerical algorithms for constrained optimization can be grouped into two types. The first type converts the constrained optimization problem into an unconstrained optimization problem using either the Lagrange multiplier method or the penalty method and solve it using the numerical algorithms in the previous section. The second group solves the constrained optimization problem directly. We will discuss both types of algorithms in the following subsections.

Lagrange multiplier method

The Lagrange multiplier method, along with the penalty method in the next subsection, is a transformation method, where the constrained optimization problem is converted into an unconstrained optimization problem. Then, the problem can be solved using the unconstrained optimization algorithms in the previous section. Consider the constrained optimization problem in Eq. (4.3), which is rewritten here:

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) \\ &\text{subject to } g_i(\mathbf{x}) \le 0, \quad i = 1, \cdots, K \\ &\qquad\qquad h_j(\mathbf{x}) = 0, \quad j = 1, \cdots, M \\ &\qquad\qquad x_l^L \le x_l \le x_l^U, \quad l = 1, \cdots, n \end{aligned} \tag{5.33}$$

where $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}^T$ is the vector of design variables, $f(\mathbf{x})$ is the objective function, $g_1(\mathbf{x}), \cdots, g_K(\mathbf{x})$ are inequality constraints, $h_1(\mathbf{x}), \cdots, h_M(\mathbf{x})$ are equality constraints, and $\mathbf{x}^L$ and $\mathbf{x}^U$ are lower- and upper-bounds of design variables, respectively. Since only active constraints contribute to the optimum design, let the number of active inequality constraints be $L < K$. Since the active inequality constraints are equivalent to equality constraints, the optimization problem can be considered as a constrained optimization with $M + L$ equality constraints.

The Lagrange multiplier method is based on the Lagrangian function defined in Eq. (4.36), which is rewritten here:

$$\text{minimize } L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{j=1}^{M} \mu_j h_j(\mathbf{x}) + \sum_{i=1}^{L} \lambda_i g_i(\mathbf{x}) \tag{5.34}$$

It is noted that the slack variables are not used because the Lagrangian function is defined only with active inequality constraints. The minimization of the unconstrained optimization problem in Eq. (5.34) is equivalent to that of the constrained optimization problem in Eq. (5.33). The only difference is that in addition to design variables, the Lagrange multipliers are also considered unknown variables. Therefore, the objective function is the Lagrangian function, and the variables are design variables and Lagrange multipliers. It is noted that the Lagrange multipliers of inequality constraints have a sign restriction, which can easily be imposed using side constraints.

Although this seems a straightforward conversion, there are two challenges associated with the Lagrange multiplier method. The first challenge is that the number of active constraints varies during design iterations. Therefore, some Lagrange multipliers may disappear while others are introduced in the middle of design iterations, which causes difficulty in initializing and updating them. It is possible that the Lagrange multipliers for all constraints are kept throughout all design iterations and those multipliers associated with inactive constraints are set to a zero value. However, this still can cause a problem especially when some constraints oscillate between active and inactive states. The second challenge is from the positive definite property of the Hessian matrix. Many Newton or quasi-Newton methods use the positive definite Hessian matrix or its approximation to enhance the convergence speed of design iterations. However, the second-order derivatives of the Lagrange multiplier are zero. Therefore, the diagonal terms of the Hessian matrix are zero, which makes the Hessian matrix positive semi-definite. Due to these two challenges, the Lagrange multiplier method is not popular in practical use.

Penalty function method

The penalty function method converts the constrained optimization problem to an unconstrained one by adding a penalty for violating constraints. A quadratic penalty is usually employed in order to make the penalty function smooth and preserve differentiability. The penalty for equality constraints is proportional to their square, while for inequality constraints it is their square when they are positive and zeroes otherwise. Therefore, the quadratic loss function is defined as

$$\Phi(\mathbf{x}, \omega) = f(\mathbf{x}) + P(\mathbf{h}, \mathbf{g}, \omega) = f(\mathbf{x}) + \omega \left\{ \sum_{j=1}^{M} \left(h_j(\mathbf{x})\right)^2 + \sum_{i=1}^{L} \left(g_i^+(\mathbf{x})\right)^2 \right\} \tag{5.35}$$

where ω is a penalty parameter, and $g_i^+(\mathbf{x}) = \max(0, g_i(\mathbf{x}))$ is only nonzero when the inequality constraint is violated.

The disadvantage of a quadratic penalty is that it is very small for small violation, so the solution will tend to violate some constraints. This can be minimized by multiplying the penalty by a large parameter ω, but that makes the problem numerically ill-conditioned. Therefore, the established procedure is to gradually increase ω as one gets closer to the solution. This is referred to as the sequential unconstrained minimization technique (SUMT). As the penalty parameter approaches infinity, the solution converges to the original optimization problem. The advantage of the penalty function method compared to the Lagrange multiplier method is that it does not introduce additional variables, but it has a disadvantage of numerical ill-conditioning of the Hessian matrix.

Because of this ill-conditioned Hessian matrix, the penalty function method is not popular any more for gradient-based algorithms, but it is still popular for gradient-free global search algorithms. This is

because most global search algorithms in Chapter 6 do not have a systematic way of including constraints. Therefore, the penalty function method is frequently used for them.

Example 5-9
Find the optimum design of the following constrained optimization problem by gradually increasing the penalty parameter from $\omega = 1$ to $\omega = 1{,}000$:

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) = x_1^2 + 10x_2^2 \\ &\text{subject to } h(\mathbf{x}) = 4 - x_1 - x_2 = 0 \end{aligned}$$

Solution:
In order to apply for the penalty function method, the following loss function is defined:

$$\Phi(\mathbf{x}, \omega) = x_1^2 + 10x_2^2 + \omega(4 - x_1 - x_2)$$

Since this is a quadratic function, it is possible to find the optimum design analytically. From the KKT condition, the optimum design can be obtained as

$$x_1 = \frac{40\omega}{10 + 11\omega}, \qquad x_2 = \frac{4\omega}{10 + 11\omega}$$

The following table shows the solution, the objective and the loss function for a series of increasing ω values. The difference between f and Φ is the penalty. For low values of ω the constraint is violated a lot, the penalty is high, but the objective is small. As we increase the penalty parameter the constraint violation decreases fast enough so that the total penalty actually decreases! For the highest penalty parameter, the sum of the variables is 3.966, so that the violation is smaller than 1%, but that may not be acceptable, in which case an even higher penalty parameter would be required.

ω	x_1	x_2	f	Φ
1	1.905	0.1905	3.992	7.619
10	3.333	0.3333	12.220	13.333
100	3.604	0.3604	14.288	14.144
1,000	3.633	0.3633	14.518	14.532

The contour of the augmented function when $\omega = 1$ as shown in Figure 5-11(a) is well behaved. Even if we obtain the optimum design analytically, it is expected that numerical algorithms would converge easily as well. On the other hand, when $\omega = 1{,}000$ as shown in Figure 5-11(b), the contour shows a canyon at the constraint boundary. For a linear constraint this may not be much of a problem, but if the constraint is nonlinear, an optimization algorithm that moves in straight lines would require large number of iterations and may get stuck. Therefore, it is expected that numerical optimization with a large value of the penalty parameter may experience difficulty in convergence. For gradient-free algorithms, we may avoid this problem by using a penalty that is proportional to the violation instead of its square, so that the penalty parameter would not need to be very high.

The contours in Figure 5-11 were obtained with the following Matlab script :

```
r=1;
x=linspace(1,5,41);   y=linspace(0.1,0.5,41);
[X,Y]=meshgrid(x,y);
Z=X.^2+10*Y.^2+r*(4-X-Y).^2;
cs=contour(X,Y,Z);   clabel(cs);
```

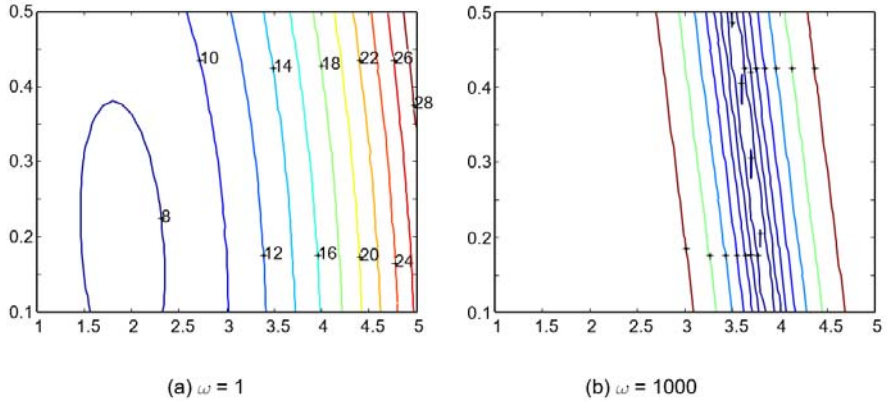


Figure 5-11: Contours of the loss function with different penalty parameters.

5.6. Constrained optimization using direct methods

In the previous section, constrained optimization problems were converted into unconstrained optimization problems using the Lagrange multiplier method and the penalty function method. However, in practice, it is popular to solve constrained optimization problems directly. The basic idea of direct methods is to calculate a search direction based on the gradients of both the objective function and the constraints. The direct methods for constrained optimization have an analogy of a person with a flashlight trying to go down a hill in a terrain with fenced areas. The flashlight can only provide a limited information of slopes (gradients), and the person uses them to select a direction to move. Then, the person can move until the slope changes upwards or the person meets with constraint boundaries.

Some algorithms move mostly away from constraint boundaries and stay inside of the feasible domain. Since the optimum design usually is located on constraint boundaries, they approach to the boundaries from the inside of the feasible domain. Some other algorithms follow the constraint boundaries. The gradient projection algorithm starts with a point on the boundary of the feasible domain and moves in the plane tangent to all the active constraints. The direction is the projection of the gradient on that plane. This move will end either when a new constraint is encountered or when the move will take it too far from the constraint boundaries due to their nonlinearity. Then there is a restoration move that brings it back to the constraint boundary.

Similar to the loss function in the previous section, the direct methods usually define a descent function or a merit function. The merit function must be reduced from one iteration to another and should be the same as the objective function $f(x)$ at the optimum design.

A practical difficulty associated with constraint boundaries is the oscillation between active/violated and inactive constraints. A constraint is considered inactive when $g(\mathbf{x}) < 0$, active when $g(\mathbf{x}) = 0$, and violated when $g(\mathbf{x}) > 0$. The oscillation frequently happens when the design is close to the constraint boundary. This oscillation can cause technical difficulty as the constraints may not consider consistently at different iterations. In order to overcome this difficulty, inactive constraints that are close to the constraint boundaries are also included in the active constraint set. The ϵ-active constraint set is defined as

$$I_\epsilon = \{i |\, g_i + \epsilon \geq 0\} \tag{5.36}$$

where $\epsilon > 0$ is a small positive constant to determine the value of constraint that can be considered active. The ϵ-active constraint set includes all active/violated constraints as well as those constraints that are close to the constraint boundary.

Sequential linear programming (SLP) method

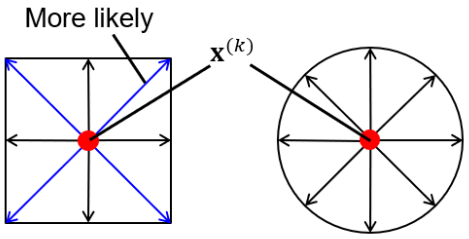
The sequential linear programming (SLP) method approximates the nonlinear problem as a sequence of linear programming problems such that the simplex method in Section 1.7.1 may be used to find the solution to each iteration. By using function values and sensitivity information, the nonlinear problem in Eq. (5.33) is linearized in a similar way as Taylor's expansion method in the first order. Let the linear approximation of the objective function be $f\left(x^{(k+1)}\right) \approx f\left(x^{(k)}\right) + \mathbf{c}^T \Delta\mathbf{x}$, that of equality constraint $h_i\left(\mathbf{x}^{(k+1)}\right) \approx h_i\left(\mathbf{x}^{(k)}\right) + \nabla h_i^T \Delta\mathbf{x}$, and that of active inequality constraint $g_j\left(\mathbf{x}^{(k+1)}\right) \approx g_j\left(\mathbf{x}^{(k)}\right) + \nabla g_j^T \Delta\mathbf{x}$. In order to simplify notations, the gradient vectors are defined as $\nabla h_i = \mathbf{n}_i$ and $\nabla g_j = \mathbf{a}_j$. Also, the gradient matrix is defined as $\mathbf{N} = [\mathbf{n}_1, \cdots, \mathbf{n}_M]$ and $\mathbf{A} = [\mathbf{a}_1, \cdots, \mathbf{a}_L]$. Then, the constrained optimization problem in Eq. (5.33) at iteration k can be written as

$$\begin{aligned} &\text{minimize } \mathbf{c}^T \Delta\mathbf{x} \\ &\text{subject to } \mathbf{n}_i^T \Delta\mathbf{x} + h_i = 0, \quad i = 1, \cdots, M \\ &\qquad\qquad \mathbf{a}_j^T \Delta\mathbf{x} + g_i \le 0, \quad j = 1, \cdots, K \\ &\qquad\qquad \Delta x_l^L \le \Delta x_l \le \Delta x_l^U, \quad l = 1, \cdots, n \end{aligned} \tag{5.37}$$

That is, the original nonlinear optimization problem with respect to $\mathbf{x}$ is converted into a linear optimization problem with respect to $\Delta\mathbf{x}$. There are many efficient algorithms to solve linear optimization problems, such as the simplex method. Once the design increment $\Delta\mathbf{x}$ is solved, the design is updated by $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}$. Therefore, the SLP method sequentially linearizes the nonlinear optimization problem, where its name comes from. It is noted that the gradient information is critical to construct the linear optimization problem.

Quadratic programming (QP) subproblem

In solving the linear optimization problem in Eq. (5.37) for $\Delta\mathbf{x}$, the move limits (side constraints or step size constraints) are critically important for convergence. Consider the move limits of two-dimensional design variables as shown in Figure 5-12(a). The red dot is the current design $\mathbf{x}^{(k)}$ and the square is the move limits. As can be shown in the figure, the length of design change $\Delta\mathbf{x}$ is large toward the corner of move limits. Therefore, design tends to change along the corners. In order to prevent such a bias, the move limits can be given in circular domain as shown in Figure 5-12(b). In this case, all design changes have the same length, and there is no preferred direction.



(a) Rectangular move limits (b) Equal-size move limits

Figure 5-12: Box-shaped move limits versus equal-size move limits.

In order to apply the equal-size design change, the linearized optimization problem in Eq. (5.37) is modified as

(5.38)

$$\frac{1}{2}\mathbf{d}^T\mathbf{d} \le \xi^2$$

Instead of design change $\Delta\mathbf{x}$, the search direction $\mathbf{d}$ is used as additional line search can be added to find the optimum design change. The last term in Eq. (5.38) impose the step-size constraint where the length of design change should be less than a pre-determined threshold. The only difficulty in Eq. (5.38) is that the constraints are nonlinear, and many linear programming algorithms, such as the simplex method, cannot be used.

Instead of solving Eq. (5.38), it is possible to show that the following quadratic programming (QP) problem is equivalent to the linearized constrained optimization with equal-size constraints in Eq. (5.38):

$$\begin{aligned} &\text{minimize } \mathbf{c}^T\mathbf{d} + \frac{1}{2}\mathbf{d}^T\mathbf{d} \\ &\text{subject to } \mathbf{N}^{\mathrm{T}}\mathbf{d} = -\mathbf{h} \\ &\qquad\qquad \mathbf{A}^T\mathbf{d} \le -\mathbf{g} \end{aligned} \tag{5.39}$$

Since the objective function of the QP is convex and since the feasible set of linear constraints is a convex set, Eq. (5.39) is a convex optimization problem, and a local optimum becomes the global optimum design. There are many numerical algorithms to solve for the QP in Eq. (5.39). This QP can be effectively solved, for example, by using the Kuhn-Tucker condition and the simplex method. Starting from the identity matrix, the Hessian matrix $\mathbf{H}$ is updated at each iteration by using the aforementioned methods in unconstrained optimization algorithms. The advantage of solving Eq. (5.39) in this way is that for positive definite $\mathbf{H}$, the problem is convex and the solution is unique. Moreover, this method does not require the move limit as in SLP.

Constrained steepest descent method

In the steepest descent algorithm detailed in Section 5.4, the descent direction $\mathbf{d}$ is obtained from the gradient of the objective function; $\mathbf{d} = -\mathbf{c}$. When constraints exist, this descent direction has to be modified in order to include their effect. In the constrained steepest descent method, this is accomplished by defining the following descent function with violated constraints:

$$\Phi(\mathbf{x}, \omega, \mathbf{h}, \mathbf{g}) = f(\mathbf{x}) + \omega V(\mathbf{h}, \mathbf{g}) \tag{5.40}$$

where $V(\mathbf{h}, \mathbf{g})$ is the maximum constraint violation and ω is a penalty parameter. The penalty parameter is user-defined but has to be larger than the absolute sum of all Lagrange multipliers. The maximum constraint violation is defined as

$$V(\mathbf{h}, \mathbf{g}) = \max\{0, |h_i|, g_j, j \in I_\epsilon\} \tag{5.41}$$

If constraints are violated, then these constraints are added to the objective function using a penalty method. The gradient of the descent function combines the effects of the objective function and the violated constraint functions. Table 5-2 summarizes the procedure of the constrained steepest descent algorithm.

Table 5-2: Procedure of the constrained steepest descent algorithm

Step	Procedure	Comment
1	Set $\mathbf{x}^{(0)}, k = 0, \omega_0 = 1$	Initial design must be given
2	Compute $f(\mathbf{x}^{(k)}), h_i(\mathbf{x}^{(k)}), g_j(\mathbf{x}^{(k)}), \mathbf{c}, \mathbf{N}, \mathbf{A}, V$	
3	Using QP subproblem, solve for $\mathbf{d}^{(k)}$	
4	Check for convergence $\|\mathbf{d}^{(k)}\| < \epsilon_2$ and $V < \epsilon_1$	Stop if converged
5	Update ω_k	

6	Line search: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$	
7	Increase $k = k + 1$, go to step 2	Design iteration

Feasible direction method

The feasible direction method is designed to allow design movement within the feasible region in each iteration. Based on the previous design, the updated design reduces the objective function and remains in the feasible region. Since all designs are feasible, a design at any iteration can be used, even if it is not an optimum design. Since this method uses the linearization of functions as in SLP, it is difficult to maintain nonlinear equality constraints. Thus, this approach is used exclusively for inequality constraints. Search direction $\mathbf{d}$ can be found by solving the following linear subproblem:

$$\begin{aligned} &\text{minimize } \bar{f} = f_0 + \mathbf{c}^T \mathbf{d} \\ &\text{subject to } \bar{g}_j = g_{j0} + \mathbf{a}_j^T \mathbf{d} \le 0 \end{aligned} \tag{5.42}$$

Knowing that $g_{j0} \le 0$, the direction that satisfies $\mathbf{a}_j^T \mathbf{d} \le 0$ is a feasible direction; i.e., the direction maintains the constrained satisfied. In addition, the direction that satisfies $\mathbf{c}^T \mathbf{d} \le 0$ is a useful direction as it reduces the objective function. Therefore, the goal is to find the direction that satisfies both the feasible and useful directions. This can be achieved by defining a new variable $\beta = \max\{\mathbf{c}^T \mathbf{d}, \mathbf{a}_j^T \mathbf{d}\}$ and solve the following optimization problem:

$$\begin{aligned} &\text{minimize } \beta(\mathbf{d}) \\ &\text{subject to } \mathbf{a}_j^T \mathbf{d} \le \beta \\ &\qquad\qquad \mathbf{c}^T \mathbf{d} \le \beta \\ &\qquad -1 \le \|\mathbf{d}\| \le 1 \end{aligned} \tag{5.43}$$

After finding a direction $\mathbf{d}$ that can reduce cost function and maintain feasibility, a line search is used to determine the step size.

Constrained quasi-Newton method

In the case of unconstrained optimization in Section 5.5, it was shown that the Newton method has a quadratic convergence, while the steepest descent method only shows linear convergence. It was also shown that since the Hessian information is expensive to calculate, quasi-Newton methods were used to approximate the Hessian or its inverse matrix. A similar approach can also be taken in constrained optimization problems.

The constrained quasi-Newton methods, or sequential quadratic programming methods, are based on solving the nonlinear KKT conditions using Newton's method. Since Newton's method requires the Hessian matrix, it is approximated using the quasi-Newton method. In order to explain the constrained quasi-Newton method, consider the following constrained optimization problem with equality constraints alone:

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) \\ &\text{subject to } h_i(\mathbf{x}) = 0 \end{aligned} \tag{5.44}$$

whose Lagrangian function becomes $L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum \lambda_i h_i(\mathbf{x})$. The KKT optimality condition can be written in the form of a nonlinear equation. First, let us define the combined variables $\mathbf{y} = \{\mathbf{x}, \boldsymbol{\lambda}\}^T$. Then the KKT condition can be written as

$$\begin{cases} \nabla_{\mathbf{x}} L = \nabla_{\mathbf{x}} f + \mathbf{N}\boldsymbol{\lambda} = 0 \\ \nabla_{\boldsymbol{\lambda}} L = \mathbf{h} = 0 \end{cases} \quad \rightarrow \quad \mathbf{F}(\mathbf{y}) = \begin{Bmatrix} \nabla_{\mathbf{x}} f + \mathbf{N}\boldsymbol{\lambda} \\ \mathbf{h} \end{Bmatrix} = 0 \tag{5.45}$$

The constrained quasi-Newton method solves the above nonlinear equations of the KKT condition using Newton's method. From Taylor series expiation, $\mathbf{F}\left(\mathbf{y}^{(k+1)}\right) = \mathbf{F}\left(\mathbf{y}^{(k)}\right) + \left(\nabla_{\mathbf{y}}\mathbf{F}\right)^T \Delta\mathbf{y}^{(k)} = 0$. By replacing $\Delta\mathbf{y} = \{\mathbf{d}, \Delta\boldsymbol{\lambda}\}^T$, this linearized incremental equation can be written as

$$\begin{bmatrix} \nabla_{\mathbf{xx}} L & \mathbf{N} \\ \mathbf{N}^T & 0 \end{bmatrix} \begin{Bmatrix} \mathbf{d}^{(k)} \\ \boldsymbol{\lambda}^{(k+1)} \end{Bmatrix} = -\begin{Bmatrix} \nabla_{\mathbf{x}} f \\ \mathbf{h} \end{Bmatrix} \tag{5.46}$$

In the above equation, $\Delta\boldsymbol{\lambda}^{(k)} = \boldsymbol{\lambda}^{(k+1)} - \boldsymbol{\lambda}^{(k)}$ is used to replace the unknown variables $\Delta\boldsymbol{\lambda}^{(k)}$ with $\boldsymbol{\lambda}^{(k+1)}$. This is the kth iteration to find the search direction and the Lagrange multiplier for the KKT condition.

In practice, instead of solving Eq. (5.46), an equivalent quadratic problem is solved. It is left as an exercise problem to show that the following quadratic problem is equivalent to solving Eq. (5.46), as

$$\begin{aligned} &\text{minimize } \mathbf{c}^T\mathbf{d} + \frac{1}{2}\mathbf{d}^T[\nabla_{\mathbf{xx}} L]\mathbf{d} \\ &\text{subject to } \mathbf{N}^T\mathbf{d} = -\mathbf{h} \end{aligned} \tag{5.47}$$

As discussed before, the Hessian matrix is expensive to calculate. Therefore, the constrained quasi-Newton method use the BFGS method to approximate it.

When there are both equality and inequality constraints, the equivalent quadratic problem is defined using equality constraints and active inequality constraints as

$$\begin{aligned} &\text{minimize } \mathbf{c}^T\mathbf{d} + \frac{1}{2}\mathbf{d}^T[\nabla_{\mathbf{xx}} L]\mathbf{d} \\ &\text{subject to } \mathbf{N}^T\mathbf{d} = -\mathbf{h} \\ &\qquad\qquad \mathbf{A}^T\mathbf{d} \le -\mathbf{g} \end{aligned} \tag{5.48}$$

Once the search direction $\mathbf{d}^{(k)}$ is found by solving the quadratic programming problem, the line search algorithm can be used to find the step size α_k and update the design $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{d}^{(k)}$.

5.7. Matlab optimization toolbox

Matlab provides various optimization functions, as shown in Table 5-3. Among various optimization algorithms, unconstrained optimization problems can be solved using fminunc and fminsearch. For constrained optimization problems with nonlinear equality and inequality constraints, fmincon is the most common function. In this section, brief explanations of these functions with their algorithms are introduced along with several examples.

Table 5-3: Matlab optimization functions

Type	Function
Minimization of a scalar function	fminbnd
Unconstrained minimization	fminunc, fminsearch
Linear programming	linprog
Quadratic programming	quadprog
Constrained minimization	fmincon
Goal attainment	fgoalattain
Minmax	fminmax
Semi-infinite minimization	fseminf
Binary integer programming	bintprog

Example 5-10

Solve the following constrained optimization problem using the Matlab fmincon function:

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) = \mathrm{e}^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1) \\ &\text{subject to } g_1(\mathbf{x}) = x_1x_2 - x_1 - x_2 + 1.5 \le 0 \\ &\qquad\qquad g_2(\mathbf{x}) = -x_1x_2 - 10 \le 0 \end{aligned}$$

Solution:

In order to call fmincon, it would be necessary to make m-files for the objective function and constraints. First, create the following m-file named objfun.m:

```
function f = objfun(x)
f = exp(x(1))*(4*x(2)^2 + 2*x(2)^2 + 4*x(1)*b(2) + 2*x(2) + 1);
```

Also, create the following m-file named confun.m

```
function [c, ceq] = confun(x)
% Nonlinear inequality constraint
c = [1.5 + x(1)*x(2) – x(1) – x(2);
      -x(1)*x(2) – 10];
% Nonlinear equality constraints
ceq = [];
```

Then, optimization can be invoked by using the following Matlab script:

```
x0 = [-1,1]; % Make a starting guess at the solution
options = optimset('LargeScale','off');
[x, fval, exitflag, output, lambda, grad, hessian] = ...
fmincon(@objfun,x0,[],[],[],[],[],[],@confun, options)
```

First, design variables are stored in the array x. The function objfun.m calculates the objective function value and return in the variable f. On the other hand, the function confun.m calculates an array of inequality constraints c and an array of equality constraints ceq. In this particular optimization problem, there are two inequality constraints and no equality constraint. Therefore, c is 2×1 array, while ceq is null. With objfun.m and confun.m being available, the Matlab script assigns the initial design to x0, defines optimization options in options, and calls the function fmincon. The function optimset assign various options that control the optimization process. Table 5-5 summarizes the available options for Matlab optimization functions, which include algorithms, convergence criteria, result displace, etc. For detailed explanations of the optimset function, the users are referred to the Matlab manual.

The function fmincon in this example returns the following outputs:

```
x =    -3.4049    1.1135
fval =   -0.1494
exitflag =     1
output =
         iterations: 13
          funcCount: 45
    constrviolation: 0
           stepsize: 1.2743e-05
          algorithm: 'interior-point'
      firstorderopt: 4.0000e-07
       cgiterations: 0
            message: 'Local minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing
in ↵feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint
tolerance. <stopping criteria details> Optimization completed: The
```

```
relative first-order optimality measure, 4.000016e-07, is less than
options.OptimalityTolerance = 1.000000e-06, and the relative maximum
constraint violation, 0.000000e+00, is less than
options.ConstraintTolerance = 1.000000e-06.'

lambda =
         eqlin: [0×1 double]
      eqnonlin: [0×1 double]
       ineqlin: [0×1 double]
         lower: [2×1 double]
         upper: [2×1 double]
    ineqnonlin: [2×1 double]

grad =
   -0.0015
    0.0579

hessian =
    0.1464    0.2038
    0.2038    0.3980
```

The function fmincon returns various outputs. First, the variable x contains optimum design variables, and the variable fval contains the value of optimum objective function. The values of constraints at the optimum design can be obtained by calling confun.m as [c, ceq] = confun(x). The meaning of variable exitflag = 1 is explained in Table 5-4, where the optimum is achieved by satisfying the optimality criterion. When the exitflag is 0, -1, or -2, the optimization iteration stops without convergence. The variable output contains various information regarding the optimization process, such as the total number of iterations and the number of function evaluations. The variable lambda contains the Lagrange multiplier for all constraints. The variables grad and hessian contain the gradient and Hessian information of the objective function at the optimum design.

Table 5-4: Interpretation of exitflag in Matlab optimization

exitflag	Meaning
1	First-order optimality measure was less than options.TolFun and maximum constraint violation was less than options.TolCon.
2	Change in x was less than options.TolX.
3	Change in the objective function value was less than options.TolFun.
4	The magnitude of the search direction was less than 2*options.TolX and constraint violation was less than options.TolCon.
5	Magnitude of directional derivative in search direction was less than 2*options.TolFun and maximum constraint violation was less than options.TolCon.
0	Number of iterations exceeded options.MaxIter or number of function evaluations exceeded options.FunEvals.
-1	The algorithm was terminated by the output function.
-2	No feasible point was found.

Table 5-5: Available optimset options

Option	Data	Meaning
Display	'off' 'iter' 'final' 'notify'	Level of display
GradObj	'on' 'off'	Objective gradient
Jacobian	'on' 'off'	Constraint gradient
LargeScale	'on' 'off'	Algorithm
MaxFunEvals	Integer	Max. number of function evaluations
MaxIter	Integer	Max. number of optimization iterations
TolFun	Real	Convergence tolerance for objective
TolX	Real	Convergence tolerance for design variables

Example 5-11

We will examine the behavior of the function fmincon for a simple example of minimizing a quadratic function in a ring. Solve the following constrained optimization problem when (a) $a = 10$ and (b) $a = 1.1$. Compare the difference between the two cases in terms of

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) = x_1^2 + ax_2^2 \\ &\text{subject to } 10^2 \le x_1^2 + x_2^2 \le 20^2 \end{aligned}$$

Solution:

(a) Since the objective function is an elliptical shape, the outer ring constraint will not be active, but it may slow down convergence by limiting the straight-line moves. As long as $a > 1$, the optimum on the inner circle will go to $x_2 = 0$. For the first case, we choose $a = 10$, which would make it easy for fmincon to find the optimum, even if we start far away at $\mathbf{x}_0 = (1,10)$. The following two Matlab functions, quad2.m and ring.m, are defined for the objective and constraint functions, respectively:

```
function f=quad2(x)
  global a
  f=x(1)^2+a*x(2)^2;
end
%
function [c,ceq]=ring(x)
  c(1)=10^2-x(1)^2-x(2)^2;
  c(2)=x(1)^2+x(2)^2-20^2;
  ceq=[];
end
```

It is noted that the inequality constraint is split into two inequality constraints in order to use the standard form of constraints. The following Matlab script is used to call the fmincon function:

```
global a
x0=[1,10];a=10;
[x,fval,exitflag,output,lambda]=fmincon(@quad2,x0,[],[],[],[],[],[],@ring)
```

The output from fmincon tells us that it satisfied convergence criteria based on the lack of progress on the objective function and constraint satisfaction. In both cases, this is based on tolerances that we can change with the optimset function, which uses default values.

```
x =   10.0000    0.0000
fval =  100.0000
exitflag =     1
output =
         iterations: 9
```

```
      funcCount: 36
 constrviolation: 0
       stepsize: 9.9662e-06
      algorithm: 'interior-point'
  firstorderopt: 2.0004e-06
   cgiterations: 2
```

With the calling sequence we had, it gives us the objective function value of 100 and the optimum $\mathbf{x} = (10,0)$. The optimization finished successfully (exitflag=1), and that it took 9 iterations and 36 function evaluations. This is remarkable since we did not provide fmincon with a routine to calculate derivatives of objective function and constraints and it had to calculate them by finite differences. So with 9 iterations, 18 of the function evaluations were used to calculate derivatives. This means that the average number of function evaluations per one dimensional search was only 2.

(b) Setting $a = 1.1$, we make the advantage of reducing x_2 and increasing x_1 much smaller. This causes fmincon to slow down a lot. It took 36 iterations with 216 function evaluations. The optimization converges to the same optimum design as with case (a). Note that each iteration now takes about 4 function evaluations!

```
x =   10.0000    0.0000
fval =  100.0000
exitflag =     1
output =
         iterations: 36
          funcCount: 216
    constrviolation: 0
           stepsize: 9.9694e-06
          algorithm: 'interior-point'
      firstorderopt: 2.0002e-06
       cgiterations: 19
```

5.8. Practical suggestions for numerical optimization

Although various numerical optimization algorithms are presented in this chapter, there are several things that the users need to pay attention in order to make the optimization solution process robust. These are applicable to all algorithms in this chapter.

1. It is always good to normalize the design space to $0 \le x_k \le 1$. Mathematically, it is possible to solve design variables with different lower- and upper-bounds. However, when the ranges of design variables are significantly different, a numerical difficulty can occur. For example, let us consider the case that both Young's modulus and Poisson's ratio are design variables. In the MKS unit system, Young's modulus of metal is in the order of 10^{11} Pa, while the Poisson's ratio is $0 < \nu < 0.5$. Such a huge difference makes it difficult for numerical algorithms to find the optimum design for both variables. Therefore, it would be better to normalize all design variables with the same range:

$$\hat{x} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{5.49}$$

where $x_{\min}$ and $x_{\max}$ are, respectively, the lower- and upper-bounds of the design variable. In many commercial optimization algorithms internally normalize design variables. When a design variable does

not have lower- and/or upper-bounds, it is better to select very low or high values for the bounds.

2. Similar to the normalization of design variables, it is also better to normalize constraints. This is because different constraints have different scales. For example, in solid mechanics, displacement constraints are in the order of 10^{-5}m, while stress constraint is in the order of 10^{8}Pa. Such a huge difference in scale can cause difficulty in satisfying the constraints. A violation of 10^{3} is ignorable for stress constraint, but it is a huge violation of the displacement constraint. Therefore, it would be necessary to scale all constraints such that their violation and margin can be interpreted in the same way. For example, when a constraint is given in the form of $g(\mathbf{x}) \leq 0$, it would be desirable to scale them such that $g = -0.1$ means a 10 percent margin, and $g = +0.1$ corresponds to a 10 percent violation. When a constraint is given in the form of $c(x) \leq c_{\text{limit}}$, the normalized constraint can be written as

$$g(\mathbf{x}) = \frac{c(\mathbf{x})}{c_{\text{limit}}} - 1 \leq 0 \tag{5.50}$$

Similar normalization can be possible for greater-than-equal-to type constraints. When a constrain bound is zero, an appropriate scale should be chosen by the users.

3. When the users have many algorithms to choose, it is better to choose an optimization algorithm that computes Lagrange multipliers. In Chapter 4, the optimality condition of a constrained optimization problem is formulated using Lagrange multipliers. These Lagrange multipliers are not only important to find an optimum design but also useful to understand the role of constraints on the optimum design. If a Lagrange multiplier is zero at the optimum design, the corresponding constraint is inactive. If a Lagrange multiplier at the optimum design is small, the influence of the corresponding constraint is not significant. On the other hand, if a Lagrange multiplier is high, the constraint is important. That is, if the constraint bound is changed slightly, the optimum design can change significantly. Therefore, it is a good practice to check the Lagrange multipliers at the optimum design to understand what constraints play an important role to determining the optimum design.

4. Most optimization algorithms in this chapter require gradient (i.e., derivative) information of the objective and constraint functions, which is why they are categorized as gradient-based optimization. There are systematic ways of calculating the gradient information [*,*], which is called design sensitivity analysis. They require additional computer program that depends on the objective and constraint functions. In many cases, design sensitivity analysis can calculate the gradient information with much cheaper computational costs. However, since it requires significant modification of the response analysis program, not all computer programs have the capability of calculating the gradient information. Instead, the gradient information is often calculated using a finite difference method, where the design variable is slightly perturbed, and the response is calculated again. Then, using the difference between the two response values, the gradient is approximated, as

$$\nabla g(\mathbf{x}) \approx \frac{g(\mathbf{x} + \Delta\mathbf{x}) - g(\mathbf{x})}{\Delta\mathbf{x}} \tag{5.51}$$

Indeed, this is the definition of the derivative as $\Delta\mathbf{x} \to 0$. The above method of calculating gradient is referred to as the forward finite difference method because the design is perturbed in the positive direction of the design variable. Gradient calculation using a finite difference method is computationally expensive because each design variable is perturbed one at a time and the response is calculated again. For example, if there are n design variables, it requires $n + 1$ response analyses to calculate the gradient information. In fact, gradient calculation is a major part of the computational cost in optimization. However, the forward

finite difference in Eq. (5.51) can cause biased gradient information as the design is perturbed in the positive direction. If the computational cost is affordable, it is recommended to use the central finite difference method, where the sensitivity is calculated by

$$\nabla g(\mathbf{x}) \approx \frac{g(\mathbf{x}+\Delta\mathbf{x}) - g(\mathbf{x}-\Delta\mathbf{x})}{2\Delta\mathbf{x}} \tag{5.52}$$

The central finite difference method is expensive as it requires perturbing the design in both positive and negative directions. When there are n design variables, the central finite difference method requires $2n + 1$ response analyses to calculate the gradient information. However, the calculated gradient is unbiased.

5. Finite element analysis programs are often used to calculate the objective and constraint functions. When the size of model is large, it is difficult to solve a large matrix equation using a direct method. Instead, iterative methods are often used to solve a large system of equations. The iterative methods often yield less accurate solutions than the direct method. Therefore, if iterative analysis methods are used for the analysis, it would be better to use an extra stringent convergence criterion during the optimization if you can afford to. Otherwise, you can get poor finite-difference derivatives and bogus local minima.

6. Optimization algorithms assume that the analysis can produce results anywhere in the design space. In practice, however, it is possible that the analysis may not be successful at some portion of the design space. Since the analysis cannot produce results at this region, the optimization algorithm may think that the objective function value is low in this region, which is wrong. Therefore, it would be necessary to carefully check the final designs with more accurate analysis codes or more refined models.

7. It would not be smart to solve an optimization problem without having any prior estimation or expectation of the optimum design. It is dangerous to use the optimization program as if it is a black-box. Before using optimization, it is crucial to gain some insight into the problem by studying carefully the results of an analysis of a representative or intuitive design.

8. It would be a bad choice if nonlinear optimization algorithms are used to solve linear problems. It is true that nonlinear algorithms can solve linear problems. However, nonlinear optimizers are not as robust as linear optimizers, and have more difficulty in calculating Lagrange multipliers. Therefore, if both the objective function and constraints are linear, it is always better to use linear optimization algorithms.

9. When optimization algorithms stop convergence iterations, it does not always mean that the optimum design has been found. There are many other reasons that the optimization algorithms stop without converging to the optimum design. Optimization algorithms can stop iterations for many reasons. For example, optimization algorithms stop when the maximum number of function evaluations is reached. Also, they stop when the maximum number of iterations is reached. The best way to confirm local optima is to check the Kuhn-Tucker conditions. In practice, this might not be an easy task. In the case of Matlab optimization algorithms, the reason of stopping the algorithms is stored in EXITFLAG variable. When EXITFLAG is negative, it means that the optimization algorithms stopped without finding local optima.

In presenting optimization results, the guiding principle is one that applies to almost any document that is long enough, so some readers may not want to read everything. The document should provide summaries that allow the reader to get the important information in the document without reading everything and indicate where in the rest of the document additional details on each topic may be found. When presenting the solution to an optimization problem, the following information should be included:

1. Summary of design variables, objective function, and constraints
2. Method of solution
3. Solution, including minimizer, the value of the objective function, the indication of which constraints are critical, and if they are not exactly zero, what is the margin or degree of violation
4. List of the other materials included with the solution which allow the reader to get additional details.

5.9. Exercise

1. Explain the differences and commonalities of steepest descent, conjugate gradients, Newton's method, and quasi-Newton methods for unconstrained minimization

2. Minimize $f(x_1, x_2) = 12x_1^2 + 4x_2^2 - 12x_1x_2 + 2x_1$ using the conjugate gradient method, starting with the initial design $\mathbf{x}^{(0)} = \{-1, -2\}$.

3. Calculate the order and rate of convergence of the sequence $x^{(k)} = 1/k$ and $x^* = 0$.

4. With an extremely robust algorithm, we can find a very accurate solution with a penalty function approach by using a very high ω. However, at some high value the algorithm will begin to falter, either taking very large number of iterations or not reaching the solution. Test fminunc and fminsearch on **Example 5-9** starting from $\mathbf{x}_0 = [2,2]$. Start with $\omega = 1000$ and increase.

5. Show that the KKT condition of the quadratic programming problem in Eq. (5.47) is equivalent to the KKT condition in Eq. (5.46).

   Solution:

---

6. Use the Matlab fminsearch function to minimize the function $f = x_1^2 + (x_1 - x_2)^2$ starting at the point $(10,1)$. Report the first 23 function evaluations and identify each with the Nelder-Mead operation that yields each point (i.e., reflection, contraction, expansion, etc.).

7. Use fminunc to minimize the Rosenbrock Banana function and compare the trajectories of fminsearch and fminunc starting from (-1.2,1), with and without the routine for calculating the gradient. Plot the three trajectories.

---

8. Minimize a quadratic objective function with a ring constraint using the Matlab `fmincon` function. Using the bounds $r_i = 10$ and $r_o = 20$. Is feasible domain convex?

$$\begin{aligned} &\underset{x_1,x_2}{\text{minimize}} \quad f(x_1, x_2) = x_1^2 + 10x_2^2 \\ &\text{subject to} \quad r_i^2 \le x_1^2 + x_2^2 \le r_o^2 \end{aligned}$$

9. For the ring problem in **Example 5-11** with $a = 10$, can you find a starting point within a circle of radius 30 around the origin that will prevent fmincon of finding the optimum?

---

10. Solve the problem of minimizing the surface area of the cylinder subject to a minimum volume constraint as an inequality constraint. Do also with Matlab by defining non-dimensional radius and height using the cubic root of the volume.

    Solution:

```
function Opt
x0=[1,1];
A=-eye(2); b=[0,0]';
options = optimset('Algorithm','active-set','Display','iter');
dopt_cov=fmincon(@(d) myfun(d),x0,A,b,[],[],[],[],@(d)
nonlcon(d),options)

function f=myfun(d)
r=d(1);
h=d(2);
f=2*pi()*r^2+2*pi()*r*h;

function [c,ceq]=nonlcon(d)
r=d(1);
h=d(2);
c=-r^2*h+1/pi();
ceq=[];
```

---

11. A 10-bar truss structure shown in the figure is under two loads, P_1 and P_2. The design goal is to minimize the weight, W, by varying the cross-sectional areas, A_i, of the truss members. The stress of the member should be less than the allowable stress with the safety factor. For manufacturing reasons, the cross-sectional areas should be greater than the minimum value. Input data are summarized in the table. Find optimum design using the fmincon Matlab function. Use initial cross-sectional area $A_i = 0.5\ \text{in}^2$. Plot objective history (objective value versus iteration). Discuss EXITFLAG. Discuss why some members have minimum area.

    In order to calculate the member forces, the following terms need to be defined first:

$$a_{11} = \left(\frac{1}{A_1} + \frac{1}{A_3} + \frac{1}{A_5} + \frac{2\sqrt{2}}{A_7} + \frac{2\sqrt{2}}{A_8}\right), a_{12} = a_{21} = \frac{1}{A_5}$$

$$a_{22} = \left(\frac{1}{A_2} + \frac{1}{A_3} + \frac{1}{A_5} + \frac{1}{A_6} + \frac{2\sqrt{2}}{A_9} + \frac{2\sqrt{2}}{A_{10}}\right)$$

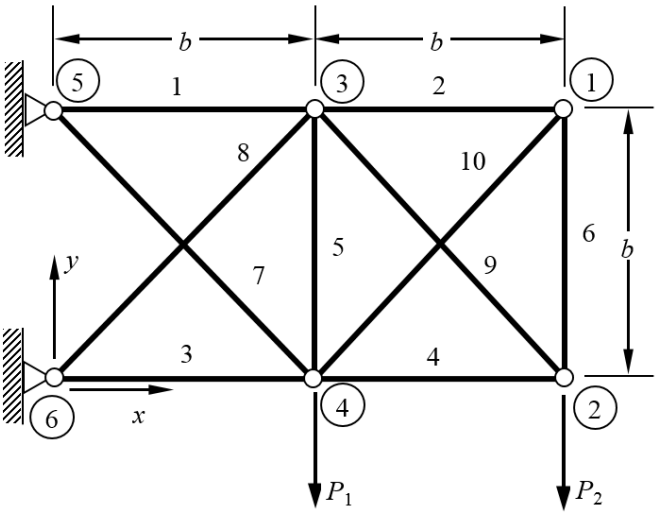
$$b_1 = \sqrt{2}\left(\frac{P_2}{A_1} - \frac{P_1 + 2P_2}{A_3} - \frac{P_2}{A_5} - \frac{2\sqrt{2}(P_1 + P_2)}{A_7}\right), b_2 = \left(-\frac{\sqrt{2}P_2}{A_4} - \frac{\sqrt{2}P_2}{A_5} - \frac{4P_2}{A_9}\right)$$

    Then, member forces are defined as

$$N_5 = -P_2 - \frac{1}{\sqrt{2}}N_8 - \frac{1}{\sqrt{2}}N_{10}, N_6 = -\frac{1}{\sqrt{2}}N_{10}$$
$$N_7 = \sqrt{2}(P_1 + P_2) + N_8, N_9 = \sqrt{2}P_2 + N_{10}$$

Parameters	Values
Dimension, b	360 inches
Safety factor, S_F	1.5
Load, P_1	66.67 kips
Load, P_2	66.67 kips
Density, ρ	0.1 lb/in^3
Modulus of elasticity, E	10^4 ksi
Allowable stress, $\sigma_{\text{allowable}}$	25 ksi*
Initial area A_i	1.0 in^2
Minimum cross-sectional area	0.1 in^2

*for Element 9, allowable stress is 75 ksi



12. .

13. .

14. .