Chapter 2

Numerical optimization

2.1 Algorithms for optimization of single-variable functions. Bracketing techniques

Consider a single variable real-valued function $f(x) : [a, b] \rightarrow \mathbb{R}$ for which it is required to find an optimum in the interval $[a,\ b]$. Among the algorithms for univariate optimization, the golden section and the Fibonacci search techniques are fast, accurate, robust and they do not require derivatives, (Sinha, 2007; Press et al., 2007; Mathews, 2005). These methods can be used if the optimum is bracketed in a finite interval and the function is unimodal on $[a,\ b]$. The algorithms will be described for finding a minimum. The problem of finding a maximum is similar and may be solved with a very slight alteration of the algorithm or by changing the sign of the function.

A function $f(x)$ is *unimodal* on an interval $[a, b]$ if it has exactly one local minimum x^* on the interval, i.e., it is strictly decreasing on $[a, x^*]$ and is strictly increasing on $[x^*, b]$.

The initial interval $[a, b]$ is called the initial interval of uncertainty. The general strategy of golden section and Fibonacci methods is to progressively reduce the interval. This can be accomplished by placing experiments and eliminating parts of the initial interval that do not contain the optimum. Consider the interval divided by two points x_1 and x_2 located at the same distance from the boundaries a and b, as shown in Figure 2.2, i.e., the distance from a to x_2 is the same as the distance from x_1 to b. The function value is evaluated

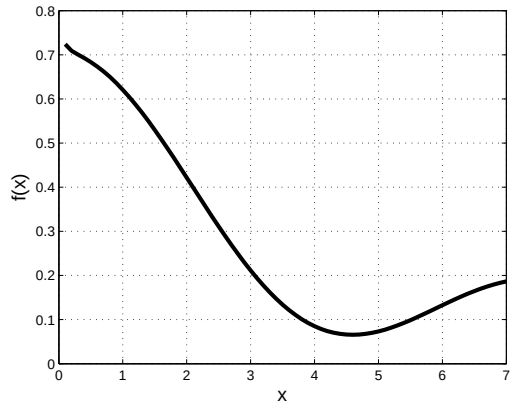


Figure 2.1: Unimodal function

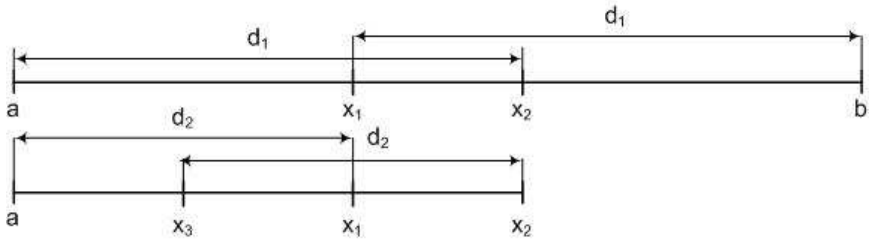


Figure 2.2: Bracketing the minimum

in these two points and the interval is reduced from a to x_1 if $f(x_1) \leq f(x_2)$, or from x_2 to b, otherwise. The procedure is iterative and it stops when the length of the interval is smaller than a given tolerance ϵ.

2.1.1 Golden Section Search

The golden section search technique (Press et al., 2007; Mathews, 2005) evaluates the function values at two interior points x_1 and x_2 chosen such that each one of them divides the interval in a ratio r, with $\frac{1}{2} < r < 1$ to preserve the order $a < x_1 < x_2 < b$. As shown in Figure 2.2, the distance d_1 can be written as:

$$d_1 = x_2 - a = b - x_1 \tag{2.1}$$

and it is equal to a fraction r of the original interval length, $b - a$. This gives:

$$x_2 - a = r(b - a) \tag{2.2}$$

$$b - x_1 = r(b - a) \tag{2.3}$$

We shall assume now that the function has a smaller value at x_1 than x_2, thus the new search interval will be $[a, x_2]$. The demonstration is similar for the case when $f(x_2) > f(x_1)$. A point x_3 is introduced within the new interval so it is located at the same distance, d_2, from x_2 as x_1 is from a:

$$d_2 = x_1 - a = x_2 - x_3 \tag{2.4}$$

The ratio r must remain constant on each new interval, which means:

$$x_1 - a = r(x_2 - a) \tag{2.5}$$

$$x_2 - x_3 = r(x_2 - a) \tag{2.6}$$

If x_2 from (2.2) and x_1 from (2.3) are replaced into (2.5), the value of r can be determined as follows:

$$b - r(b - a) - a = r\left[a + r(b - a) - a\right] \tag{2.7}$$

Rearranging, we get:

$$(b - a)(1 - r - r^2) = 0 \tag{2.8}$$

and since the initial length of the interval is nonzero, the ratio r is one of the roots of

$$1 - r - r^2 = 0 \tag{2.9}$$

The solutions of (2.9) are:

$$r_1 = \frac{-1 + \sqrt{5}}{2} = 0.618, \quad r_2 = \frac{-1 - \sqrt{5}}{2} \tag{2.10}$$

but only r_1 obeys the initial requirement that $\frac{1}{2} < r < 1$.

The golden section search guarantees that after each new interval reduction the minimum will be bracketed in an interval having the length equal to 0.618 times the size of the preceding one.

Algorithm 1 summarizes the technique for finding the minimum of a single variable function $f(x)$ within a given interval $[a, b]$ with a tolerance ε.

Algorithm 1 Golden Section Search

```
Define function f(x)
Define boundaries a, b and tolerance ε
d = b − a
while b − a ≥ ε do
  d ← d × 0.618
  x1 ← b − d
  x2 ← a + d
  if f(x1) ≤ f(x2) then
    b ← x2
  else
    a ← x1
  end if
end while
```

2.1.2 Fibonacci Search

The Fibonacci search is an optimization method similar to the golden section search. It is used to find the minimum or maximum of a unimodal function, $f(x)$, on a closed interval, $[a,\ b]$, (Jeter, 1986; Pierre, 1986; Luenberger, 2003).

The Fibonacci search is the strategy that yields the smallest possible interval of uncertainty in which the optimum lies, after n tests are completed, (Pierre, 1986), or ensures the final interval is within the user-specified tolerance (ε) in a number n of iterations.

The method is based on the Fibonacci numbers which are defined by:

$$F_0 = F_1 = 1, \quad F_k = F_{k-1} + F_{k-2}, \quad k = 2, 3, ... \tag{2.11}$$

The first numbers of this sequence are $1, 1, 2, 3, 5, 8, 13, ...$

In Figure 2.3 consider the procedure is at the iteration k. As in the golden section search we shall determine the ratio r_k. In this case r_k is not constant but varies with the number of iterations (Mathews, 2005).

The function f is tested at x_{1k} and x_{2k} and the interval will be reduced around the minimum. Without loss of generality, assume the minimum of

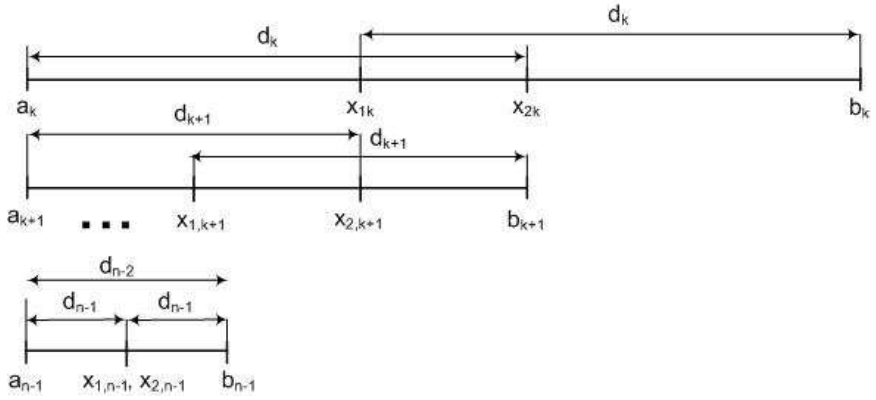


Figure 2.3: Fibonacci search

the two values at x_{1k}, thus the new search interval is $[a_{k+1} = a_k, b_{k+1} = x_{2k}]$. The point x_{1k} is relabeled as $x_{2,k+1}$.

At each iteration, the interval is reduced by a ratio $\frac{1}{2} < r_k < 1$, so the distance d_k is calculated as:

$$d_k = r_k(b_k - a_k) = r_k d_{k-1} \tag{2.12}$$

and each new interval has the length d_k.

We shall obtain a relation between r_k and r_{k+1} starting with the observation that:

$$x_{2k} - x_{1k} = b_{k+1} - x_{2,k+1} \tag{2.13}$$

From Figure 2.3, the left hand side of the relation (2.13) can be written as:

$$\begin{aligned} x_{2k} - x_{1k} &= (b_k - a_k) - (x_{1k} - a_k) - (b_k - x_{2k}) \\ &= (b_k - a_k) - (1 - r_k)(b_k - a_k) - (1 - r_k)(b_k - a_k) \\ &= (b_k - a_k)(2r_k - 1) \end{aligned} \tag{2.14}$$

and the right hand side:

$$\begin{aligned} b_{k+1} - x_{2,k+1} &= (1 - r_{k+1})(b_{k+1} - a_{k+1}) \\ &= (1 - r_{k+1})r_k(b_k - a_k) \end{aligned} \tag{2.15}$$

The relations (2.13), (2.14) and (2.15) give:

$$2r_k - 1 = (1 - r_{k+1})r_k \tag{2.16}$$

or

$$r_{k+1} = \frac{1 - r_k}{r_k} \tag{2.17}$$

Now we introduce the Fibonacci numbers. Replace:

$$r_k = \frac{F_{n-k}}{F_{n-k+1}} \tag{2.18}$$

and we obtain:

$$r_{k+1} = \frac{1 - \frac{F_{n-k}}{F_{n-k+1}}}{\frac{F_{n-k}}{F_{n-k+1}}} = \frac{F_{n-k-1}}{F_{n-k}} \tag{2.19}$$

When $k = \overline{1, n-1}$ the ratio r_k will have the values:

$$\begin{aligned} r_1 &= \frac{F_{n-1}}{F_n} \\ r_2 &= \frac{F_{n-2}}{F_{n-1}} \\ &... \\ r_{n-2} &= \frac{F_2}{F_3} \\ r_{n-1} &= \frac{F_1}{F_2} = \frac{1}{2} \end{aligned} \tag{2.20}$$

At the last step, $n-1$, no new points can be added. The distance $d_{n-1} = d_{n-2}r_{n-1} = d_{n-2}/2$, i.e., at step $n-1$ the points $x_{1,n-1}$ and $x_{2,n-2}$ are equal and:

$$r_n = \frac{F_0}{F_1} = \frac{1}{1} = 1 \tag{2.21}$$

The last interval obtained is $d_n = r_n(b_n - a_n) = r_n d_{n-1} = d_{n-1}$ and no other steps are possible. It may be written as:

$$\begin{aligned} d_n &= r_n d_{n-1} = \frac{F_0}{F_1} d_{n-1} = \frac{F_0}{F_1}\frac{F_1}{F_2} d_{n-1}... \\ &= \frac{F_0}{F_1}\frac{F_1}{F_2} \cdots \frac{F_{n-2}}{F_{n-1}}\frac{F_{n-1}}{F_n} d_1 = \frac{F_0}{F_n} d_0 = \frac{1}{F_n} d_0 \end{aligned} \tag{2.22}$$

The distance d_0 is the initial length of the interval of uncertainty, $d_0 = b - a$.

If the minimizer has to be found with a tolerance of ε, then the length of the last interval must be:

$$\frac{1}{F_n}(b-a) < \varepsilon \tag{2.23}$$

and the necessary number of iterations to reach the minimum with the tolerance ε is the smallest integer for which the following inequality holds:

$$F_n > \frac{b-a}{\varepsilon} \tag{2.24}$$

Algorithm 2 summarizes the Fibonacci search.

Algorithm 2 Fibonacci Search

```
Define function f(x)
Define boundaries a, b and tolerance ε
Initialize the Fibonacci sequence F_0 = F_1 = 1
while F_n ≤ (b − a)/ε do
  Calculate Fibonacci numbers F_n = F_{n−1} + F_{n−2}
end while
d = b − a
for k=1:n do
  d ← d × F_{n−k}/F_{n−k+1}
  x_1 ← b − d
  x_2 ← a + d
  if f(x_1) ≤ f(x_2) then
    b ← x_2
  else
    a ← x_1
  end if
end for
```

If the number of iterations has to be determined from knowledge of the tolerance ϵ, the Fibonacci numbers are calculated and stored until the inequality (2.24) becomes true.

If the number of iterations is given, a more convenient way to calculate the Fibonacci numbers is be the use of explicit formula:

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right] \tag{2.25}$$

Both algorithms for single-variable function optimization presented above can be applied when the function f is not differentiable. For a small number of iterations the Fibonacci method is more efficient since it provides the smallest final interval of uncertainty in a given number of steps. When n is large, the methods are almost identical because the ratio F_{n-k}/F_{n-k+1} converges to the golden ratio 0.618, when k approaches ∞.

2.2 Algorithms for unconstrained multivariable optimization

2.2.1 Basic concepts

Over the last forty years many powerful algorithms have been developed for unconstrained function optimization. All of these require an initial estimate of the optimum point, denoted by $\mathbf{x}_0$. This is determined from knowledge about the application. Beginning at $\mathbf{x}_0$, optimization algorithms generate a sequence of iterates $\mathbf{x}_k,\ k = 1, 2, ...$, that terminate when either no more progress can be made, or when a solution point has been approximated with sufficient accuracy. The decision of moving from one point $\mathbf{x}_k$ to the next is taken based on the information about the function f at $\mathbf{x}_k$ and finding a new iterate $\mathbf{x}_{k+1}$ with a lower function value than $\mathbf{x}_k$, (Snyman, 2005; Nocedal and Wright, 1999).

The algorithms terminate when one, or all, of the following situations occur:

- two successive points are closer than a specified tolerance ε_1:

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\| < \varepsilon_1 \tag{2.26}$$

- the function value calculated at two successive points does not vary more than a given tolerance ε_2:

$$|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \varepsilon_2 \tag{2.27}$$

- the gradient of f is sufficiently close to 0:

$$\|\nabla f(\mathbf{x}_k)\| < \varepsilon_3 \tag{2.28}$$

where $\|\mathbf{x}\|$ is the Euclidian norm of a vector $\mathbf{x} = [x_1\ x_2\ ...\ x_n]^T$, given by:

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2} \tag{2.29}$$

In general, there are two strategies from moving from one point to the next: *line search* and *trust region*. An extensive description of algorithms from both categories can be found in (Nocedal and Wright, 1999; Snyman, 2005; Conn et al., 2000).

2.2.1.1 Line search algorithms

Searching along a line for a minimum point is a basic part of most nonlinear programming algorithms. To initiate a line search with respect to a function f, two vectors must be specified: the initial point $\mathbf{x_0}$ and the direction $\mathbf{d}$ in which the search is to be made, (Luenberger, 2003).

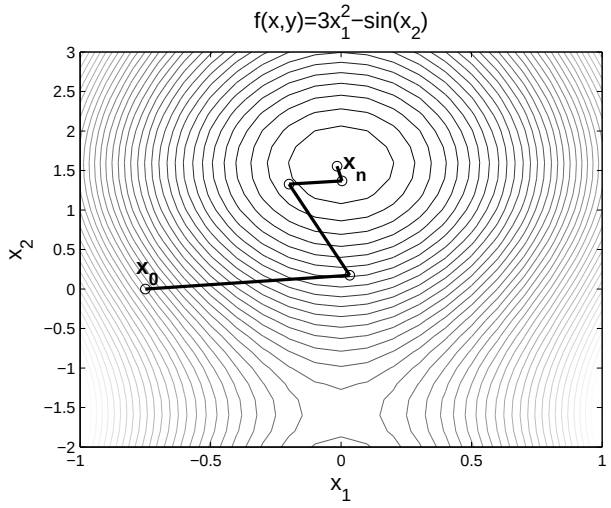


Figure 2.4: Sequence of steps for minimization of a two variable function

Algorithms of this type calculate a new point $\mathbf{x_{k+1}}$ (Figure 2.4) from:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + s_k \mathbf{d_k} \tag{2.30}$$

where $\mathbf{x_k}$ and $\mathbf{d_k}$ are the current point and the direction of search, respectively, and s_k is the step length which is determined by solving a one-dimensional minimization problem:

$$F(s_k) = f(\mathbf{x_k} + s_k \mathbf{d_k}) \tag{2.31}$$

The line search is necessary for the following reason. If the step size is very small, the value of the function decreases by a small amount. However, large values of the step length may lead to the situation when the minimum point is again not reached in a small number of iterations as shown in Figure 2.5. The function f is reduced at every step but with a very small amount compared to the step length.

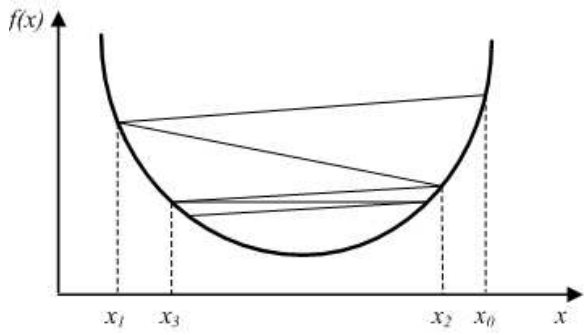


Figure 2.5: Large step size

The general structure of a line search descent method is given in Algorithm 3. Various descent methods within this class differ according to the way in which the descent directions are chosen. Another important consideration is the method used for line search (Snyman, 2005).

A possibility to control the step size control is to implement a procedure for minimization of $F(s_k)$ based on golden ratio or Fibonacci search as presented in previous sections.

Sophisticated algorithms for one-dimensional minimization are in general not chosen because they would require a large number of function eval-

Algorithm 3 General scheme for line search algorithms

Define function $f(\mathbf{x})$ and tolerance ε
Select a starting point $\mathbf{x_0}$
repeat
  Select a descent direction $\mathbf{d_k}$
  Perform a one-dimensional line search in the direction $\mathbf{d_k}$, i.e.

$$\min_{s_k} F(s_k) = \min_{s_k} f(\mathbf{x_k} + s_k \mathbf{d_k})$$

  or compute a step length that provides a decrease in the value of f (e.g. using Armijo, Goldstein or Wolfe rules)
  Compute

$$\mathbf{x_{k+1}} = \mathbf{x_k} + s_k \mathbf{d_k}$$

until one of the conditions (2.26), (2.27) or (2.28) are true

uations and thus, a significant increase of the computation time. More practical strategies perform an inexact line search to identify a step length that achieves adequate reductions in f. Some line search algorithms try out a sequence of candidates for s_k stopping to accept one of these values when certain conditions are satisfied. A detailed description of Armijo, Goldstein and Wolfe conditions is given in (Nocedal and Wright, 1999; Snyman, 2005; Luenberger, 2003).

A basic *backtracking* procedure for line search is given in algorithm 4, (Gould and Leyffer, 2002).

Algorithm 4 Backtracking line search

Select an initial step length s_0 (e.g. $s_0 = 1$) and set $i = 0$
repeat
  $i \leftarrow i + 1$
  Set $s_i = \tau s_{i-1}$, where $\tau \in (0, 1)$, (e.g. $\tau = \frac{1}{2}$)
until $f(\mathbf{x}_k + s_i \mathbf{d}_k) < f(\mathbf{x}_k)$
Set $s_k = s_i$

For a given direction $\mathbf{d}_k$, at the current point $\mathbf{x}_k$, this algorithm will decrease the step size until the value of f at the next point is smaller than $f(\mathbf{x}_k)$. It will prevent the step getting too small, but does not prevent large step sizes relative to the decrease of f.

The *Armijo condition* is formulated such that it will provide a *sufficient*

decrease in f. Consider the function F defined by (2.31). For a fixed α, $0 < \alpha < 1$, the step s_k is considered to be not too large and gives a sufficient decrease of f if, (Luenberger, 2003; Gould and Leyffer, 2002):

$$F(s_k) \leq F(0) + \alpha F'(0) s_k \tag{2.32}$$

Condition (2.32) is illustrated in Figure 2.6.

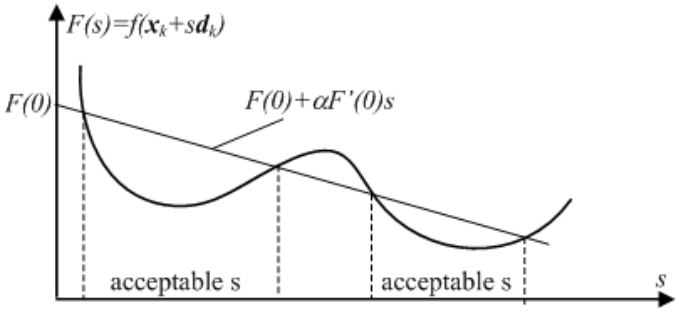


Figure 2.6: Armijo condition

In terms of the original notation, the condition (2.32) is obtained from:

$$\begin{aligned} F(0) &= f(\mathbf{x_k} + 0 \cdot \mathbf{d_k}) = f(\mathbf{x_k}) && (2.33)\\ F'(s_k) &= \mathbf{d}_k^T \nabla f(\mathbf{x_k} + s_k \mathbf{d_k}) && (2.34)\\ F'(0) &= \mathbf{d}_k^T \nabla f(\mathbf{x_k}) && (2.35)\\ F(0) + \alpha F'(0) s_k &= f(\mathbf{x_k}) + \alpha \mathbf{d}_k^T \nabla f(\mathbf{x_k}) s_k && (2.36) \end{aligned}$$

and (2.32) becomes:

$$f(\mathbf{x_k} + s_k \mathbf{d_k}) \leq f(\mathbf{x_k}) + \alpha \mathbf{d}_k^T \nabla f(\mathbf{x_k}) s_k \tag{2.37}$$

A backtracking-Armijo algorithm is given below.

Another popular approaches to determine an appropriate step size are Goldstein and Wolfe tests, (Luenberger, 2003; Nocedal and Wright, 1999).

The *Goldstein test* for the step length requires that for an α selected such that $0 < \alpha < \frac{1}{2}$:

$$f(\mathbf{x_k}) + (1-\alpha) \mathbf{d}_k^T \nabla f(\mathbf{x_k}) s_k \leq f(\mathbf{x_k} + s_k \mathbf{d_k}) \leq f(\mathbf{x_k}) + \alpha \mathbf{d}_k^T \nabla f(\mathbf{x_k}) s_k \tag{2.38}$$

Algorithm 5 Backtracking Armijo

Select an initial step length s_0 (e.g. $s_0 = 1$), $\alpha \in (0, 1)$ and set $i = 0$
repeat
  $i \leftarrow i + 1$
  Set $s_i = \tau s_{i-1}$, where $\tau \in (0, 1)$, (e.g. $\tau = \frac{1}{2}$)
until $f(\mathbf{x_k} + s_i\mathbf{d_k}) \leq f(\mathbf{x_k}) + \alpha\mathbf{d}_k^T\nabla f(\mathbf{x_k})s_i$
Set $s_k = s_i$

The *Wolfe* test requires an additional condition next to relation (2.37) to be satisfied:

$$f(\mathbf{x_k} + s_k\mathbf{d_k}) \leq f(\mathbf{x_k}) + \alpha_1\mathbf{d}_k^T\nabla f(\mathbf{x_k})s_k \tag{2.39}$$

$$\mathbf{d}_k^T\nabla f(\mathbf{x_k} + s_k\mathbf{d_k}) \geq \alpha_2\mathbf{d}_k^T\nabla f(\mathbf{x_k}) \tag{2.40}$$

where $0 < \alpha_1 < \alpha_2 < 1$.

2.2.1.2 Trust region methods

Another strategy approximates the function f to be minimized, with a model function, m_k in a neighborhood, N, named the trust region, around the current point $\mathbf{x_k}$. The subproblem which is to be solved now is minimizing the model function over N. If the solution does not produce a sufficient decrease of f, the trust region is shrunk around $\mathbf{x_k}$. The model function m_k is usually a quadratic form obtained from a two-term Taylor series approximation.

The general algorithm can be summarized as:

- Choose a maximum distance from the current point - the trust region radius
- Find a direction and a step that produce the best improvement possible in this region
- If the step is unsatisfactory, reduce the size of the trust region and start again

Algorithms using trust region strategy are discussed in detail in (Nocedal and Wright, 1999; Conn et al., 2000).

2.2.2 Newton methods

2.2.2.1 The algorithm

Newtons method (known also as Newton-Raphson method), with all its variations, is one of the most important methods in numerical unconstrained optimization. Basically, it is a numerical procedure for finding the roots of a real-valued function. It can also be used to find local extrema of functions using the necessary condition for a point to be stationary in unconstrained optimization.

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a multivariable function to be minimized or maximized. A point $\mathbf{x_0}$ is a stationary point of f if the gradient calculated at this location is zero, i.e., $\nabla f(\mathbf{x_0}) = 0$. Thus, the Newton method can be applied to find the roots of $\nabla f(\mathbf{x})$.

Assume that the function f is twice differentiable and the Hessian $\mathbf{H}(\mathbf{x})$ is continuous. The elements of $\mathbf{H}(\mathbf{x})$ are the second derivatives of the objective function $H_{ij} = \partial^2 f / \partial x_i \partial x_j$. In case the Hessian is not available and its computation is difficult or not possible, other methods must be applied.

The Newton method for the minimization of $f(\mathbf{x})$ can be derived assuming that, given $\mathbf{x}_k$, the next point $\mathbf{x}_{k+1}$ is obtained by minimizing a quadratic approximation of f. The Taylor expansion of $f(\mathbf{x_{k+1}})$ around $\mathbf{x}_k$ is:

$$f(\mathbf{x_{k+1}}) \approx f(\mathbf{x}_k) + (\mathbf{x}_{k+1} - \mathbf{x}_k)^T \nabla f(\mathbf{x}_k) + \frac{1}{2}(\mathbf{x}_{k+1} - \mathbf{x}_k)^T \mathbf{H}(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) \tag{2.41}$$

The stationary point of (2.41) is computed from setting the gradient of $f(\mathbf{x_{k+1}})$ equal to zero:

$$\nabla f(\mathbf{x_{k+1}}) = \nabla f(\mathbf{x}_k) + \mathbf{H}(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = 0 \tag{2.42}$$

From (2.42):

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k) \tag{2.43}$$

which is the iterative scheme of the method.

For a multivariable function $f(\mathbf{x})$ the Newton method can be described as in Algorithm 6.

The stop criterion for this algorithm can be one or more of the conditions (2.26), (2.27) or (2.28), or, resulting from the same relations, the norm of

$[\mathbf{H}(\mathbf{x}_k)]^{-1}\nabla f(\mathbf{x}_k)$ has to be sufficiently small:

$$||H(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)|| \leq \varepsilon \tag{2.44}$$

Algorithm 6 Newton Search Method

Define $f(\mathbf{x})$ and a tolerance ε
Compute the gradient $\nabla f(\mathbf{x})$
Compute the Hessian matrix $\mathbf{H}(\mathbf{x})$
Choose an initial point $\mathbf{x_0}$
Set $k = 0$
while $||H(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)|| > \varepsilon$ **do**
  Calculate a new point $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$
  Set $k \leftarrow k + 1$
end while

Note that for a single variable function $f(x)$ the iterative scheme can be written as:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{2.45}$$

2.2.2.2 Geometrical interpretation

The method may be introduced by a geometrical interpretation for a single variable function $f(x)$. Suppose we want to find a root of some differentiable function $y = g(x)$. In case $g(x) = f'(x)$, the problem of finding a root of $g(x)$ is the same as finding a stationary point for $f(x)$.

As shown in Figure 2.7 we choose an initial point x_0. Consider the tangent line to the graph of g at the point $(x_0, g(x_0))$.

Suppose this tangent line crosses the x-axis at $x = x_1$. A new tangent line to the graph of g in x_1 will intersect the axis at $x = x_2$. If this process is repeated, the points x_k get closer to a root of g.

We will derive a formula for calculating x_{k+1} in terms of x_k. The tangent line at $(x_k, g(x_k))$ is:

$$g(x) - g(x_k) = g'(x_k)(x - x_k) \tag{2.46}$$

To obtain the point of intersection with the x-axis we set $g(x) = 0$ and solve

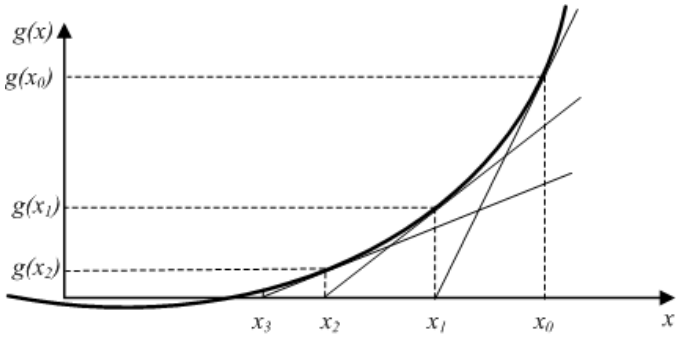


Figure 2.7: Newton method

for x:

$$0 - g(x_k) = g'(x_k)(x - x_k) \tag{2.47}$$

$$x = x_k - \frac{g(x_k)}{g'(x_k)} \tag{2.48}$$

Let us denote the new point by x_{k+1}. The iterative scheme is:

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)} \tag{2.49}$$

When the function g is the first derivative of a function f, $g(x) = f'(x)$, the relation (2.49) can be written as:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{2.50}$$

2.2.2.3 Implementation aspects

If the function $f(\mathbf{x}) = f(x_1, x_2, ...x_n)$ is quadratic, i.e., it can be written in the form

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c \tag{2.51}$$

where $\mathbf{A}$ is a symmetric $n \times n$ matrix, $\mathbf{b}$ is an $n \times 1$ vector and c a scalar, the relation (2.41) is not an approximation and the stationary point can be determined in one step.

The gradient of f is obtained as:

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b} \tag{2.52}$$

and the solution of the optimization problem is:

$$\mathbf{x}^* = -\mathbf{A}^{-1}\mathbf{b} \tag{2.53}$$

The Newton method will converge to the exact stationary point in one step, no matter which is the initial point selected.

Example 2.1 *Consider the function* $f(x_1, x_2) = x_1^2 + 2x_2^2 + x_1 + 7$

It can be written in the form (2.51) as:

$$f(x_1, x_1) = \frac{1}{2}[x_1 \; x_2] \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [1 \; 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 7 \tag{2.54}$$

Let $\mathbf{x}_0 = [x_{10} \; x_{20}]^T$ *be the initial point of the Newton algorithm. The gradient and Hessian matrix for function* f *are:*

$$\nabla f(x_1, x_2) = \begin{bmatrix} 2x_1 + 1 \\ 4x_2 \end{bmatrix}, \quad \mathbf{H}(x_1, x_2) = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix} \tag{2.55}$$

The next point is calculated from:

$$\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{H}(\mathbf{x}_0)^{-1} \nabla f(\mathbf{x}_0) \tag{2.56}$$

or:

$$\begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} x_{10} \\ x_{10} \end{bmatrix} - \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} 2x_{10} + 1 \\ 4x_{20} \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} \\ 0 \end{bmatrix} \tag{2.57}$$

Notice that the gradient calculated at this new point $[-\frac{1}{2} \quad 0]$ *is zero, thus the condition (2.44) is satisfied and the algorithm stops here.*

If the function is not quadratic the selection of the initial point is very important for the location of the extremum found by the Newton algorithm.

The convergence towards a local extremum is very fast, but the method does not return the type of the stationary point unless the second order conditions are verified at the solution.

Since the calculus of the second order derivatives and the Hessian matrix is a requirement for the implementation of the Newton method, the second order conditions can be easily examined.

Example 2.2 *Consider an univariate function $f(x) = x^4 - x^2$ for which we shall implement a Newton method to find local extrema. The first and second derivatives can be calculated symbolically:*

$$f'(x) = 4x^3 - 2x \tag{2.58}$$

$$f''(x) = 12x^2 - 2 \tag{2.59}$$

A starting point x_0 and a tolerance (i.e. $\varepsilon = 10^{-5}$) must be selected. The next points are calculated according to the iterative scheme:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{2.60}$$

Figures 2.8 and 2.9 illustrate the result of the Newton method for different values of the initial point x_0, the extremum obtained x^* and the type of extremum determined directly from the computation of the second derivative at x^*.

The advantages of Newton method include:

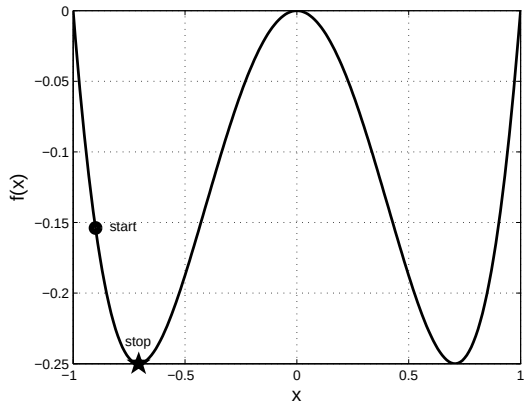
- The method is powerful and easy to implement
- It will converge to a local extremum from any sufficiently close starting value

However, the method has some drawbacks:

- The method requires the knowledge of the gradient and the Hessian, which is not always possible. For large problems, determining the Hessian matrix by the method of finite-differences is costly in terms of function evaluations.
- There is no guarantee of convergence unless the initial point is reasonably close to the extremum, thus the global convergence properties are poor.

2.2.2.4 Modified Newton method

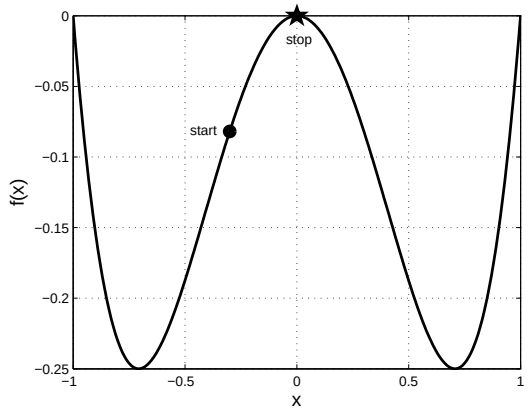
The classical Newton method requires calculation of the Hessian at each iteration. Moreover, the matrix has to be inverted which may be inconvenient



$$\begin{aligned} x_0 &= -0.9 \\ x^* &= -0.707 \\ f''(x^*) &= 4 > 0 \end{aligned}$$

$x*$ is a local minimum

Figure 2.8:



$$\begin{aligned} x_0 &= -0.3 \\ x^* &= 0 \\ f''(x^*) &= -2 < 0 \end{aligned}$$

$x*$ is a local maximum

Figure 2.9:

for large problems. A modified Newton method uses the Hessian calculated at the initial point. The convergence of the algorithm is still good for starting points close to extrema but the convergence is linear instead of quadratic (as in the case of the classical method).

Algorithm 7 describes the procedure.

Algorithm 7 Modified Newton method

Define $f(\mathbf{x})$ and a tolerance ε
Choose an initial point $\mathbf{x_0}$
Compute the gradient $\nabla f(\mathbf{x})$
Compute and evaluate the Hessian at the initial point, $\mathbf{H}(\mathbf{x}_0)$
Compute the inverse of the Hessian, $\mathbf{H}(\mathbf{x}_0)^{-1}$
while $||H(\mathbf{x}_0)^{-1}\nabla f(\mathbf{x}_k)|| > \varepsilon$ **do**
  Calculate a new point $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}(\mathbf{x}_0)^{-1}\nabla f(\mathbf{x}_k)$
  Set $k \leftarrow k + 1$
end while

2.2.3 Gradient methods

2.2.3.1 Directional derivative and the gradient

The *directional derivative* $\nabla_u f(x_0, y_0)$ is *the rate* at which the function f(x,y) *changes* at a point (x_0,y_0) in the direction u, (Weisstein, 2004a). It is a vector form of the usual derivative, and, when u is a unit vector, can be defined as:

$$\nabla_u f = \nabla f \cdot u \tag{2.61}$$

where ∇f is the gradient of function f and $(\cdot)$ is the scalar (or inner) product of vectors.

It generalizes the notion of a partial derivative, in which the direction is always taken parallel to one of the coordinate axes. The slopes in the direction of axes (x_1 and x_2 in Figure 2.10) are the partial derivatives, and the slopes in other directions (e.g. u) are the directional derivatives.

The inner product from the definition of the directional derivative can be expressed as:

$$\nabla_u f = \nabla f \cdot u = \|\nabla f\|\|u\| \cos\theta = \|\nabla f\| \cos\theta \tag{2.62}$$

where $\|\nabla f\|$ and $\|u\|$ are the norms of the corresponding vectors and θ is the angle between them (Figure 2.11). Because u is a unit vector, the norm is equal to one.

From (2.62) we obtain that:

- the largest possible value of $\nabla_u f$ is obtained when $\cos\theta$ assumes its

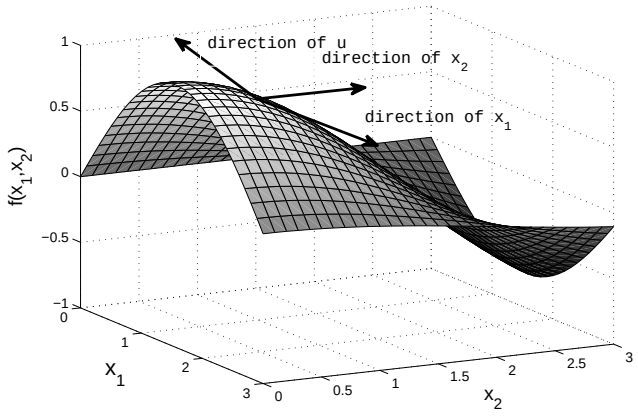


Figure 2.10: Directions of x_1, x_2 and u

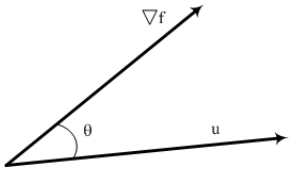


Figure 2.11: Inner product of vectors ∇f and u

maximum value of 1 at $\theta = 0$, i.e. the vector u points in the direction of the gradient ∇f. Thus, the directional derivative will be maximized in the direction of the gradient and this maximum value will be the magnitude of the gradient ($\|\nabla f\|$).

- if $\theta = \pi$, the cosine has its smallest value of -1 and the unit vector u points in the opposite direction of the gradient, $-\nabla f$. The directional derivative will be minimized in the direction of $-\nabla f$ and $\nabla_u f = -\|\nabla f\|$.

In other words, the function f will increase most rapidly (or it has the greatest positive rate of change) in the direction of the gradient and will decrease most rapidly in the direction of $-\nabla f$.

Example 2.3 *For example, consider a function of two variables:*

$$f(x_1, x_2) = sin(x_1) \cdot cos(0.8x_2) \tag{2.63}$$

The gradient of f represented as a vector field is shown in Figure2.12. It points in the direction of maximum increase of the function.

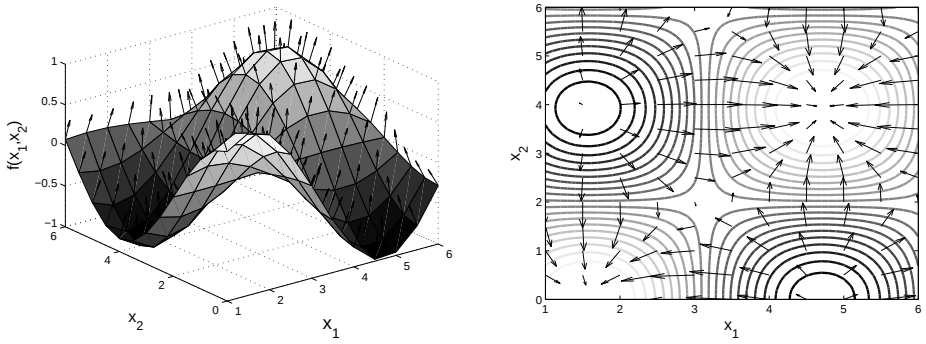


Figure 2.12: Direction of gradient

2.2.3.2 The method of steepest descent

The *method of steepest descent* is an algorithm for finding a *local minimum* of a function using the gradient. It belongs to a category named *first order methods* because they employ first order partial derivatives.

As explained in section 2.2.3.1, the gradient of a function f is a vector that points to the direction of maximum increase in the value of f. Therefore, going towards the opposite direction (given by $-\nabla f$)) means moving towards the minimum.

If a starting point $\mathbf{x}_0$ is selected in the neighborhood of a local minimum, the method moves in successive points, from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$ in the direction of the local downhill gradient (i.e. along the line extended from $\mathbf{x}_k$ in the direction opposite to the gradient, $-\nabla f(\mathbf{x}_k)$).

Another version of this algorithm, known as *steepest ascent*, will approach a local maximum if the search is performed in the direction of $\nabla f(\mathbf{x}_k)$.

The search starts at a point $\mathbf{x}_0$ and then slides down the gradient according to the iterative scheme:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - s_k \nabla f(\mathbf{x}_k) \tag{2.64}$$

where s_k is the step size for iteration k.

A usual implementation of the scheme uses the *normalized gradient* instead

of $\nabla f(\mathbf{x}_k)$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - s_k \frac{\nabla f(\mathbf{x}_k)}{\|\nabla f(\mathbf{x}_k)\|} \tag{2.65}$$

In this case, the distance between $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$ is exactly the step length s_k.

The next iterate is computed in the direction of the negative gradient at this new point $\mathbf{x}_{k+1}$ and we obtain a zig-zag pattern as illustrated in Figure 2.13. Iterations continue until the extremum has been determined within a chosen accuracy ϵ. The convergence criterion may be one of those given by the relations (2.26), (2.27) or (2.28).

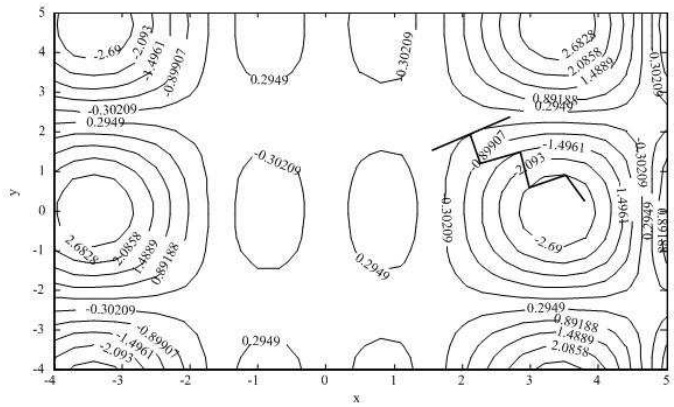


Figure 2.13: A path of steepest descent

The step size can be selected at the beginning of the procedure and fixed for all iterations. In this case it is possible to obtain oscillations around the local minimum and a very large number of iterations until the procedure will eventually converge to a solution.

Better implementations use a line search procedure to determine a step size that ensures a smaller value of the function at the next iteration, or minimizes $f(\mathbf{x}_k - s_k \nabla f(\mathbf{x}_k))$ with respect to s_k. This is often referred to as the *optimal gradient method*.

Alternatively, if an exact line search is laborious, the value of s_k, can be modified during the iterations, if necessary, making sure that one of Armijo, Goldstein or Wolfe conditions are satisfied.

Algorithm 8 describes the procedure.

Quadratic functions

Algorithm 8 Steepest Descent

Define function $f(\mathbf{x})$
Calculate the gradient $\nabla f(\mathbf{x})$
Select an initial point $\mathbf{x}_0$
Select a tolerance ε
Set $k = 0$
repeat
Perform a line search to determine the step length s_k that minimizes the function in the direction of minus gradient:

$$\min_{s_k>0} f\left(\mathbf{x}_k - s_k \frac{\nabla f(\mathbf{x}_k)}{\|\nabla f(\mathbf{x}_k)\|}\right)$$

Calculate a new point:

$$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - s_k \frac{\nabla f(\mathbf{x}_k)}{\|\nabla f(\mathbf{x}_k)\|}$$

Set $k \leftarrow k + 1$
until $\|\nabla f(\mathbf{x}_k)\| < \varepsilon$ (or $\|\mathbf{x}_{k+1} - \mathbf{x}_k\| < \varepsilon$ or $|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| < \varepsilon$)

In the particular case when the function to be minimized is quadratic, we are able to determine explicitly the optimal step size that minimizes $f(\mathbf{x}_k - s_k \nabla f(\mathbf{x}_k))$. Consider a function having the form:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{Q}\mathbf{x} + \mathbf{b}^T \mathbf{x} + c \tag{2.66}$$

where the $n \times n$ matrix $\mathbf{Q}$ is symmetric and positive definite, $\mathbf{b}$ is a $n \times 1$ vector and c is a scalar.

The gradient of f is obtained as:

$$\nabla f(\mathbf{x}) = \mathbf{Q}\mathbf{x} + \mathbf{b} \tag{2.67}$$

Notice also that the matrix $\mathbf{Q}$ is the Hessian of f because it can be obtained as the second derivative of f with respect to the vector $\mathbf{x}$.

We shall determine the step s_k that minimizes $f(\mathbf{x}_k - s_k \nabla f(\mathbf{x}_k))$ for a

point $\mathbf{x}_k$.

$$\begin{aligned} f(\mathbf{x}_k - s_k \nabla f(\mathbf{x}_k)) &= \frac{1}{2}(\mathbf{x}_k - s_k \nabla f(\mathbf{x}_k))^T \mathbf{Q}(\mathbf{x}_k - s_k \nabla f(\mathbf{x}_k)) \\ &+ \mathbf{b}^T(\mathbf{x}_k - s_k \nabla f(\mathbf{x}_k)) + c \end{aligned} \tag{2.68}$$

When $\mathbf{x}_k$ is given, the function $f(\mathbf{x}_k - s_k \nabla f(\mathbf{x}_k))$ depends only on s_k and the derivative is:

$$\begin{aligned} \frac{df}{ds_k} &= -\nabla f(\mathbf{x}_k)^T \mathbf{Q}(\mathbf{x}_k - s_k \nabla f(\mathbf{x}_k)) - \mathbf{b}^T \nabla f(\mathbf{x}_k) \\ &= -\nabla f(\mathbf{x}_k)^T \mathbf{Q}\mathbf{x}_k + \nabla f(\mathbf{x}_k)^T \mathbf{Q} s_k \nabla f(\mathbf{x}_k) - \mathbf{b}^T \nabla f(\mathbf{x}_k) \end{aligned} \tag{2.69}$$

Because $\mathbf{b}^T \nabla f(\mathbf{x}_k) = \nabla f(\mathbf{x}_k)^T \mathbf{b}$ the relation (2.69) becomes:

$$\begin{aligned} \frac{df}{ds_k} &= -\nabla f(\mathbf{x}_k)^T \mathbf{Q}\mathbf{x}_k + \nabla f(\mathbf{x}_k)^T \mathbf{Q} s_k \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)^T \mathbf{b} \\ &= s_k \nabla f(\mathbf{x}_k)^T \mathbf{Q} \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)^T (\mathbf{Q}\mathbf{x}_k + \mathbf{b}) \\ &= s_k \nabla f(\mathbf{x}_k)^T \mathbf{Q} \nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k) \end{aligned} \tag{2.70}$$

where the last term in the right hand side was obtained using (2.67). If we set the derivative df/ds_k equal to zero, the optimal step is given by:

$$s_k = \frac{\nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k)}{\nabla f(\mathbf{x}_k)^T \mathbf{Q} \nabla f(\mathbf{x}_k)} \tag{2.71}$$

Thus, for quadratic functions, the method does not require a line search procedure and the steepest descent iteration is given by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left(\frac{\nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k)}{\nabla f(\mathbf{x}_k)^T \mathbf{Q} \nabla f(\mathbf{x}_k)} \right) \nabla f(\mathbf{x}_k) \tag{2.72}$$

The advantages of the steepest descent method include:

- It is simple, easy to understand and apply and iterations are fast because it requires only the gradient. The Hessian or its inverse are not needed.
- It is stable and improves the solution at each step. If the minimum

points exist, the method will locate them, but a small number of iterations is not guaranteed.

The method has some drawbacks:

- It may have a slow convergence, especially in cases when the curvature in different directions is very different.
- Line search may be time consuming and using a fixed step can give poor results.
- It is very much dependent on a good selection of the starting point and may be used when there is an indication of where the minimum is.

2.2.4 Conjugate gradient methods

The method of gradients often has slow convergence especially when the problem is poorly scaled i.e. when the contours are extremely elongated, due to the fact that it enforces successive orthogonal search directions (Snyman, 2005).

A basic characteristic of conjugate directions methods is to find the minimum of a quadratic function in a finite number of steps. These methods have been introduced for the solution of systems of linear equations and have later been extended to the solution of unconstrained optimization problems for non-quadratic functions (Astolfi, 2001).

Two vectors $\mathbf{d}_i,\ \mathbf{d}_j \in \mathbb{R}^n$ are said to be *orthogonal* if the scalar product $\mathbf{d}_i^T \mathbf{d}_j = 0$.

Two vectors $\mathbf{d}_i,\ \mathbf{d}_j \in \mathbb{R}^n$ are $\mathbf{Q}$-*orthogonal* or *conjugate* with respect to a symmetric positive definite matrix $\mathbf{Q}$ if:

$$\mathbf{d}_i^T \mathbf{Q} \mathbf{d}_j = 0 \tag{2.73}$$

This can be viewed as a generalization of orthogonality, for which $\mathbf{Q}$ is the unity matrix.

Consider the function $f(x_1, x_2)$ represented by the elliptical contour lines in Figure 2.14a) and the vectors d_0 and d_1. If the $x_1 - x_2$ space is "stretched" until the ellipses become circles and the vectors d_0 and d_1 will appear as perpendicular, they are Q-orthogonal.

In Figure 2.14a), the method of gradients will choose g_1 as a search direction, which is perpendicular to d_0. The method of conjugate gradients will choose d_1 that points to the minimum located in the center of the ellipse.

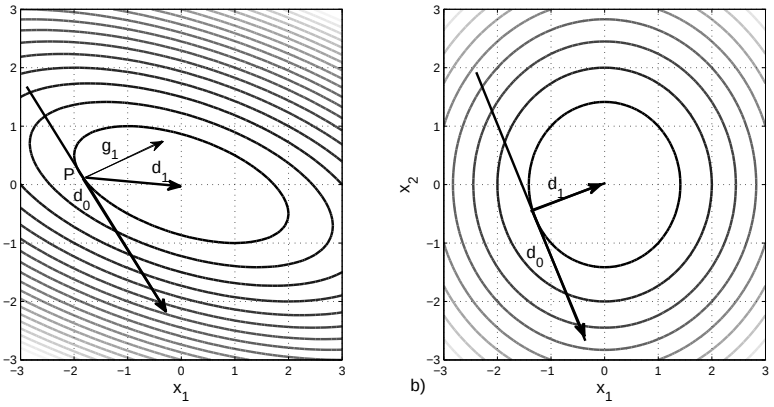


Figure 2.14: **Q**-orthogonal (a) and orthogonal vectors (b)

Detailed demonstrations of the conjugate gradient method is given in (Nocedal and Wright, 1999; Astolfi, 2001; Snyman, 2005). A summary and the algorithm of the method is given below.

The successive points of the iterative procedure, converging to a minimum of the function $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}$, are calculated according to the relation:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{d}_k \tag{2.74}$$

The step s_k is calculated by an exact or approximate line search method and $\mathbf{d}_k, k = \overline{0,n}$ are the mutually conjugate search directions.

Starting at an initial point $\mathbf{x}_0$, the first direction of move is minus gradient:

$$\mathbf{d}_0 = -\nabla f(\mathbf{x}_0) \tag{2.75}$$

The next directions are chosen so that

$$\mathbf{d}_{k+1} = -\nabla f(\mathbf{x}_{k+1}) + \beta_k \mathbf{d}_k \tag{2.76}$$

where the coefficient β_k is calculated by one of the formulas named Fletcher-Reeves (FR), Polak-Ribiere (PR), or Hestenes-Stiefel (HS) after their develop-

ers.

Fletcher-Reeves formula:

$$\beta_k = \frac{\nabla f(\mathbf{x}_{k+1})^T \nabla f(\mathbf{x}_{k+1})}{\nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k)} \tag{2.77}$$

Polak-Ribière formula:

$$\beta_k = \frac{\nabla f(\mathbf{x}_{k+1})^T \left(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)\right)}{\nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k)} \tag{2.78}$$

Hestenes-Stiefel formula:

$$\beta_k = \frac{\nabla f(\mathbf{x}_{k+1})^T \left(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)\right)}{\mathbf{d}_k^T \left(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)\right)} \tag{2.79}$$

Algorithm 9 describes the procedure.

Algorithm 9 Conjugate Gradient

Define function $f(\mathbf{x})$
Set $k = 0$ and select a starting point $\mathbf{x_0}$
Compute the first direction $\mathbf{d}_0 = -\nabla f(\mathbf{x}_0)$
repeat
Perform a one-dimensional line search in the direction $\mathbf{d_k}$, i.e.

$$\min_{s_k} F(s_k) = \min_{s_k} f(\mathbf{x_k} + s_k \mathbf{d_k})$$

Compute

$$\mathbf{x_{k+1}} = \mathbf{x_k} + s_k \mathbf{d_k}$$

Compute β_k using one of FR (2.77), PR (2.78) or HS (2.79) formulas.
Compute the new direction

$$\mathbf{d}_{k+1} = -\nabla f(\mathbf{x}_{k+1}) + \beta_k \mathbf{d}_k$$

Set $k \leftarrow k + 1$
until convergence criterion is satisfied.

- If the function f is quadratic, i.e. can be written in the form:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c \tag{2.80}$$

the step length can be determined from:

$$s_k = -\frac{\nabla f(\mathbf{x}_k)^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{Q} \mathbf{d}_k} \tag{2.81}$$

The demonstration is similar to the computations in (2.67)-(2.71).

- For quadratic functions the method converges in n iterations, where n is the number of variables.

- A practical issue regarding the implementation of this method relates to the accuracy of the line search method. Even small deviations can cause the search vectors to lose Q-orthogonality, and, in this case, a number of iterations greater than n will be necessary for locating the minimum.

- The loss of conjugacy that results from non-quadratic terms can be avoided if the procedure is modified and the algorithm restarted in cycles where the first search direction is the steepest descent and the initial point of each cycle is the value $\mathbf{x}_n$, obtained after n iterations, (Snyman, 2005).

Example 2.4 *We compare the performance of conjugate gradient algorithm to steepest descent when minimizing the function:*

$$f(x_1, x_2) = x_1^2 + 4x_2^2 + 2x_1x_2 \tag{2.82}$$

starting from an initial point $\mathbf{x}_0 = [-2.5 \;\; 0]^T$.

The function is quadratic and can be written in the form:

$$f(x_1, x_2) = \frac{1}{2}[x_1 \;\; x_2] \begin{bmatrix} 2 & 2 \\ 2 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{2}\mathbf{x}^T \mathbf{Q} \mathbf{x} \tag{2.83}$$

and the gradient vector is computed as:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 2x_1 + 2x_2 \\ 2x_1 + 8x_2 \end{bmatrix} \tag{2.84}$$

An initial direction is set as minus the gradient computed at $\mathbf{x}_0$*:*

$$\mathbf{d}_0 = -\nabla f(\mathbf{x}_0) = -\nabla f(-2.5, 0) = -\begin{bmatrix} 2\cdot(-2.5) + 2\cdot 0 \\ 2\cdot(-2.5) + 8\cdot 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \tag{2.85}$$

The optimal step length for quadratic functions, given by (2.81) is:

$$s_0 = -\frac{\nabla f(\mathbf{x}_0)^T \mathbf{d}_0}{\mathbf{d}_0^T \mathbf{Q} \mathbf{d}_0} = -\frac{\begin{bmatrix} -5 & -5 \end{bmatrix} \begin{bmatrix} 5 \\ 5 \end{bmatrix}}{\begin{bmatrix} 5 & 5 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 8 \end{bmatrix} \begin{bmatrix} 5 \\ 5 \end{bmatrix}} = 0.1429 \tag{2.86}$$

A new point is calculated using $\mathbf{d}_0$ *and* s_0*:*

$$\mathbf{x}_1 = \mathbf{x}_0 + s_0 \mathbf{d}_0 = \begin{bmatrix} -2.5 \\ 0 \end{bmatrix} + 0.1429 \begin{bmatrix} 5 \\ 5 \end{bmatrix} = \begin{bmatrix} -1.7857 \\ 0.7143 \end{bmatrix} \tag{2.87}$$

for which the gradient is

$$\nabla f(\mathbf{x}_1) = \begin{bmatrix} -2.1429 \\ 2.1429 \end{bmatrix} \tag{2.88}$$

We compute the next iterate using the Fletcher-Reeves formula to determine the coefficients β_k*:*

$$\beta_0 = \frac{\nabla f(\mathbf{x}_1)^T \nabla f(\mathbf{x}_1)}{\nabla f(\mathbf{x}_0)^T \nabla f(\mathbf{x}_0)} = \frac{\begin{bmatrix} -2.1429 & 2.1429 \end{bmatrix} \begin{bmatrix} -2.1429 \\ 2.1429 \end{bmatrix}}{\begin{bmatrix} -5 & -5 \end{bmatrix} \begin{bmatrix} -5 \\ -5 \end{bmatrix}} = 0.1837 \tag{2.89}$$

The new direction, $\mathbf{Q}$*-orthogonal or conjugate to* $\mathbf{d}_0$*, is given by:*

$$\mathbf{d}_1 = -\nabla f(\mathbf{x}_1) + \beta_0 \mathbf{d}_0 = -\begin{bmatrix} -2.1429 \\ 2.1429 \end{bmatrix} + 0.1837 \begin{bmatrix} 5 \\ 5 \end{bmatrix} = \begin{bmatrix} 3.0612 \\ -1.2245 \end{bmatrix}, \tag{2.90}$$

the optimal step length for this iteration is:

$$s_1 = -\frac{\nabla f(\mathbf{x}_1)^T \mathbf{d}_1}{\mathbf{d}_1^T \mathbf{Q} \mathbf{d}_1} = 0.5833 \tag{2.91}$$

and finally, the new point:

$$\mathbf{x}_2 = \mathbf{x}_1 + s_1 \mathbf{d}_1 = \begin{bmatrix} -1.7857 \\ 0.7143 \end{bmatrix} + 0.5833 \begin{bmatrix} 3.0612 \\ -1.2245 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.92}$$

is the minimizer of $f(x_1, x_2)$. It was obtained in $n = 2$ iterations. Figure 2.15 illustrates the contour lines of f and the location of the points computed at each step.

For comparison, the steepest descent method as presented in Algorithm 8 has been implemented for the minimization of the same function starting from the same initial point. The step length at each iteration has been computed using (2.71).

As seen in Figure 2.16, the search path is composed of successive orthogonal directions that slowly converge to the minimum located at $[0 \;\; 0]$.

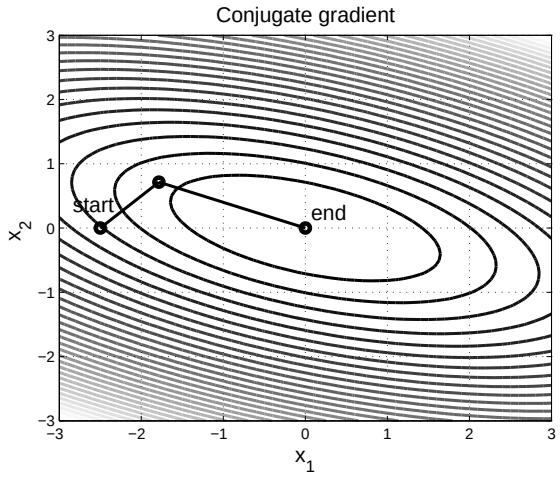


Figure 2.15: Conjugate gradient algorithm for minimization of $f(x_1, x_2)$

2.2.5 Quasi-Newton methods

An important aspect of gradient methods is that the second derivatives of the function to be minimized are not required. On the other hand, the Newton method assumes that the function is approximately quadratic around

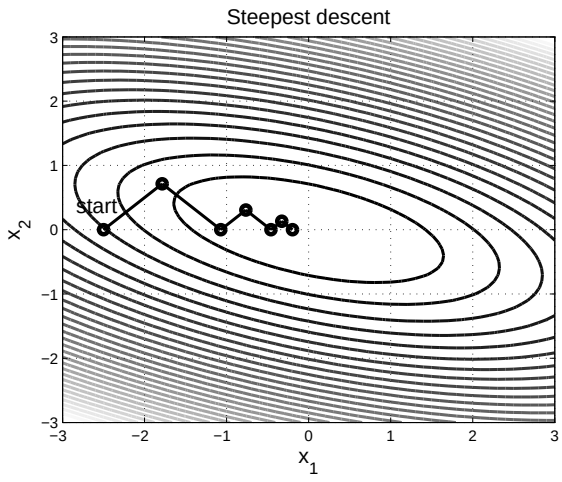


Figure 2.16: Steepest descent algorithm for minimization of $f(x_1, x_2)$

the minimum and uses the gradient and the Hessian to locate the stationary point.

Quasi-Newton methods also known as *variable metric methods* build up an approximation of the inverse Hessian using successive gradient vectors. We shall introduce the methods without demonstration. A detailed analysis of Quasi-Newton methods is given by Nocedal and Wright (1999) and Luenberger (2003).

The problem to be solved is the minimization of a function $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$. The gradient is denoted $\nabla f(\mathbf{x})$ and the Hessian is $\mathbf{H}(\mathbf{x})$.

Starting at an initial point $\mathbf{x}_0$, a general relation to calculate a new iterate for the algorithm of a modified Newton method is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - s_k \mathbf{B}_k \nabla f(\mathbf{x}_k) \tag{2.93}$$

- If $\mathbf{B}_k = \mathbf{I}$, the resulting method is steepest descent
- If $s_k = 1$ and $\mathbf{B}_k = \mathbf{H}^{-1}(\mathbf{x}_k)$, the relation (2.93) is the classical Newton method
- If $s_k = 1$ and $\mathbf{B}_k = \mathbf{H}^{-1}(\mathbf{x}_0)$, the relation (2.93) is the modified Newton method
- If the matrix $\mathbf{H}(\mathbf{x})$ and its inverse are difficult or impossible to compute,

quasi-Newton methods will determine an approximation of the inverse of the Hessian, as presented below.

Note that $\mathbf{d}_k = -\mathbf{B}_k \nabla f(\mathbf{x}_k)$ is a vector giving the search direction and s_k is the step length.

Quasi-Newton methods use the general algorithm (2.93), where the matrix $\mathbf{B}_k$ is initially chosen as the unit matrix ($\mathbf{B}_0 = \mathbf{I}$) and updated at each iteration according to the relation:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \mathbf{B}_k^u \tag{2.94}$$

The update matrix $\mathbf{B}_k^u$ can be calculated using one of the widely used formulas: Davidon-Fletcher-Powell or Broyden-Fletcher-Goldfarb-Shanno. Both of them require the value of the point and the gradient at the current and previous iteration:

$$\Delta \mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k \tag{2.95}$$

$$\Delta \mathbf{G}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) \tag{2.96}$$

The *Davidon-Fletcher-Powell (DFP)* update formula is:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \mathbf{B}_k^u = \mathbf{B}_k + \frac{\Delta \mathbf{x}_k \Delta \mathbf{x}_k^T}{\Delta \mathbf{x}_k^T \Delta \mathbf{G}_k} - \frac{\mathbf{B}_k \Delta \mathbf{G}_k (\mathbf{B}_k \Delta \mathbf{G}_k)^T}{\Delta \mathbf{G}_k^T \mathbf{B}_k \Delta \mathbf{G}_k} \tag{2.97}$$

The *Broyden-Fletcher-Goldfarb-Shanno (BFGS)* update relation is given by:

$$\mathbf{B}_{k+1} \quad = \mathbf{B}_k + \mathbf{B}_k^u = \mathbf{B}_k + \frac{\Delta \mathbf{G}_k \Delta \mathbf{G}_k^T}{\Delta \mathbf{G}_k^T \Delta \mathbf{x}_k} - \frac{\mathbf{B}_k \Delta \mathbf{x}_k (\mathbf{B}_k \Delta \mathbf{x}_k)^T}{\Delta \mathbf{x}_k^T \mathbf{B}_k \Delta \mathbf{x}_k} \tag{2.98}$$

The general procedure is described in Algorithm 10.

- Numerical experiments have shown that, although more complicated, BFGS's performance is superior to DFP. Hence, BFGS is often preferred over DFP.

- An important property of DFP and BFGS formulas is that if $\mathbf{B}_k$ is positive definite and the step size s_k is chosen to satisfy the Wolfe conditions then $\mathbf{B}_{k+1}$ is also positive definite.

Algorithm 10 Quasi-Newton

Define $f(\mathbf{x})$ and a tolerance ε
Compute the gradient $\nabla f(\mathbf{x})$
Choose an initial point $\mathbf{x_0}$
Set $k = 0$ and $\mathbf{B}_0 = \mathbf{I}$
repeat
Compute the search direction:

$$\mathbf{d}_k = -\mathbf{B}_k \nabla f(\mathbf{x}_k)$$

Compute the step length s_k by a line search procedure that minimizes the function in the direction $\mathbf{d}_k$:

$$\min_{s_k > 0} f(\mathbf{x}_k + s_k \mathbf{d}_k)$$

Compute a new point $\mathbf{x}_{k+1} = \mathbf{x}_k + s_k \mathbf{d}_k = \mathbf{x}_k - s_k \mathbf{B}_k \nabla f(\mathbf{x}_k)$
Compute the differences:

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

$$\Delta \mathbf{G}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$$

Update the matrix $\mathbf{B}_{k+1}$ according to the DFP (2.97) or BFGS (2.98) formulas:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \mathbf{B}_k^u$$

Set $k \leftarrow k + 1$
until one of the conditions (2.26), (2.27) or (2.28) are true

- For a quadratic objective function, DFP method simultaneously generates the directions of the conjugate gradient method while constructing the inverse Hessian $\mathbf{B}_k$.

2.2.6 Algorithms for minimization without derivatives

In situations when function derivatives are difficult or expensive to compute, the function is not very smooth or it is too noisy, methods which rely exclusively on the values of the objective function have to be employed. This type of algorithms do not calculate or estimate derivatives at any point and are identified as *zero-order methods* or *derivative-free* methods.

Although the number of methods that require only function evaluations

and no derivative computation is very large, we present only the widely used Nelder-Mead and Rosenbrock method as examples of algorithms in this category.

2.2.6.1 Nelder-Mead method

The *Nelder-Mead method* (or *downhill simplex method*, or *amoeba method*) solves unconstrained optimization problems of minimizing a nonlinear function $f : \mathbb{R}^n \to \mathbb{R}$. It does not require any derivative information, which makes it suitable for optimization of non-smooth and even discontinuous functions, (Press et al., 2007; Mathews and Fink, 2004). The first simplex-based direct search method was proposed by Spendley, Hext and Himsworth in 1962. In 1965, Nelder and Mead modified the original method and their approach gained popularity very quickly.

A *simplex* in $\mathbb{R}^n$ is defined as the convex hull of $n + 1$ vertices, $V_1, V_2, \ldots V_n \in \mathbb{R}^n$. For example, a simplex in $\mathbb{R}^2$ is a triangle, and a simplex in $\mathbb{R}^3$ is a tetrahedron (Figures 2.17, 2.18). The method compares the objective function values at the $N + 1$ vertices and moves towards the optimal point iteratively.

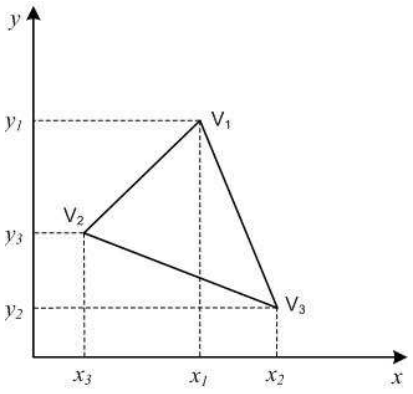


Figure 2.17: A simplex in $\mathbb{R}^2$

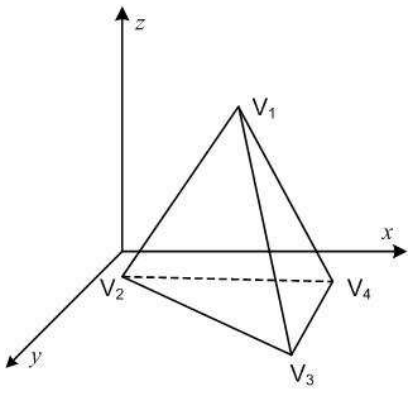


Figure 2.18: A simplex in $\mathbb{R}^3$

The algorithm will be presented as given in (Mathews and Fink, 2004) for a function of two variables, $f : \mathbb{R}^2 \to \mathbb{R}$, $f = f(x, y)$.

The initial triangle. Start with a triangle $\Delta V_1 V_2 V_3$, (Figure 2.17). The function $f(x, y)$ is evaluated at each of the three vertices having the coor-

dinates: $V_1(x_1, y_1)$, $V_2(x_2, y_2)$, $V_3(x_3, y_3)$. The subscripts are then reordered so that $f(x_1, y_1) < f(x_2, y_2) < f(x_3, y_3)$. The vertex where the function has the best value (the smallest out of three in case of a minimization problem) is denoted by B, the second best is denoted by G and the one where the function has its worst value by W. After reordering, these points will have the coordinates:

$$B = (x_1, y_1), \quad G = (x_2, y_2), \quad W = (x_3, y_3) \tag{2.99}$$

Figure 2.19 shows an example of an initial triangle and the notations B, G and W according to the function value at the vertices.

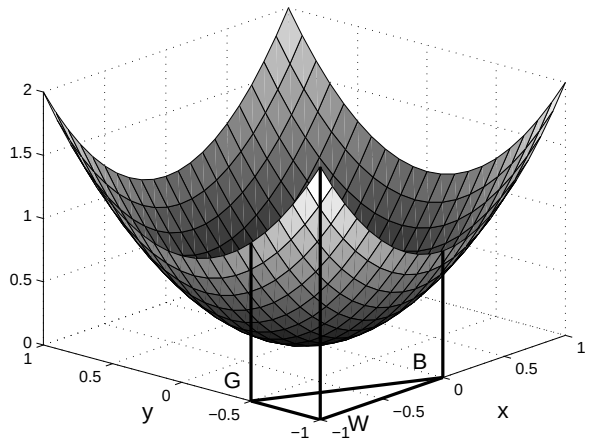


Figure 2.19: Example of an initial triangle and the vertices B, G, W

Midpoint of the BG side. The procedure uses the midpoint M, of the line segment joining B and G (Figure 2.20). Its coordinates are obtained as:

$$x_M = \frac{x_1 + x_2}{2}, \quad y_M = \frac{y_1 + y_2}{2} \tag{2.100}$$

or, using vector notation:

$$M = \frac{B + G}{2} = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right) \tag{2.101}$$

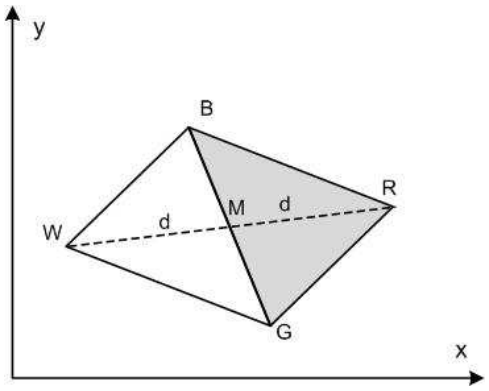


Figure 2.20: The triangle BGW , midpoint M and reflected point R

Reflection. If we move from vertex W towards B or G, along the sides, the function decreases (because B corresponds to the smallest value), or, smaller values of f lie away from W, on the opposite side of the line between B and G. A point R is determined by "reflecting" the triangle through the BG side. It is located at the same distance (d) from M, as W is to M, along the extended line that starts at W and passes through the midpoint of BG, (Figure 2.20). The vector formula for the coordinates of R is:

$$R = M + (M - W) = 2M - W \tag{2.102}$$

Expansion. If the function value at R is smaller than the function value at W then we have moved in the correct direction toward the minimum. To accelerate the search, we extend the line segment through M and R to the point E with an additional distance d, (Figure 2.21), and we obtain an expanded triangle ΔBGE. If the function value at E is less than the function value at R, then we have found a better vertex than R. The vector formula for E is:

$$E = R + (R - M) = 2R - M \tag{2.103}$$

Contraction. If the function values at R and W are the same, another point must be tested. Perhaps the function is smaller at M but we cannot

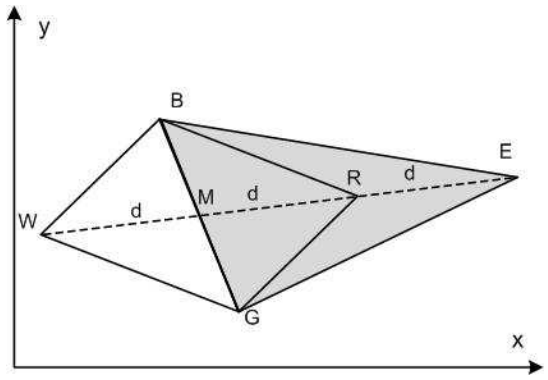


Figure 2.21: The triangle BGW and point R and extended point E

replace W with M because we need a triangle. Consider the two midpoints $C1$ and $C2$ of the line segments WM and MR respectively (Figure 2.22). The point with the smaller function value is called C and the new triangle is ΔBGC.

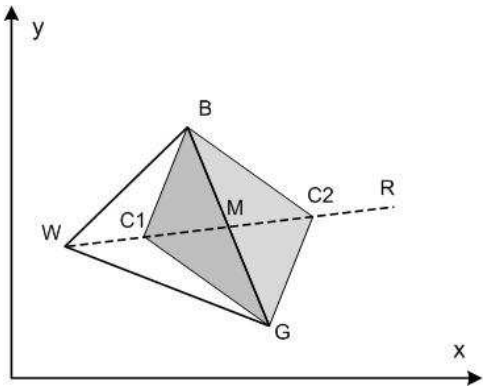


Figure 2.22: The contraction points

The points $C1$ and $C2$ have the coordinates:

$$C1 = W + \frac{M - W}{2} = \frac{W + M}{2}, \quad C2 = R - \frac{R - M}{2} = \frac{R + M}{2} \tag{2.104}$$

Shrinking. If the function value at C is not less than the function value at W, the points G and W must be shrunk toward B. The point G is replaced by M and W is replaced by S, which is the midpoint of the line segment

joining B and W (Figure 2.23). Similar to M, the midpoint S can be calculated from:

$$S = \frac{B + W}{2} \tag{2.105}$$

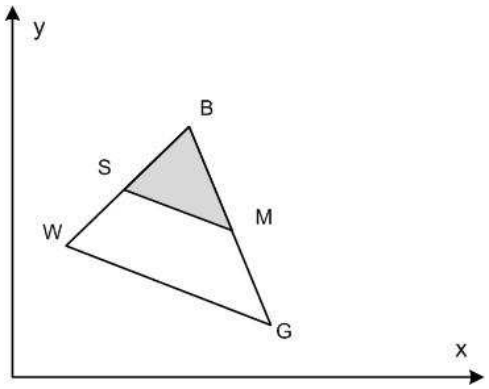


Figure 2.23: Shrinking toward B

The algorithm must terminate in a finite number of steps or iterations. The stop criterion can be one or all of the following:

- The simplex becomes sufficiently small, or the vertices are within a given tolerance ε
- The function does not change for successive iterations.
- The number of iterations or function evaluations exceeds a maximum value allowed.

For the two-dimensional case, the details are summarized in Algorithm 11, (Mathews and Fink, 2004).

Example 2.5 *Use the Nelder-Mead algorithm to find the minimum of:*

$$f(x, y) = (x - 10)^2 + (y - 10)^2 \tag{2.106}$$

We calculate the first steps and illustrate them graphically.

Algorithm 11 Nelder-Mead method

Define function $f(x, y)$ and tolerance ε
Select 3 initial vertices $V_1,\ V_2,\ V_3$
while stop criterion not fulfilled **do**
  Compute $f(V_1)$, $f(V_2)$, $f(V_3)$ and set the labels B, G, W
  Compute $M = (B + G)/2$ and $R = 2M - W$
  if $f(R) < f(W)$ **then**
    Compute E=2R-M, f(E)
    if $f(E) < f(R)$ **then**
      Replace W with E
    else
      Replace W with R
    end if
  else
    Compute C1 and C2 and choose C
    if $f(C) < f(W)$ **then**
      Replace W with C
    else
      Compute S and f(S)
      Replace W with S
      Replace G with M
    end if
  end if
end while

First iteration *Start with three vertices conveniently chosen at the origin and on the axes (Figure 2.24):*

$$V_1 = (0,0), \quad V_2 = (2,0), \quad V_3 = (0,6) \tag{2.107}$$

and calculate the function values $f(x,y)$ at these points:

$$f(0,0) = 200, \quad f(2,0) = 164, \quad f(0,6) = 116 \tag{2.108}$$

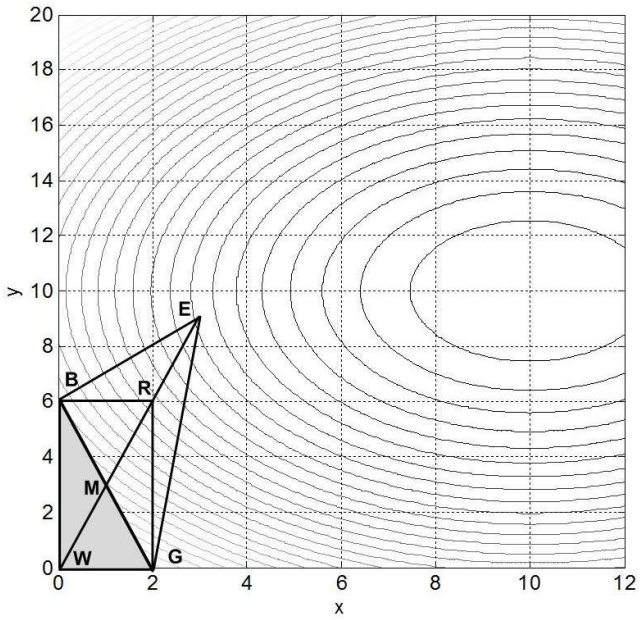


Figure 2.24: The first sequence of triangles

Since $f(0,6) < f(2,0) < f(0,0)$, the vertices will be assigned the names:

$$B = (0,6), \quad G = (2,0), \quad W = (0,0) \tag{2.109}$$

The midpoint of BG is:

$$M = \frac{B+G}{2} = (1,3) \tag{2.110}$$

and the reflected point:

$$R = 2M - W = (2, 6) \tag{2.111}$$

The function value at R, $f(R) = f(2, 6) = 80 < f(G) < f(B)$ and the extended point E will be calculated from:

$$E = 2R - M = (3, 9) \tag{2.112}$$

The new triangle is ΔBGE.

Second iteration *Since the function value at E is $f(3, 9) = 50 < f(G) < f(B)$, the vertices labels for the current stage are (Figure 2.25):*

$$B = (3, 9), \;\; G = (0, 6), \;\; W = (2, 0) \tag{2.113}$$

The new midpoint of BG and the reflected point R are:

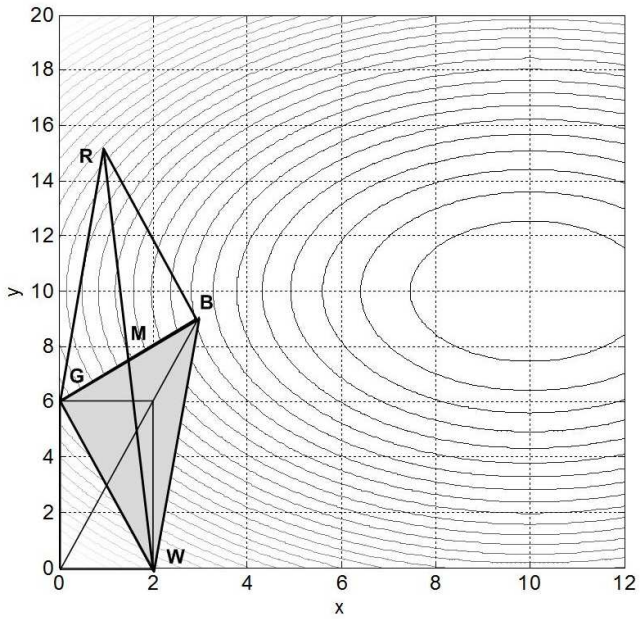


Figure 2.25: The second sequence of triangles

$$M = \frac{B+G}{2} = (1.5, 7.5), \quad R = 2M - W = (1, 15) \tag{2.114}$$

The value of f at the reflected point is: $f(1,15) = 106$. It is less than $f(G) = f(0,6) = 116$ but greater than $f(B) = f(3,9) = 50$. Therefore, the extended point will not be calculated and the new triangle is ΔRGB.

Third iteration *Because $f(3,9) < f(1,15) < f(0,6)$, the new labels are (Figure 2.26):*

$$B = (3,9), \quad G = (1,15), \quad W = (0,6) \tag{2.115}$$

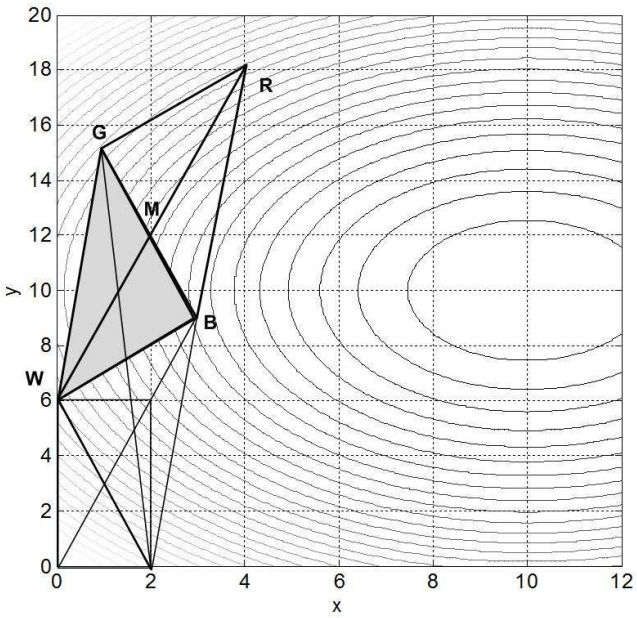


Figure 2.26: The third sequence of triangles

A new midpoint and a reflected point are calculated:

$$M = \frac{B+G}{2} = (2, 12), \quad R = 2M - W = (4, 18) \tag{2.116}$$

and the function takes at R the value $f(R) = f(4,18) = 100$. This is again less than the value at G but it is not less than the value at B, thus the point R will be a vertex of the next triangle ΔRGB.

Fourth iteration *Because $f(3,9) < f(4,18) < f(1,15)$, the new labels are (Fig-*

ure 2.27):

$$B = (3, 9), \quad G = (4, 18), \quad W = (1, 15) \tag{2.117}$$

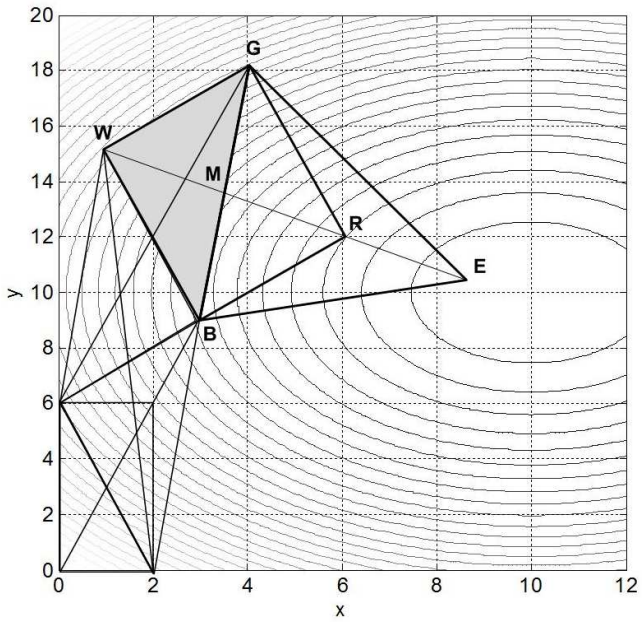


Figure 2.27: The fourth sequence of triangles

The midpoint of BG and R result as:

$$M = \frac{B+G}{2} = (3.5, 13.5), \quad R = 2M - W = (6, 12) \tag{2.118}$$

Because $f(R) = f(6, 12) = 20 < f(B)$, we shall build the extended point E:

$$E = 2R - M = (8.5, 10.5) \tag{2.119}$$

The function takes here the value $f(E) = f(8.5, 10.5) = 2.5$, thus the new triangle is ΔGBE.

The calculations continue until the function values at the vertices have almost equal values (with some allowable tolerance ε). It will still take a large number of steps until the procedure will reach the minimum point $(10, 10)$ where the function

value is 0*, but the steps performed above show that the function is decreasing at each iteration.*

2.2.6.2 Rosenbrock method

The main drawback of the Nelder-Mead method is that general convergence properties have not been proven. An alternative for a derivative-free algorithm is the method of Rosenbrock whose global convergence to a local optima is assured.

This zero-order method solves the unconstrained optimization problem of minimizing a nonlinear function $f : \mathbb{R}^n \to \mathbb{R}$, (Rosenbrock, 1960). Starting at an initial point $\mathbf{x}_0$, the method explores locally along n orthogonal directions seeking for a better solution. The first set of directions is usually composed of the base vectors of an n-dimensional coordinate system. During the iterative process, this set is recalculated such that the first vector to point into the direction of the maximum function decrease.

Example 2.6 *Consider the orthogonal coordinate system $x_1 - x_2$ shown in Figure 2.28. A set of normalized vectors in the direction of the axes is $(\mathbf{d}_1, \mathbf{d}_2)$ given*

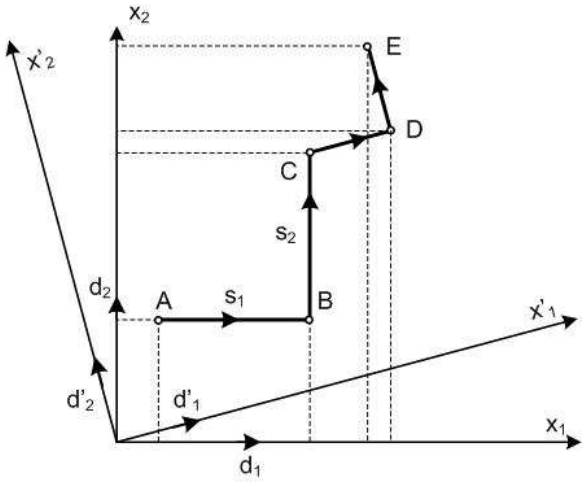


Figure 2.28: Rotation of a coordinate system

by:

$$\mathbf{d}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{d}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.120}$$

Consider an initial point A having the coordinates given by the vector $\mathbf{x}_A = [x_{1A}\ x_{2A}]^T$. We move to point B that is at a distance s_1 in the direction of the x_1-axis, this being the direction d_1. The coordinates of B are calculated as:

$$\begin{bmatrix} x_{1B} \\ x_{2B} \end{bmatrix} = \begin{bmatrix} x_{1A} \\ x_{1A} \end{bmatrix} + s_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.121}$$

or, in a vector form:

$$\mathbf{x}_B = \mathbf{x}_A + s_1 \mathbf{d}_1 \tag{2.122}$$

From B we move a distance s_2 in the direction of $\mathbf{d}_2$ to obtain a point C:

$$\mathbf{x}_C = \mathbf{x}_B + s_2 \mathbf{d}_2 \tag{2.123}$$

The coordinate system is now rotated and the new orthogonal directions of the axes are $\mathbf{d'}_1$, $\mathbf{d'}_2$. If we move further on to the point D along $\mathbf{d'}_1$ and then to E along $\mathbf{d'}_2$, with the distances s'_1 and s'_2, the coordinates are given by similar relations:

$$\mathbf{x}_D = \mathbf{x}_C + s'_1 \mathbf{d'}_1 \tag{2.124}$$

$$\mathbf{x}_E = \mathbf{x}_D + s'_2 \mathbf{d'}_2 \tag{2.125}$$

The initial data for the Rosenbrock method include, (Lucaci and Agachi, 2002):

- an initial vector $\mathbf{x}_0 = [x_{10}\ x_{20}\ \ldots x_{n0}]^T$
- a set of initial orthogonal normalized vectors $\mathbf{d}_{10}$, $\mathbf{d}_{20}$, ..., $\mathbf{d}_{n0}$. Usually they are taken as the columns of an n-th order unit matrix.
- an initial vector of step lengths along each direction $\mathbf{s} = [s_{10}\ s_{20}\ \ldots s_{n0}]^T$
- the accuracy tolerance (ε) of the solution

The problems that have to be met in the implementation of the method are given below, (Rosenbrock, 1960):

Step length. The initial step length will be changed during the algorithm. If a move was successful in one direction $\mathbf{d}_i$, i.e., the function decreases or $f(\mathbf{x} + s_i \mathbf{d}_i) < f(\mathbf{x})$, the step length will be multiplied by a factor

$\alpha > 1$. If the move failed, the step was multiplied by $-\beta$ where $0 < \beta < 1$. Thus, at the next iteration, the move will be performed on the direction opposite to $\mathbf{d}_i$ and with a smaller step length. If the step size was initially so small that it made no change in the value of f, it would be increased on the next attempt. Each such attempt will be called a "trial".

Directions. An important issue related to this method is to decide when and how to change the set of directions. The exploration along the axes followed by the calculation of a new point in case of a successful result is continued until at least one trial had been successful in each direction and one had failed.

In this case a new set of directions $\mathbf{d'}_k,\ k = \overline{1,n}$ is calculated using the following procedure:

Let D_i be the algebraic sum of all successful step lengths in the direction $\mathbf{d}_i$ and compute the following vectors:

$$\mathbf{A}_1 = D_1\mathbf{d}_1 + D_2\mathbf{d}_2 + \ldots + D_n\mathbf{d}_n \tag{2.126}$$

$$\mathbf{A}_2 = D_2\mathbf{d}_2 + \ldots + D_n\mathbf{d}_n \tag{2.127}$$

$$\ldots$$

$$\mathbf{A}_n = D_n\mathbf{d}_n \tag{2.128}$$

$$\tag{2.129}$$

Thus $\mathbf{A}_i$ is the vector joining the initial and final points obtained by use of the vectors $\mathbf{d}_k$, $k = \overline{i,n}$. Orthogonal unit vectors of the new coordinate system ($\mathbf{d'}_1$, $\mathbf{d'}_2$, ..., $\mathbf{d'}_n$) are obtained from $\mathbf{A}_i$ by the Gram-Schmidt orthogonalization procedure:

$$\mathbf{B}_1 = \mathbf{A}_1, \quad \mathbf{d'}_1 = \frac{\mathbf{B}_1}{\|\mathbf{B}_1\|} \tag{2.130}$$

$$\mathbf{B}_2 = \mathbf{A}_2 - \frac{\mathbf{A}_2^T\mathbf{B}_1}{\|\mathbf{B}_1\|^2}\mathbf{B}_1, \quad \mathbf{d'}_2 = \frac{\mathbf{B}_2}{\|\mathbf{B}_2\|} \tag{2.131}$$

$$\ldots$$

$$\mathbf{B}_n = \mathbf{A}_n - \sum_{j=1}^{n-1}\frac{\mathbf{A}_n^T\mathbf{B}_j}{\|\mathbf{B}_j\|^2}\mathbf{B}_j, \quad \mathbf{d'}_n = \frac{\mathbf{B}_n}{\|\mathbf{B}_n\|} \tag{2.132}$$

The new set of directions was calculated such that $\mathbf{d'}_1$ lies along the direction of fastest decrease, $\mathbf{d'}_2$ along the best direction which can be found normal to $\mathbf{d'}_1$ and so on.

For the practical implementation we have to consider the following issues:

- The algorithm terminates when a convergence criterion is fulfilled: the distance between two consecutive points, $\mathbf{x}_{k+1}$ and $\mathbf{x}_k$ is smaller than a given tolerance ε or the function value is not improved more than ε:

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \leq \varepsilon, \quad \text{or} \quad f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) \leq \varepsilon \tag{2.133}$$

- The set of directions will be re-calculated when there is a success and a failure on each direction, which is similar to an oscillation. During the exploration on a direction $\mathbf{d}_i$ a variable $success(i)$ can be set equal to 1 to mark a successful move and a variable $fail(i)$ can be set to 1 to mark a failure. If all variables of this type are 1 the new directions have to be determined.

The method is summarized in Algorithm 12.

Example 2.7 *Using the Rosenbrock method minimize the 'banana' function (or Rosenbrock function):*

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{2.134}$$

The algorithm was implemented with the following initial conditions:

- *Initial point* $\mathbf{x}_0 = [0 \;\; 0]^T$
- *Initial step lengths* $\mathbf{s} = [0.5 \;\; 0.5]^T$
- *The factors* $\alpha = 3$ *and* $\beta = -0.5$

and the global minimum at $(1, \; 1)$ *was obtained with an accuracy* $\varepsilon = 10^{-6}$*. The result is shown in Figure 2.29.*

Algorithm 12 Rosenbrock method

Define function $f(\mathbf{x})$
Select an initial point $\mathbf{x}_0$
Initialize n orthogonal directions $\mathbf{d}_{10}$, $\mathbf{d}_{20}$, ..., $\mathbf{d}_{n0}$ as the columns of a unit matrix
Initialize the vector of step lengths $\mathbf{s} = [s_{10}\ s_{20}\ \dots s_{n0}]^T$
Select factors $\alpha > 1$ and $-1 < \beta < 0$
Select accuracy tolerance ε
Set $k = 1$
while stop criterion not fulfilled **do**
  Initialize the vector of successful steps for all directions $D = [0\ 0\ \dots 0]$
  Initialize a flag for oscillation: $oscillation = false$
  Initialize a vector to store successes on each direction: $success = [0\ 0\ \dots 0]$
  Initialize a vector to store failures on each direction: $fail = [0\ 0\ \dots 0]$
  while $oscillation = false$ and stop criterion not fulfilled **do**
    for i=1:n **do**
      if $f(\mathbf{x}_k + s_i\mathbf{d}_i) < f(\mathbf{x}_k)$ **then**
        Move to the next point. Compute $\mathbf{x}_{k+1} = \mathbf{x}_k + s_i\mathbf{d}_i$
        Set $k \leftarrow k + 1$
        Mark a success on direction $\mathbf{d}_i$. Set $success(i) = 1$
        Add the step length to D_i. Set $D_i \leftarrow D_i + s_i$
        Increase the step length. Set $s_i \leftarrow s_i \cdot \alpha$
      else
        Mark a failure on direction $\mathbf{d}_i$. Set $fail(i) = 1$
        Decrease the step length. Set $s_i \leftarrow s_i \cdot \beta$
      end if
    end for
    if all $sucess(i) = 1$ and all $fail(i) = 1$ **then**
      Set $oscillation = true$
    end if
  end while
  Compute the new directions using the Gram-Schmidt procedure from vectors D and $\mathbf{d}_j,\ j = \overline{1, n}$
end while

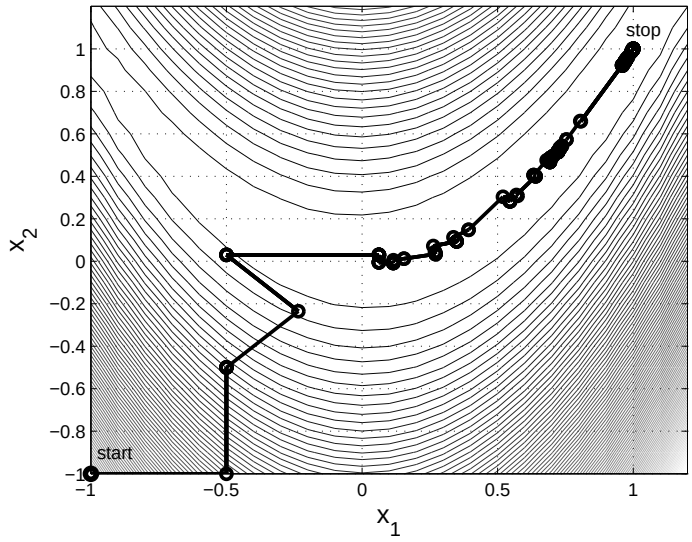


Figure 2.29: Minimization of Rosenbrock banana function