Chapter 3

Gradient-Based Optimization

3.1 Introduction

In Chapter 2 we described methods to minimize (or at least decrease) a function of one variable. While problems with one variable do exist in MDO, most problems of interest involve multiple design variables. In this chapter we consider methods to solve such problems, restricting ourselves to smooth unconstrained problems. Later, we extend this to constrained problems in Chapter 5, and remove the smoothness requirement in Chapter 6.

The unconstrained optimization problem can be stated as,

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{with respect to} \quad & x \in \mathbb{R}^n \end{aligned} \tag{3.1}$$

where x is the n-vector $x = [x_1, x_2, \ldots, x_n]^T$. The objective function f can be nonlinear, but one important assumption in this chapter is that f is a sufficiently smooth function of x: in some cases we will demand that f has continuous first (partial) derviatives, and in others we require f to also have continuous second (partial) derivatives.

For large numbers of variables n, gradient-based methods are usually the most efficient algorithms. This class of methods uses the gradient of the objective function to determine the most promising directions along which we should search. And here is where the line search comes in: it provides a way to search for a better point along a line in n-dimensional space.

All algorithms for unconstrained gradient-based optimization can be described as shown in Algorithm 13. The outer loop represents the *major iterations*. The design variables are updated at each major iteration k using

$$x_{k+1} = x_k + \underbrace{\alpha_k p_k}_{\Delta x_k} \tag{3.2}$$

where p_k is the search direction for major iteration k, and α_k is the accepted step length from the line search. The line search usually involves multiple iterations that do not count towards the major iterations. This is an important distinction that needs to be considered when comparing the computational cost of the various approaches.

The two iterative loops represent the two subproblems in this type of algorithm: 1) The computation a search direction p_k, and; 2) The search for an acceptable step size (controlled by α_k). These two subproblems are illustrated in Fig. 3.1. The various types of gradient-based algorithms are classified based on the method that is used for computing the search direction (the first subproblem). Any line search that satisfies sufficient decrease can be used with a given gradient-based method.

Algorithm 4 General algorithm for smooth functions

```
Input: Initial guess, $x_0$
Output: Optimum, $x^*$
  $k \leftarrow 0$
  while Not converged do
    Compute a search direction $p_k$
    Find a step length $\alpha_k$, such that $f(x_k + \alpha_k p_k) < f(x_k)$ (the curvature condition may also be included)
    Update the design variables: $x_{k+1} \leftarrow x_k + \alpha_k p_k$
    $k \leftarrow k + 1$
  end while
```

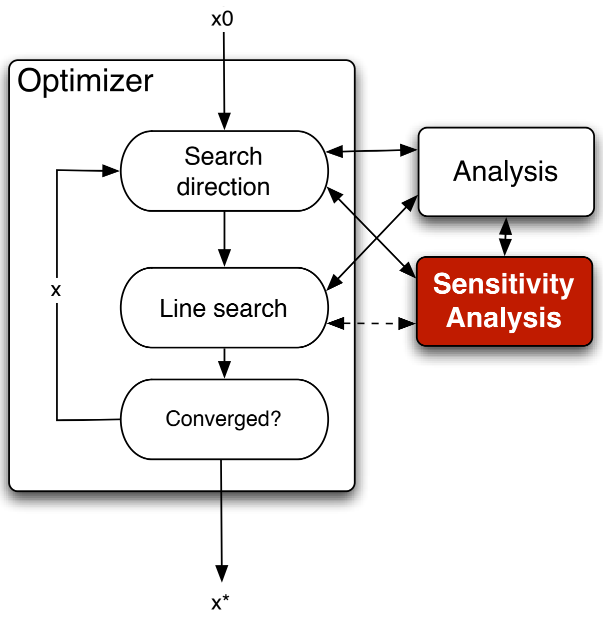


Figure 3.1: General gradient-based method and its relation to sensitivity analysis. “Analysis” is the evaluation of the objective function f, and “Sensitivity Analysis” the evaluation of ∇f

3.2 Gradients and Hessians

Consider a function $f(x)$. The *gradient* of this function is given by the vector of partial derivatives with respect to each of the independent variables,

$$\nabla f(x) \equiv g(x) \equiv \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{3.3}$$

In the multivariate case, the gradient vector is perpendicular to the the *hyperplane* tangent to the contour surfaces of constant f.

Higher derivatives of multi-variable functions are defined as in the single-variable case, but note that the number of gradient components increase by a factor of n for each differentiation.

While the gradient of a function of n variables is an n-vector, the "second derivative" of an n-variable function is defined by n^2 partial derivatives (the derivatives of the n first partial derivatives with respect to the n variables):

$$\frac{\partial^2 f}{\partial x_i \partial x_j}, \quad i \neq j \quad \text{and} \quad \frac{\partial^2 f}{\partial x_i^2}, \quad i = j. \tag{3.4}$$

If the partial derivatives $\partial f/\partial x_i$, $\partial f/\partial x_j$ and $\partial^2 f/\partial x_i \partial x_j$ are continuous and f is single valued, $\partial^2 f/\partial x_i \partial x_j = \partial^2 f/\partial x_j \partial x_i$. Therefore the second-order partial derivatives can be represented by a square symmetric matrix called the *Hessian matrix*,

$$\nabla^2 f(x) \equiv H(x) \equiv \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial^2 x_n}, \end{bmatrix} \tag{3.5}$$

which contains $n(n+1)/2$ independent elements.

If f is quadratic, the Hessian of f is constant, and the function can be expressed as

$$f(x) = \frac{1}{2} x^T H x + g^T x + \alpha. \tag{3.6}$$

3.3 Optimality Conditions

As in the single-variable case, the optimality conditions can be derived from the Taylor-series expansion of f about x^*:

$$f(x^* + \varepsilon p) = f(x^*) + \varepsilon p^T g(x^*) + \frac{1}{2} \varepsilon^2 p^T H(x^* + \varepsilon \theta p) p, \tag{3.7}$$

where $\theta \in (0, 1)$, ε is a scalar, and p is an n-vector.

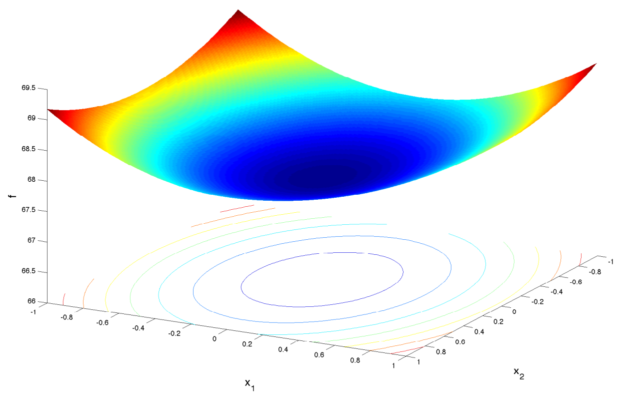
For x^* to be a local minimum, then for any vector p there must be a finite ε such that $f(x^*+\varepsilon p) \geq f(x^*)$, i.e. there is a neighborhood in which this condition holds. If this condition is satisfied, then $f(x^* + \varepsilon p) - f(x^*) \geq 0$ and the first and second order terms in the Taylor-series expansion must be greater than or equal to zero.

As in the single variable case, and for the same reason, we start by considering the first order terms. Since p is an arbitrary vector and ε can be positive or negative, every component of the gradient vector $g(x^*)$ must be zero.

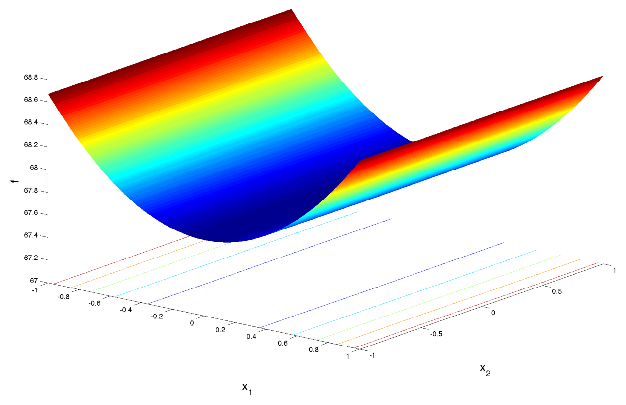
Now we have to consider the second order term, $\varepsilon^2 p^T H(x^* + \varepsilon \theta p) p$. For this term to be non-negative, $H(x^* + \varepsilon \theta p)$ has to be positive semi-definite, and by continuity, the Hessian at the optimum, $H(x^*)$ must also be positive semi-definite.

- The matrix $H \in \mathbb{R}^{n \times n}$ is *positive definite* if $p^T H p > 0$ for all nonzero vectors $p \in \mathbb{R}^n$. If $H = H^T$ then all the eigenvalues of H are strictly positive.

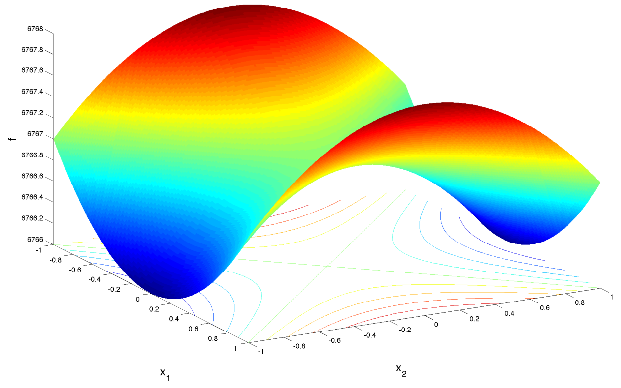
- The matrix $H \in \mathbb{R}^{n \times n}$ is *positive semi-definite* if $p^T H p \geq 0$ for all vectors $p \in \mathbb{R}^n$. If $H = H^T$ then the eigenvalues of H are positive or zero.



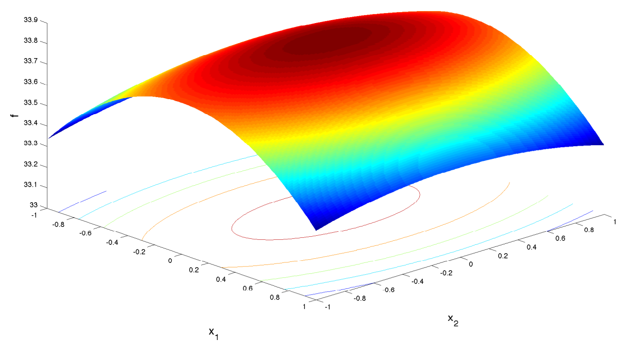
(a) Positive definite



(b) Positive semi-definite



(c) Indefinite



(d) Negative definite

Figure 3.2: Various possibilities for a quadratic function

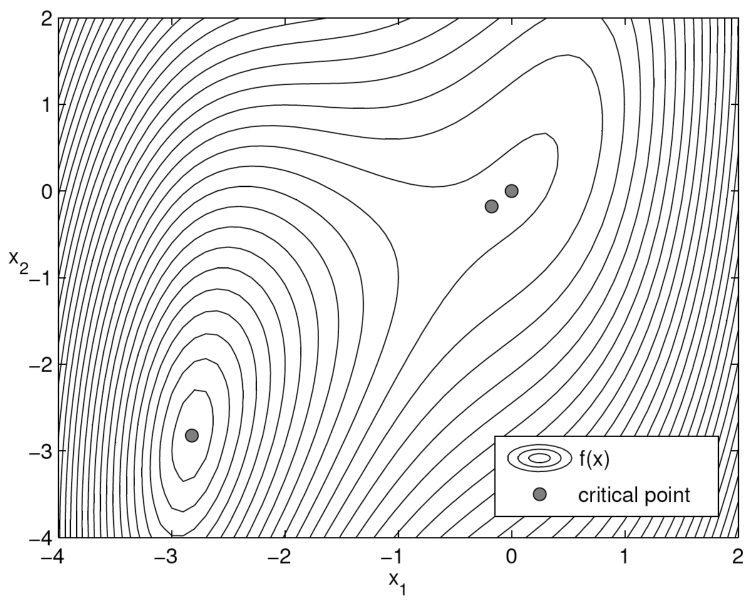


Figure 3.3: Critical points of $f(x) = 1.5x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + 0.5x_1^4$

- The matrix $H \in \mathbb{R}^{n\times n}$ is *indefinite* if there exists $p, q \in \mathbb{R}^n$ such that $p^T Hp > 0$ and $q^T Hq < 0$. If $H = H^T$ then H has eigenvalues of mixed sign.
- The matrix $H \in \mathbb{R}^{n\times n}$ is *negative definite* if $p^T Hp < 0$ for all nonzero vectors $p \in \mathbb{R}^n$. If $H = H^T$ then all the eigenvalues of H are strictly negative

Necessary conditions (for a local minimum):

$$\|g(x^*)\| = 0 \quad \text{and} \quad H(x^*) \quad \text{is positive semi-definite.} \tag{3.8}$$

Sufficient conditions (for a strong local minimum):

$$\|g(x^*)\| = 0 \quad \text{and} \quad H(x^*) \quad \text{is positive definite.} \tag{3.9}$$

Example 3.4. Critical Points of a Function Consider the function:

$$f(x) = 1.5x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + 0.5x_1^4$$

Find all stationary points of f and classify them.

Solve $\nabla f(x) = 0$, get three solutions:

$$\begin{array}{rl} (0,0) & \text{local minimum} \\ 1/2(-3-\sqrt{7}, -3-\sqrt{7}) & \text{global minimum} \\ 1/2(-3+\sqrt{7}, -3+\sqrt{7}) & \text{saddle point} \end{array}$$

To establish the type of point, we have to determine if the Hessian is positive definite and compare the values of the function at the points.

3.4 Steepest Descent Method

The steepest descent method uses the gradient vector at x_k as the search direction for the major iteration k. As mentioned previously, the gradient vector is orthogonal to the plane tangent to the isosurfaces of the function. The gradient vector, $g(x_k)$, is also the direction of maximum rate of change (maximum increase) of the function at that point. This rate of change is given by the norm, $\|g(x_k)\|$.

Algorithm 5 Steepest descent

```
Input: Initial guess, $x_0$, convergence tolerances, $\varepsilon_g, \varepsilon_a$ and $\varepsilon_r$.
Output: Optimum, $x^*$
  $k \leftarrow 0$
  repeat
    Compute the gradient of the objective function, $g(x_k) \equiv \nabla f(x_k)$
    Compute the normalized search direction, $p_k \leftarrow -g(x_k)/\|g(x_k)\|$
    Perform line search to find step length $\alpha_k$
    Update the current point, $x_{k+1} \leftarrow x_k + \alpha_k p_k$
    $k \leftarrow k+1$
  until $|f(x_k) - f(x_{k-1})| \leq \varepsilon_a + \varepsilon_r |f(x_{k-1})|$ and $\|g(x_{k-1})\| \leq \varepsilon_g$
```

Here, $|f(x_{k+1}) - f(x_k)| \leq \varepsilon_a + \varepsilon_r |f(x_k)|$ is a check for the successive reductions of f. ε_a is the absolute tolerance on the change in function value (usually small $\approx 10^{-6}$) and ε_r is the relative tolerance (usually set to 0.01).

If we use an exact line search, the steepest descent direction at each iteration is orthogonal to the previous one, i.e.,

$$\begin{aligned}
\frac{\mathrm{d}f(x_{k+1})}{\mathrm{d}\alpha} &= 0 \\
\Rightarrow \quad \frac{\partial f(x_{k+1})}{\partial x_{k+1}} \frac{\partial \left(x_k + \alpha p_k\right)}{\partial \alpha} &= 0 \\
\Rightarrow \quad \nabla^T f(x_{k+1}) p_k &= 0 \\
\Rightarrow -g^T(x_{k+1}) g(x_k) &= 0
\end{aligned}$$

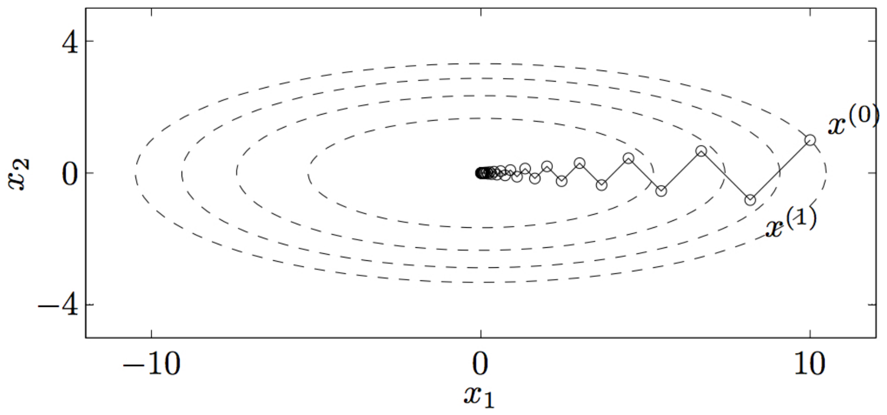
Because of this property of the search directions, the steepest descent method "zigzags" in the design space and is inefficient, in general. Although a substantial decrease may be observed in the first few iterations, the method is usually very slow to converge to a local optimum. In particular, while the algorithm is guaranteed to converge, it may take an infinite number of iterations. The rate of convergence is linear.

Consider, for example the quadratic function of two variables,

$$f(x) = (1/2)(x_1^2 + 10x_2^2)$$

.

The following figure shows the iterations given by steepest descent when started at $x_0 = (10, 1)$.



For steepest descent and other gradient methods that do not produce well-scaled search directions, we need to use other information to guess a step length.

One strategy is to assume that the first-order change in x_k will be the same as the one obtained in the previous step. i.e, that $\bar{\alpha} g_k^T p_k = \alpha_{k-1} g_{k-1}^T p_{k-1}$ and therefore:

$$\bar{\alpha} = \alpha_{k-1} \frac{g_{k-1}^T p_{k-1}}{g_k^T p_k}. \tag{3.10}$$

Example 3.5. Steepest Descent:
Consider the following function.

$$f(x_1, x_2) = 1 - e^{-(10x_1^2 + x_2^2)}. \tag{3.11}$$

The function f is not quadratic, but, as $|x_1|$ and $|x_2| \to 0$, we see that

$$f(x_1, x_2 = 10x_1^2 + x_2^2 + \mathrm{O}(x_1^4) + \mathrm{O}(x_2^4)$$

Thus, this function is essentially a quadratic near the minimum $(0, 0)^T$. Figure 3.4 shows the path taken by steepest descent starting from the initial point, $(-0.1, 0.6)^T$.

3.5 Conjugate Gradient Method

A small modification to the steepest descent method takes into account the history of the gradients to move more directly towards the optimum.

Suppose we want to minimize a convex quadratic function

$$\phi(x) = \frac{1}{2} x^T A x - b^T x \tag{3.12}$$

where A is an $n \times n$ matrix that is symmetric and positive definite. Differentiating this with respect to x we obtain,

$$\nabla \phi(x) = Ax - b \equiv r(x). \tag{3.13}$$

Minimizing the quadratic is thus equivalent to solving the linear system,

$$\nabla \phi = 0 \Rightarrow Ax = b. \tag{3.14}$$

The conjugate gradient method is an iterative method for solving linear systems of equations such as this one.

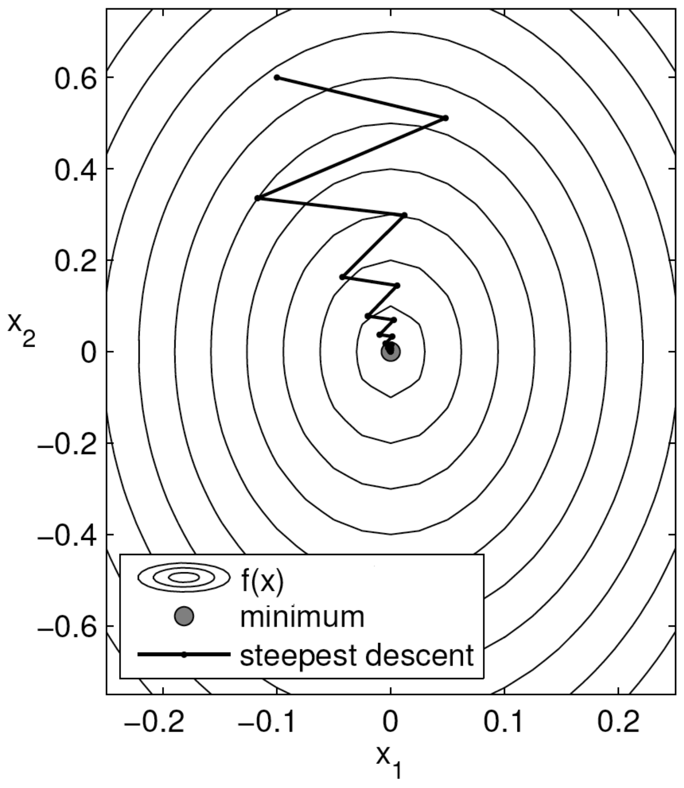


Figure 3.4: Solution path of the steepest descent method

A set of nonzero vectors $\{p_0, p_1, \dots, p_{n-1}\}$ is *conjugate* with respect to A if

$$p_i^T A p_j = 0, \quad \text{for all} \quad i \neq j. \tag{3.15}$$

Suppose that we start from a point x_0 and a set of conjugate directions $\{p_0, p_1, \dots, p_{n-1}\}$ to generate a sequence $\{x_k\}$ where

$$x_{k+1} = x_k + \alpha_k p_k \tag{3.16}$$

where α_k is the minimizer of ϕ along $x_k + \alpha p_k$, given by

$$\begin{aligned}\frac{d\phi(x_k + \alpha p_k)}{d\alpha} = \frac{d}{d\alpha}\left[\frac{1}{2}(x_k + \alpha p_k)^T A (x_k + \alpha p_k) - b^T(x_k + \alpha p_k)\right] &= 0 \\ \Rightarrow \quad p_k^T A(x_k + \alpha p_k) - p_k^T b &= 0 \\ \Rightarrow p_k^T \underbrace{(Ax_k - b)}_{r_k} + \alpha p_k^T A p_k &= 0\end{aligned}$$

Rearranging the last equation we find

$$\alpha_k \equiv -\frac{r_k^T p_k}{p_k^T A p_k} \tag{3.17}$$

We will see that for any x_0 the sequence $\{x_k\}$ generated by the conjugate direction algorithm converges to the solution of the linear system in at most n steps.

The conjugate directions are linearly independent, so they span n-space. Therefore, we can expand the vector $x^* - x_0$ using the p_k as a basis (recall that x^* is the solution).

$$x^* - x_0 = \sigma_0 p_0 + \cdots + \sigma_{n-1} p_{n-1} \tag{3.18}$$

Premultiplying by $p_k^T A$ and using the conjugacy property we obtain

$$\sigma_k = \frac{p_k^T A(x^* - x_0)}{p_k^T A p_k} \tag{3.19}$$

Now we will show that the σ's are really the α's defined in (3.17). A given iteration point can be written as a function of the search directions and step sizes so far,

$$x_k = x_0 + \alpha_0 p_0 + \cdots + \alpha_{k-1} p_{k-1}. \tag{3.20}$$

Premultiplying by $p_k^T A$ and using the conjugacy property we obtain

$$p_k^T A(x_k - x_0) = 0. \tag{3.21}$$

Therefore

$$\begin{aligned} p_k^T A(x^* - x_0) &= p_k^T A(x^* - x_k + x_k - x_0) \\ &= p_k^T A(x^* - x_k) + \underbrace{p_k^T A(x_k - x_0)}_{=0} \\ &= p_k^T (\underbrace{Ax^*}_{=b} - Ax_k) \\ &= p_k^T \underbrace{(b - Ax_k)}_{=-r_k} \\ &= -p_k^T r_k \end{aligned}$$

Dividing the leftmost term and the rightmost term by $p_k^T A p_k$ we get the equality,

$$\frac{p_k^T A(x^* - x_0)}{p_k^T A p_k} = -\frac{p_k^T r_k}{p_k^T A p_k} \Rightarrow \sigma_k = \alpha_k. \tag{3.22}$$

In light of this relationship and (3.18), if we step along the conjugate directions using α_k, we will solve the linear system $Ax = b$ in at most n iterations.

There is a simple interpretation of the conjugate directions. If A is diagonal, the isosurfaces are ellipsoids with axes aligned with coordinate directions. We could then find minimum by performing univariate minimization along each coordinate direction in turn and this would result in convergence to the minimum in n iterations.

When A a positive-definite matrix that is not diagonal, its contours are still elliptical, but they are not aligned with the coordinate axes. Minimization along coordinate directions no longer leads to solution in n iterations (or even a finite n). Thus, we need a different set of search directions to recover convergence in n iterations: this is the role of the conjugate directions.

The original variables x are related to the new ones by $x = S\hat{x}$. Inverting this transformation,

$$\hat{x} = S^{-1} x, \tag{3.23}$$

where $S = \begin{bmatrix} p_0 & p_1 & \cdots & p_{n-1} \end{bmatrix}$, a matrix whose columns are the set of conjugate directions with respect to A. The quadratic now becomes

$$\hat{\phi}(\hat{x}) = \frac{1}{2}\hat{x}^T \left(S^T A S\right) \hat{x} - \left(S^T b\right)^T \hat{x} \tag{3.24}$$

By conjugacy, $\left(S^T A S\right)$ is diagonal so we can do a sequence of n line minimizations along the coordinate directions of $\hat{x}$. Each univariate minimization determines a component of x^* correctly.

When the conjugate-gradient method is adapted to general nonlinear problems, we obtain the *Fletcher–Reeves method.*

Algorithm 6 Nonlinear conjugate gradient method

```
Input: Initial guess, $x_0$, convergence tolerances, $\varepsilon_g, \varepsilon_a$ and $\varepsilon_r$.
Output: Optimum, $x^*$
  $k \leftarrow 0$
  repeat
    Compute the gradient of the objective function, $g(x_k)$
    if k=0 then
      Compute the normalized steepest descent direction, $p_k \leftarrow -g(x_k)/\|g(x_k)\|$
    else
      Compute $\beta \leftarrow \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$
      Compute the conjugate gradient direction $p_k = -g_k/\|g(x_k)\| + \beta_k p_{k-1}$
    end if
    Perform line search to find step length $\alpha_k$
    Update the current point, $x_{k+1} \leftarrow x_k + \alpha_k p_k$
    $k \leftarrow k + 1$
  until $|f(x_k) - f(x_{k-1})| \leq \varepsilon_a + \varepsilon_r |f(x_{k-1})|$ and $\|g(x_{k-1})\| \leq \varepsilon_g$
```

Usually, a *restart* is performed every n iterations for computational stability, i.e. we start with a steepest descent direction.

The conjugate gradient method does not produce well-scaled search directions, so we can use same strategy to choose the initial step size as for steepest descent.

Several variants of the Fletcher–Reeves CG method have been proposed. Most of these variants differ in their definition of β_k. For example, Dai and Yuan [2] propose

$$\beta_k = \frac{\|g_k\|^2}{(g_k - g_{k-1})^T p_{k-1}}. \tag{3.25}$$

Another variant is the Polak–Ribière formula

$$\beta_k = \frac{g_k^T \left(g_k - g_{k-1}\right)}{g_{k-1}^T g_{k-1}}. \tag{3.26}$$

The only difference relative to the steepest descent is that the each descent direction is modified by adding a contribution from the previous direction.

The convergence rate of the nonlinear conjugate gradient is linear, but can be superlinear, converging in n to $5n$ iterations.

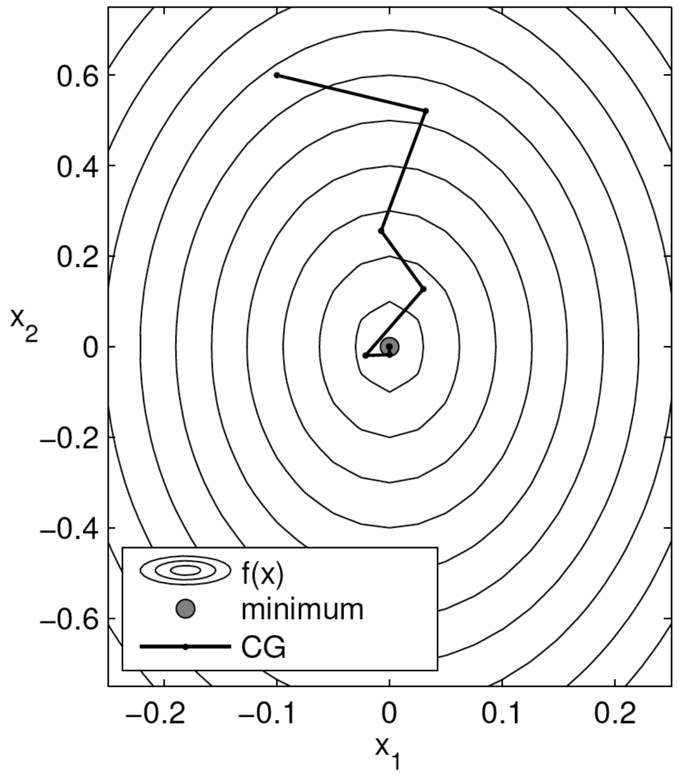


Figure 3.5: Solution path of the nonlinear conjugate gradient method.

Since this method is just a minor modification away from steepest descent and performs much better, there is no excuse for steepest descent!

Example 3.6. Conjugate Gradient Method:
Figure 3.5 plots the solution path of the nonlinear conjugate gradient method applied to the objective function f given by (3.11). The qualitative improvement over the steepest-descent path (Figure 3.4) is evident.

3.6 Newton's Method

The steepest descent and conjugate gradient methods only use first order information (the first derivative term in the Taylor series) to obtain a local model of the function.

Newton methods use a second-order Taylor series expansion of the function about the current design point, i.e. a quadratic model

$$f(x_k + s_k) \approx f_k + g_k^T s_k + \frac{1}{2} s_k^T H_k s_k, \tag{3.27}$$

where s_k is the step to the minimum. Differentiating this with respect to s_k and setting it to zero, we can obtain the step for that minimizes this quadratic,

$$H_k s_k = -g_k. \tag{3.28}$$

This is a linear system which yields the *Newton step*, s_k, as a solution.

If f is a quadratic function and H_k is positive definite, Newton's method requires only one iteration to converge from any starting point. For a general nonlinear function, Newton's method converges quadratically if x_0 is sufficiently close to x^* and the Hessian is positive definite at x^*.

As in the single variable case, difficulties and even failure may occur when the quadratic model is a poor approximation of f far from the current point. If H_k is not positive definite, the quadratic model might not have a minimum or even a stationary point. For some nonlinear functions, the Newton step might be such that $f(x_k + s_k) > f(x_k)$ and the method is not guaranteed to converge.

Another disadvantage of Newton's method is the need to compute not only the gradient, but also the Hessian, which contains $n(n+1)/2$ second order derivatives.

A small modification to Newton's method is to perform a line search along the Newton direction, rather than accepting the step size that would minimize the quadratic model.

Algorithm 7 Modified Newton's method

```
Input: Initial guess, $x_0$, convergence tolerances, $\varepsilon_g, \varepsilon_a$ and $\varepsilon_r$.
Output: Optimum, $x^*$
  $k \leftarrow 0$
  repeat
    Compute the gradient of the objective function, $g(x_k)$
    Compute the Hessian of the objective function, $H(x_k)$
    Compute the search direction, $p_k = -H^{-1} g_k$
    Perform line search to find step length $\alpha_k$, starting with $\alpha = 1$
    Update the current point, $x_{k+1} \leftarrow x_k + \alpha_k p_k$
    $k \leftarrow k + 1$
  until $|f(x_k) - f(x_{k-1})| \leq \varepsilon_a + \varepsilon_r |f(x_{k-1})|$ and $\|g(x_{k-1})\| \leq \varepsilon_g$
```

Although this modification increases the probability that $f(x_k + p_k) < f(x_k)$, it is still vulnerable to the problem of having an Hessian that is not positive definite and has all the other disadvantages of the pure Newton's method.

We could also introduce a modification to use a symmetric positive definite matrix instead of the real Hessian to ensure descent:

$$B_k = H_k + \gamma I,$$

where γ is chosen such that all the eigenvalues of B_k are greater than a tolerance $\delta > 0$ (if δ is too close to zero, B_k might be ill-conditioned, which can lead to round of errors in the solution of the linear system).

When using Newton's method, or quasi-Newton methods described next, the starting step length $\bar{\alpha}$ is usually set to 1, since Newton's method already provides a good guess for the step size.

The step size reduction ratio (ρ in the backtracking line search) sometimes varies during the optimization process and is such that $0 < \rho < 1$. In practice ρ is not set to be too close to 0 or 1.

Example 3.7. Modified Newton's Method:
In this example, we apply Algorithm 7 to the function f given by (3.11) starting from $(-0.1, 0.6)^T$. The resulting optimization path is shown in Fig. 3.6, where we can see the rapid convergence of the method: indeed, the first step is indistinguishable from subsequent steps, since f is almost a quadratic. However, moving the initial guess slightly away from the origin leads to divergence, because the Newton search direction ceases to be a descent direction.

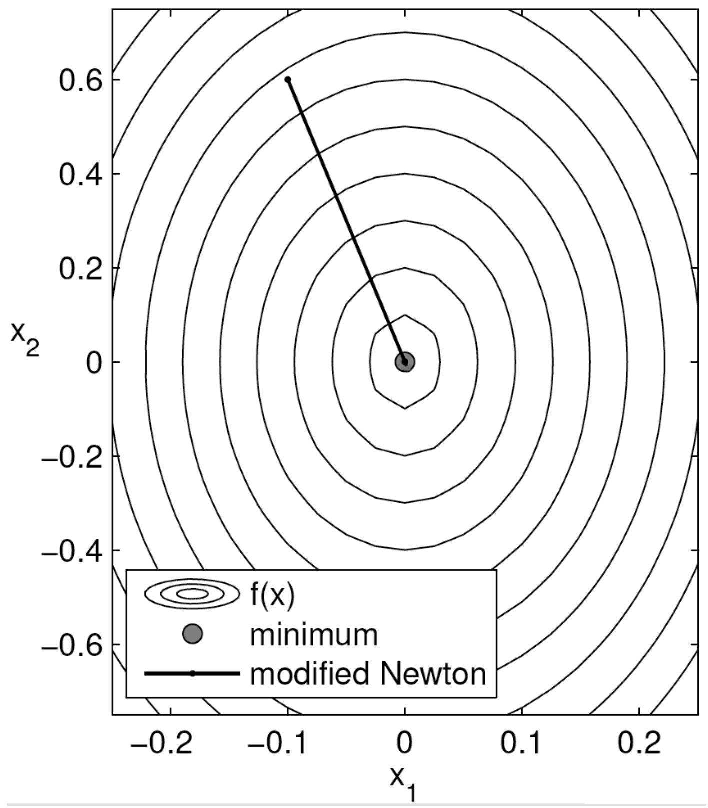


Figure 3.6: Solution path of the modified Newton's method.

3.7 Quasi-Newton Methods

This class of methods uses first order information only, but build second order information — an approximate Hessian — based on the sequence of function values and gradients from previous iterations. The various quasi-Newton methods differ in how they update the approximate Hessian. Most of these methods also force the Hessian to be symmetric and positive definite, which can greatly improve their convergence properties.

3.7.1 Davidon–Fletcher–Powell (DFP) Method

One of the first quasi-Newton methods was devised by Davidon in 1959, who a physicist at Argonne National Laboratories. He was using a coordinate descent method, and had limited computer resources, so he invented a more efficient method that resulted in the first quasi-Newton method.

This was one of the most revolutionary ideas in nonlinear optimization. Davidon's paper was not accepted for publication! It remained a technical report until 1991, when it was published as the first paper in the inaugural issue of the SIAM Journal on Optimization [3].

Fletcher and Powell later demonstrated modified the method and showed that it was much faster than current ones [5], and hence it became known as the Davidon–Fletcher–Powell (DFP) method.

Suppose we model the objective function as a quadratic at the current iterate x_k:

$$\phi_k(p) = f_k + g_k^T p + \frac{1}{2} p^T B_k p, \tag{3.29}$$

where B_k is an $n \times n$ symmetric positive definite matrix that is *updated* every iteration.

The step p_k that minimizes this convex quadratic model can be written as,

$$p_k = -B_k^{-1} g_k. \tag{3.30}$$

This solution is used to compute the search direction to obtain the new iterate

$$x_{k+1} = x_k + \alpha_k p_k \tag{3.31}$$

where α_k is obtained using a line search.

This is the same procedure as the Newton method, except that we use an approximate Hessian B_k instead of the true Hessian.

Instead of computing B_k "from scratch" at every iteration, a quasi-Newton method updates it in a way that accounts for the curvature measured during the most recent step. We want to build an updated quadratic model,

$$\phi_{k+1}(p) = f_{k+1} + g_{k+1}^T p + \frac{1}{2} p^T B_{k+1} p. \tag{3.32}$$

What requirements should we impose on B_{k+1}, based on the new information from the last step?

Using the secant method we can find the univariate quadratic function along the previous direction p_k based on the two last two gradients g_{k+1} and g_k, and the last function value f_{k+1}. The slope of the univariate function is the gradient of the function projected onto the p direction, i.e., $f' = g^T p$. The univariate quadratic is given by

$$\phi_{k+1}(\theta) = f_{k+1} + \theta f'_{k+1} + \frac{\theta^2}{2} \tilde{f}''_{k+1} \tag{3.33}$$

where $s = \alpha\|p\|$ and $\tilde{f}''_{k+1}$ is the approximation to the curvature of f, which is given by a forward finite difference on the slopes, i.e.,

$$\tilde{f}''_{k+1} = \frac{f'_{k+1} - f'_k}{\alpha_k \|p_k\|} \tag{3.34}$$

These slopes are easily obtained by projecting the respective gradients onto the last direction p_k. The result is a quadratic that matches f at the current point, since $\phi(0) = f_{k+1}$. The slope of this quadratic is given by

$$\phi'_{k+1}(\theta) = f'_{k+1} + \theta \tilde{f}''_{k+1} \tag{3.35}$$

At $\theta = 0$ this slope also matches that of f, since $\phi'_{k+1}(0) = f'_{k+1}$.

The previous point x_k corresponds to $\theta = -\alpha_k\|p_k\|$. Substituting the curvature approximation (3.34) into the slope of the quadratic (3.35), and evaluating it at x_k we obtain

$$\phi'_{k+1}(-\alpha_k\|p_k\|) = f'_k \tag{3.36}$$

Thus, the quadratic based on the secant method matches the slope and value at the current point, and the slope of the previous point. This is illustrated in Fig. 3.7.

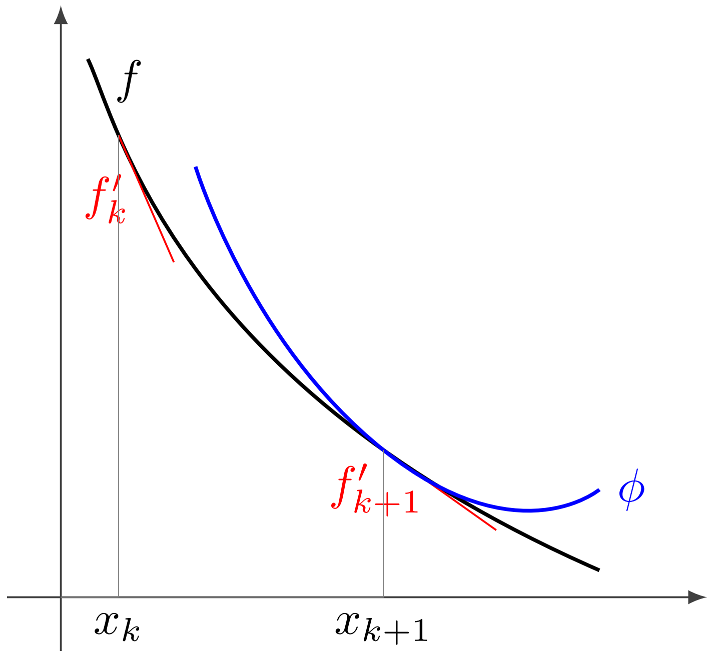


Figure 3.7: Projection of the quadratic model onto the last search direction, illustrating the secant condition

Going back to n-dimensional space, the gradient of the quadratic model (3.32) is

$$\nabla\phi_{k+1}(p) = g_{k+1} + B_{k+1}p. \tag{3.37}$$

For this gradient to match that of the actual function at the previous point x_k,

$$\nabla\phi_{k+1}\left(-\alpha_k p_k\right) = g_{k+1} - \alpha_k B_{k+1} p_k = g_k. \tag{3.38}$$

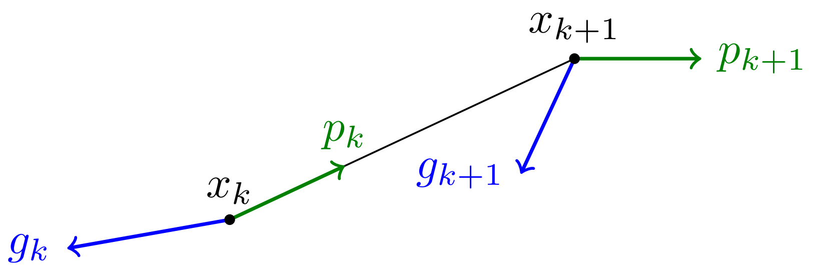


Figure 3.8: The difference in the gradients at two subsequent points g_k and g_{k+1} is projected onto the direction p_k to enforce the secant condition show in Fig. 3.7.

Rearranging we obtain,

$$B_{k+1}\alpha_k p_k = g_{k+1} - g_k. \tag{3.39}$$

which is called the *secant condition*.

For convenience, we will set the difference of the gradients to $y_k = g_{k+1} - g_k$, and $s_k = x_{k+1} - x_k$ so the secant condition is then written as

$$B_{k+1}s_k = y_k. \tag{3.40}$$

We have $n(n+1)/2$ unknowns and only n equations, so this system has an infinite number of solutions for B_{k+1}. To determine the solution uniquely, we impose a condition that among all the matrices that satisfy the secant condition, selects the B_{k+1} that is "closest" to the previous Hessian approximation B_k, i.e. we solve the following optimization problem,

$$\begin{aligned} \text{minimize} \quad & \|B - B_k\| \\ \text{with respect to} \quad & B \\ \text{subject to} \quad & B = B^T, \quad Bs_k = y_k. \end{aligned} \tag{3.41}$$

Using different matrix norms result in different quasi-Newton methods. One norm that makes it easy to solve this problem and possesses good numerical properties is the weighted Frobenius norm

$$\|A\|_W = \|W^{1/2}AW^{1/2}\|_F, \tag{3.42}$$

where the norm is defined as $\|C\|_F = \sum_{i=1}^n \sum_{j=1}^n c_{ij}^2$. The weights W are chosen to satisfy certain favorable conditions. The norm is adimensional (i.e., does not depend on the units of the problem) if the weights are chosen appropriately.

For further details on the derivation of the quasi-Newton methods, see Dennis and Moré's review [4].

Using this norm and weights, the unique solution of the norm minimization problem (3.41) is,

$$B_{k+1} = \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) B_k \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) + \frac{y_k y_k^T}{y_k^T s_k}, \tag{3.43}$$

which is the *DFP updating formula* originally proposed by Davidon.

Using the inverse of B_k is usually more useful, since the search direction can then be obtained by matrix multiplication. Defining,

$$V_k = B_k^{-1}. \tag{3.44}$$

The DFP update for the inverse of the Hessian approximation can be shown to be

$$V_{k+1} = V_k - \frac{V_k y_k y_k^T V_k}{y_k^T V_k y_k} + \frac{s_k s_k^T}{y_k^T s_k} \tag{3.45}$$

Note that this is a rank 2 update. A rank one update is $B_{k+1} = B_k + uv^T$, which adds a matrix whose columns are the same vector u multiplied by the scalars in v. this means that the columns are all linearly dependent and the matrix spans 1-space and is rank 1. A rank two update looks like $B_{k+1} = B_k + u_1 v_1^T + u_2 v_2^T$ and the update spans 2-space.

Algorithm 8 Quasi-Newton method with DFP update

```
Input: Initial guess, $x_0$, convergence tolerances, $\varepsilon_g, \varepsilon_a$ and $\varepsilon_r$.
Output: Optimum, $x^*$
  $k \leftarrow 0$
  $V_0 \leftarrow I$
  repeat
    Compute the gradient of the objective function, $g(x_k)$
    Compute the search direction, $p_k \leftarrow -V_k g_k$
    Perform line search to find step length $\alpha_k$, starting with $\alpha \leftarrow 1$
    Update the current point, $x_{k+1} \leftarrow x_k + \alpha_k p_k$
    Set the step length, $s_k \leftarrow \alpha_k p_k$
    Compute the change in the gradient, $y_k \leftarrow g_{k+1} - g_k$
    $A_k \leftarrow \frac{V_k y_k y_k^T V_k}{y_k^T V_k y_k}$
    $B_k \leftarrow \frac{s_k s_k^T}{s_k^T y_k}$
    Compute the updated approximation to the inverse of the Hessian, $V_{k+1} \leftarrow V_k - A_k + B_k$
  until $|f(x_k) - f(x_{k-1})| \leq \varepsilon_a + \varepsilon_r |f(x_{k-1})|$ and $\|g(x_{k-1})\| \leq \varepsilon_g$
```

3.7.2 Broyden–Fletcher–Goldfarb–Shanno (BFGS) Method

The DFP update was soon superseded by the BFGS formula, which is generally considered to be the most effective quasi-Newton update. Instead of solving the norm minimization problem (3.41) of B we now solve the same problem for its inverse V using equivalent conditions and and the solution is given by,

$$V_{k+1} = \left[I - \frac{s_k y_k^T}{s_k^T y_k} \right] V_k \left[I - \frac{y_k s_k^T}{s_k^T y_k} \right] + \frac{s_k s_k^T}{s_k^T y_k}. \tag{3.46}$$

The relative performance between the DFP and BFGS methods is problem dependent.

Example 3.8. BFGS:
As for the previous methods, we apply BFGS to the function f given by (3.11). The solution path, starting at $(-0.1, 0.6)^T$, is shown in 3.10. Compare the path of BFGS with that from modified Newton's method (Figure 3.6) and the conjugate gradient method (Figure 3.5).

Example 3.9. Minimization of the Rosenbrock Function
Minimize the function,

$$(x) = 100 \left(x_2 - x_1^2 \right)^2 + (1 - x_1)^2, \tag{3.47}$$

starting from $x_0 = (-1.2, 1.0)^T$.

Figure 3.9: Broyden, Fletcher, Goldfarb and Shanno at the NATO Optimization Meeting (Cambridge, UK, 1983), which was a seminal meeting for continuous optimization (courtesy of Prof. John Dennis)

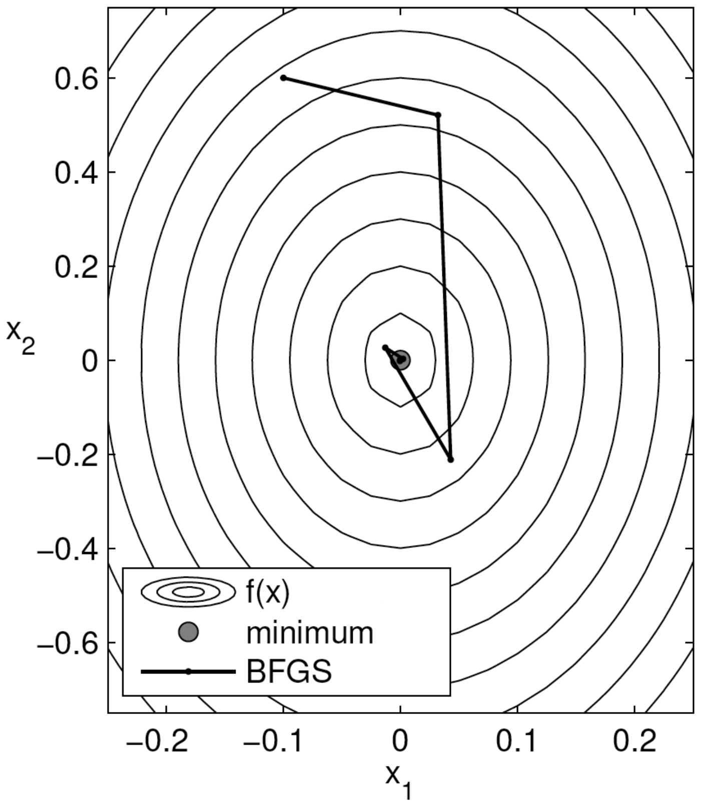


Figure 3.10: Solution path of the BFGS method.

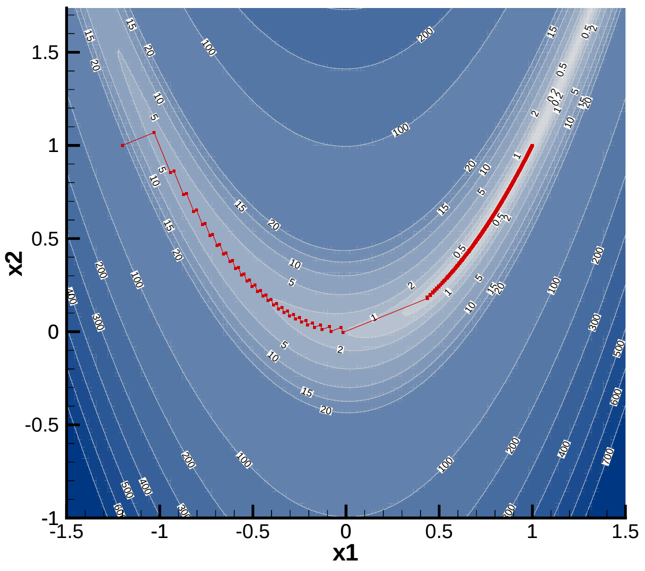


Figure 3.11: Solution path of the steepest descent method

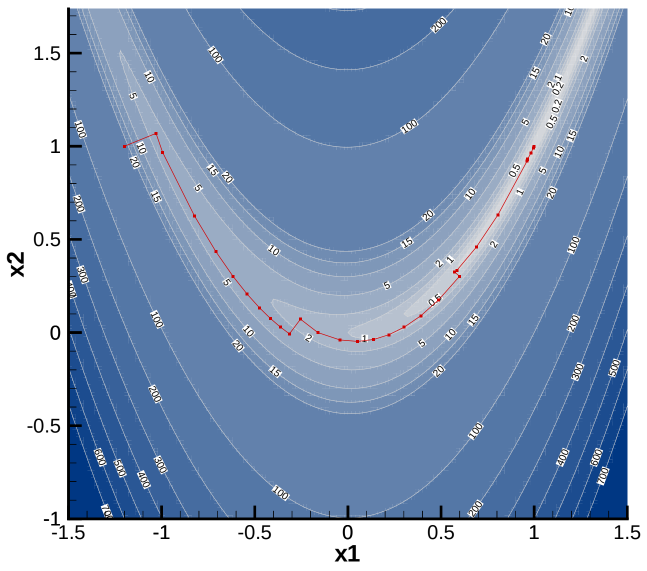


Figure 3.12: Solution path of the nonlinear conjugate gradient method

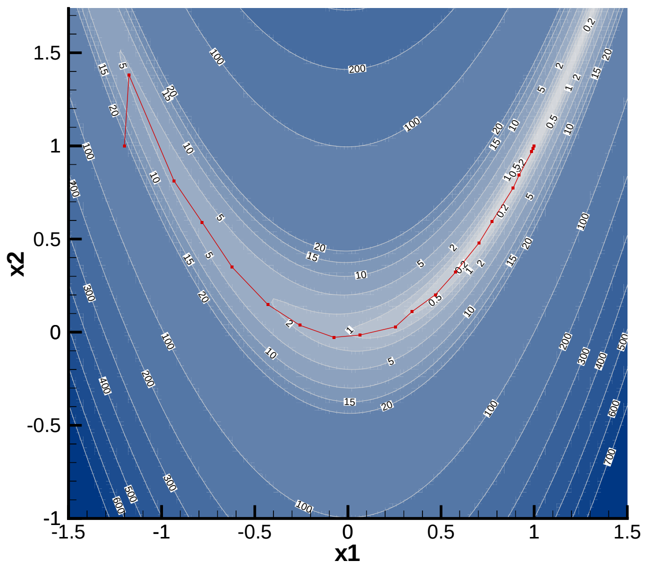


Figure 3.13: Solution path of the modified Newton method

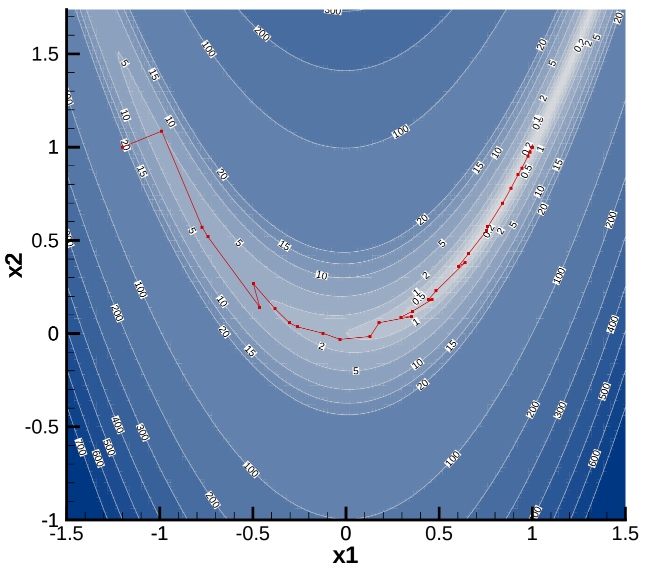


Figure 3.14: Solution path of the BFGS method

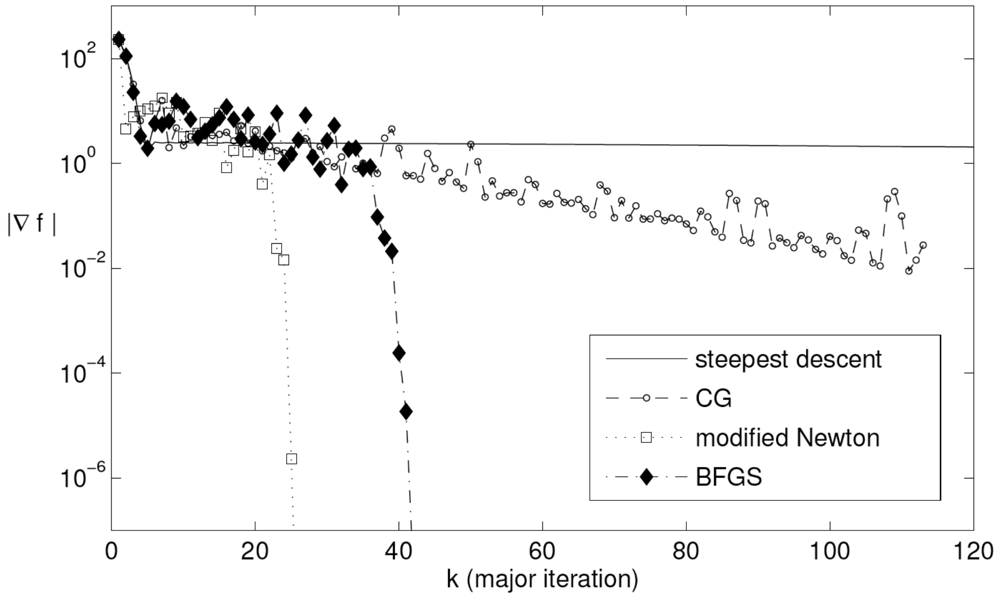


Figure 3.15: Comparison of convergence rates for the Rosenbrock function

3.7.3 Symmetric Rank-1 Update Method (SR1)

If we drop the requirement that the approximate Hessian (or its inverse) be positive definite, we can derive a simple rank-1 update formula for B_k that maintains the symmetry of the matrix and satisfies the secant equation. Such a formula is given by the symmetric rank-1 update (SR1) method (we use the Hessian update here, and not the inverse V_k):

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}. \tag{3.48}$$

With this formula, we must consider the cases when the denominator vanishes, and add the necessary safe-guards:

1. if $y_k = B_k s_k$ then the denominator is zero, and the only update that satisfies the secant equation is $B_{k+1} = B_k$ (i.e., do not change the matrix).

2. if $y_k \neq B_k s_k$ and $(y_k - B_k s_k)^T s_k = 0$ then there is no symmetric rank-1 update that satisfies the secant equation.

To avoid the second case, we update the matrix only if the following condition is met:

$$|y_k^T (s_k - B_k y_k)| \geq r \|s_k\| \|y_k - B_k s_k\|,$$

where $r \in (0, 1)$ is a small number (e.g., $r = 10^{-8}$). Hence, if this condition is not met, we use $B_{k+1} = B_k$.

Why would we be interested in a Hessian approximation that is potentially indefinite? In practice, the matrices produced by SR1 have been found to approximate the true Hessian matrix well (often better than BFGS). This may be useful in trust-region methods (see next section) or constrained optimization problems; in the latter case the Hessian of the Lagrangian is often indefinite, even at the minimizer.

To use the SR1 method in practice, it may be necessary to add a diagonal matrix γI to B_k when calculating the search direction, as was done in modified Newton's method. In addition, a simple back-tracking line search can be used, since the Wolfe conditions are not required as part of the update (unlike BFGS).

3.8 Trust Region Methods

Trust region, or "restricted-step" methods are a different approach to resolving the weaknesses of the pure form of Newton's method, arising from an Hessian that is not positive definite or a highly nonlinear function.

One way to interpret these problems is to say that they arise from the fact that we are stepping outside a the region for which the quadratic approximation is reasonable. Thus we can overcome this difficulties by minimizing the quadratic function within a region around x_k within which we *trust* the quadratic model.

The reliability index, r_k, is the ratio of the actual reduction to the predicted reduction; the closer it is to unity, the better the agreement. If $f_{k+1} > f_k$ (new point is worse), r_k is negative.

Algorithm 9 Trust region algorithm

```
Input: Initial guess $x_0$, convergence tolerances, $\varepsilon_g, \varepsilon_a$ and $\varepsilon_r$, initial size of the trust region, $h_0$
Output: Optimum, $x^*$
  $k \leftarrow 0$
  repeat
    Compute the Hessian of the objective function $H(x_k)$, and solve the quadratic subproblem:
        $$\begin{aligned} \text{minimize} \quad & q(s_k) = f(x_k) + g(x_k)^T s_k + \frac{1}{2} s_k^T H(x_k) s_k \\ \text{w.r.t.} \quad & s_k \\ \text{s.t.} \quad & -h_k \le s_k \le h_k, \quad i = 1, \ldots, n \end{aligned}$$
    Evaluate $f(x_k + s_k)$ and compute the ratio that measures the accuracy of the quadratic model,
        $$r_k \leftarrow \frac{f(x_k) - f(x_k + s_k)}{f(x_k) - q(s_k)} = \frac{\Delta f}{\Delta q}$$
    if $r_k < 0.25$ then
      $h_{k+1} \leftarrow \frac{\|s_k\|}{4}$ {Model is not good; shrink the trust region}
    else if $r_k > 0.75$ and $h_k = \|s_k\|$ then
      $h_{k+1} \leftarrow 2h_k$ {Model is good and new point on edge; expand trust region}
    else
      $h_{k+1} \leftarrow h_k$ {New point with trust region and the model is reasonable; keep trust region the
      same size}
    end if
    if $r_k \le 0$ then
      $x_{k+1} \leftarrow x_k$ {Keep trust region centered about the same point}
    else
      $x_{k+1} \leftarrow x_k + s_k$ {Move center of trust region to new point}
    end if
    $k \leftarrow k + 1$
  until $|f(x_k) - f(x_{k-1})| \le \varepsilon_a + \varepsilon_r |f(x_{k-1})|$ and $\|g(x_{k-1})\| \le \varepsilon_g$
```

The initial value of h is usually 1. The same stopping criteria used in other gradient-based methods are applicable.

Bibliography

[1] Ashok D. Belegundu and Tirupathi R. Chandrupatla. *Optimization Concepts and Applications in Engineering*, chapter 3. Prentice Hall, 1999.

[2] Y.H. Dai and Y. Yuan. A nonlinear conjugate gradient with a strong global convergence property. *SIAM Journal on Optimization*, 10(1):177–182, 1999.

[3] William C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, February 1991.

[4] J.E. Dennis and J. J. Moreé. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19(1):46–89, 1977.

[5] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168, 1963. doi: 10.1093/comjnl/6.2.163.

[6] Chinyere Onwubiko. *Introduction to Engineering Design Optimization*, chapter 4. Prentice Hall, 2000.