2 Single Variable Minimization and Line Searches

2.1 Motivation

Consider a scalar function, f, that depends on a single independent variable, x. Suppose we want to find the value of x where $f(x)$ is a minimum value. Furthermore, we want to do with,

- Low computational cost (few iterations and low cost per iteration)
- Low memory requirements
- Low failure rate

Often the computational effort is dominated by the computation of f and its derivatives so some of the requirements can be translated into: evaluate $f(x)$ and $\mathrm{d}f/\mathrm{d}x$ as few times as possible.

2.2 Optimality Conditions

Types of of minima:

1. Strong local minimum
2. Weak local minimum
3. Global minimum

Taylor's theorem. If $f(x)$ is n times differentiable, then $\theta \quad (0 \leq \theta \leq 1)$ exists such that

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \cdots + \frac{1}{(n-1)!}h^{n-1}f^{n-1}(x) + \underbrace{\frac{1}{n!}h^n f^n(x+\theta h)}_{\mathcal{O}(h^n)} \tag{2.1}$$

Assume f is twice-continuously differentiable and a minimum of f exists at x^*. Using a Taylor series expansion of f about x^*,

$$f(x^* + \varepsilon) = f(x^*) + \varepsilon f'(x^*) + \frac{1}{2}\varepsilon^2 f''(x^* + \theta\varepsilon) \tag{2.2}$$

(i.e. use $n = 2$ in Taylor's theorem above)

For a local minimum at x^*, we require that $f(x^* + \varepsilon) \geq f(x^*)$ for a range $-\delta \leq \varepsilon \leq \delta$, where δ is an arbitrary positive number.

Given this definition, and the Taylor series expansion (2.2), for $f(x^*)$ to be a local minimum, we need $\varepsilon f'(x^*) + \frac{1}{2}\varepsilon^2 f''(x^* + \theta\varepsilon) \geq 0$.

Note that for any finite values of $f'(x^*)$ and $f''(x^*)$, we can always chose a ε small enough such that $\varepsilon f'(x^*) \gg \frac{1}{2}\varepsilon^2 f''(x^*)$. Therefore, we should start by considering the first derivative term.

For $\varepsilon f'(x^*)$ to be non-negative, then $f'(x^*) = 0$, because the sign of ε is arbitrary. This is the *first-order optimality condition*.

Because the first derivative term is zero, we have to consider the second derivative term. This term must now must be non-negative for a local minimum at x^*. Since ε^2 is always positive, then we require that $f''(x^*) \geq 0$. This is the *second-order optimality condition*. Terms higher than second order can always be made smaller than the second-order term by choosing a small enough ε.

Necessary conditions (local minimum):

$$f'(x^*) = 0; \qquad f''(x^*) \geq 0 \tag{2.3}$$

Sufficient conditions (strong minimum):

$$f'(x^*) = 0; \qquad f''(x^*) > 0 \tag{2.4}$$

The optimality conditions can be used to:

1. Verify that a point is a minimum (sufficient conditions).
2. Realize that a point is not a minimum (necessary conditions).
3. The conditions can be solved to find a minimum.

Gradient-based minimization methods find a local minima by finding points that satisfy the optimality conditions.

2.3 Root Finding and Function Minimization

Solving the first-order optimality conditions, that is, finding x^* such that $f'(x^*) = 0$, is equivalent to finding the roots of the first derivative of the function to be minimized. Therefore, root finding methods can be used to find stationary points and are useful in function minimization.

Using machine precision, it is not possible find the exact zero, so we will be satisfied with finding an x^* that belongs to an interval $[a, b]$ such that

$$f(a)f(b) < 0 \quad \text{and} \quad |a - b| < \delta \tag{2.5}$$

where δ is a "small" tolerance. This tolerance might be dictated by the machine representation (using double precision this is usually 1×10^{-16}), the precision of the function evaluation, or a limit on the number of iterations we want to perform with the root finding algorithm.

The *rate of convergence* is a measure of how fast an iterative method converges to the numerical solution. An iterative method is said to converge with order r when r is the largest number such that

$$0 \leq \lim_{k \to \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^r} < \infty. \tag{2.6}$$

where k is iteration number.

This is to say that the above limit must be a positive constant. This constant is the *asymptotic error constant*.

$$\lim_{k \to \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^r} = \gamma \tag{2.7}$$

If the limit is zero when $r = 1$, we have a special case called *superlinear convergence*.

When solving real problems, we don't know the exact x^* in advance, but it is useful to plot $\|x_{k+1} - x_k\|$ and $\|g_k\|$ in a log-axis plot versus k.

Example 2.1: Rate of Convergence
Consider the sequence

$$x_k = c^{2^k}, \tag{2.8}$$

where $0 \leq c < 1$. Each term in this sequence is the square of the previous element, and the limiting value is zero. We can show that

$$\frac{|x_{k+1} - 0|}{|x_k - 0|^2} = c \tag{2.9}$$

Thus $r = 2$ and the sequence converges quadratically. This means that the that the number of correct figures roughly doubles with each iteration.

2.3.1 Method of Bisection

In this method for finding the zero of a function f, we first establish a bracket $[x_1, x_2]$ for which the function values $f_1 = f(x_1)$ and $f_2 = f(x_2)$ have opposite sign.

The function is then evaluated at the midpoint, $x = \frac{1}{2}(x_1 + x_2)$. If $f(x)$ and f_1 have opposite signs, then x and f become x_2 and f_2 respectively. On the other hand, if $f(x)$ and f_1 have the same sign, then x and f become x_1 and f_1.

This process is then repeated until the desired accuracy is obtained.

Bisection is guaranteed to find the zero to a specified tolerance δ (if the function evaluation is precise enough) in about $\log_2(x_1 - x_2)/\delta$ evaluations, if $[x_1, x_2]$ is the initial interval.

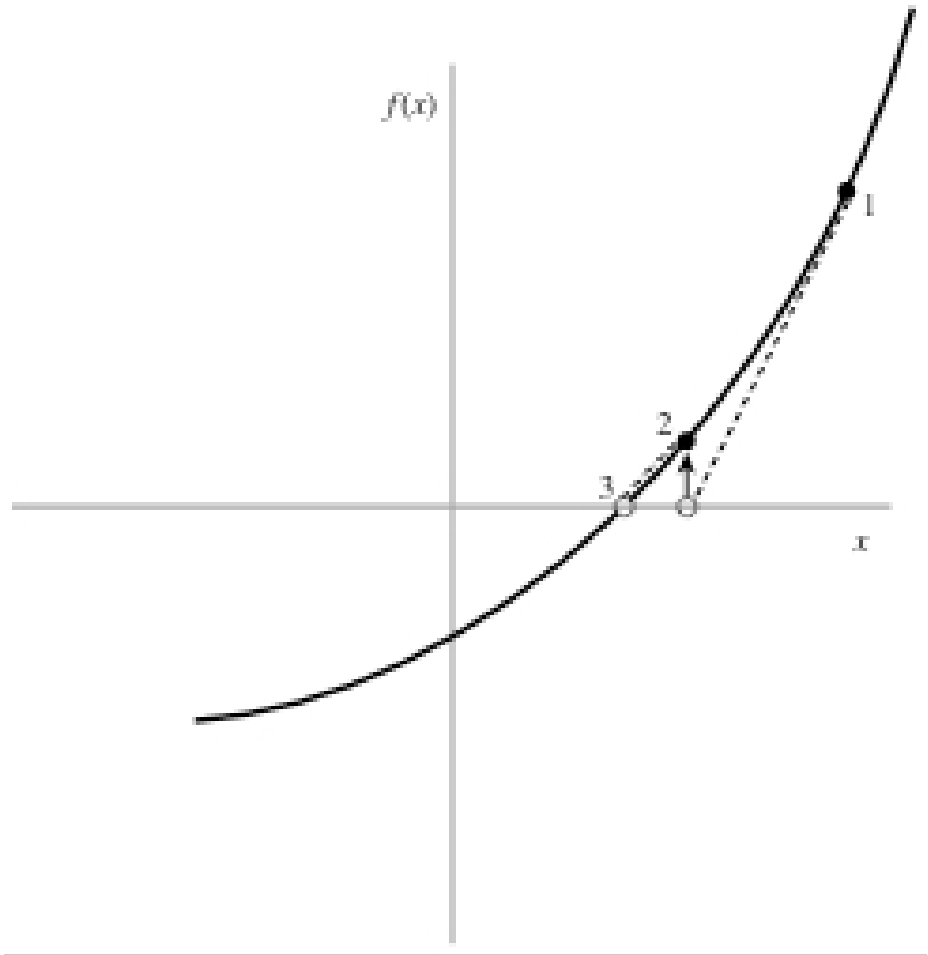
To find the minimum of a function using bisection, we would evaluate the derivative of f at each iteration, instead of the value.

The bisection algorithm, for example, exhibits a linear rate of convergence ($r = 1$) and the asymptotic error is $1/2$.

2.3.2 Newton's Method

Newton's method for finding a zero can be derived from the Taylor's series expansion by setting the function to zero and ignoring the terms of order higher than two.

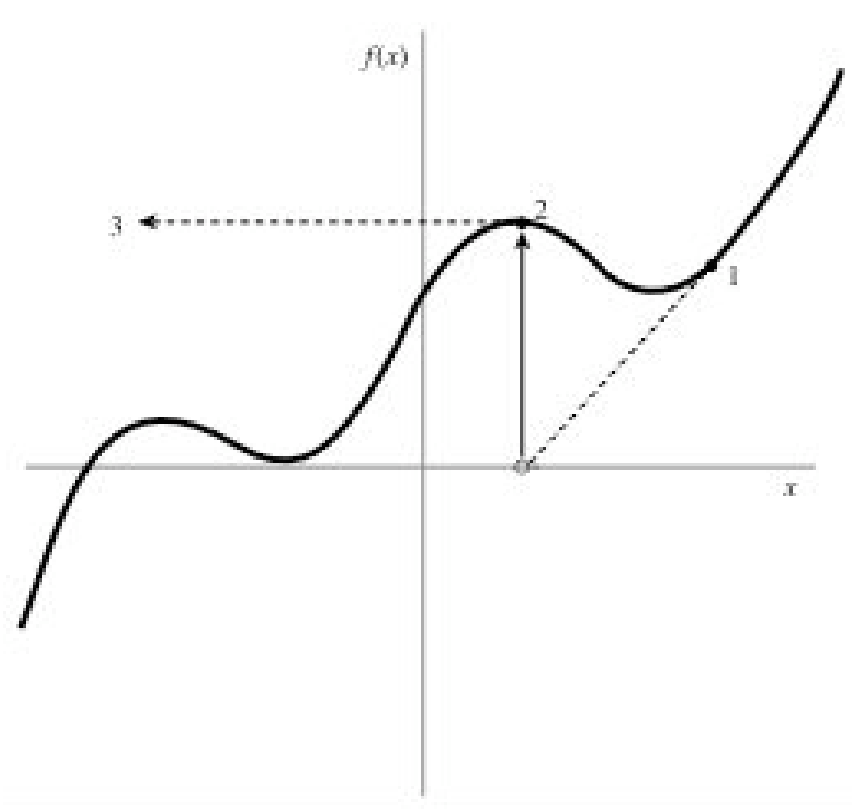
$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \qquad (2.10)$$

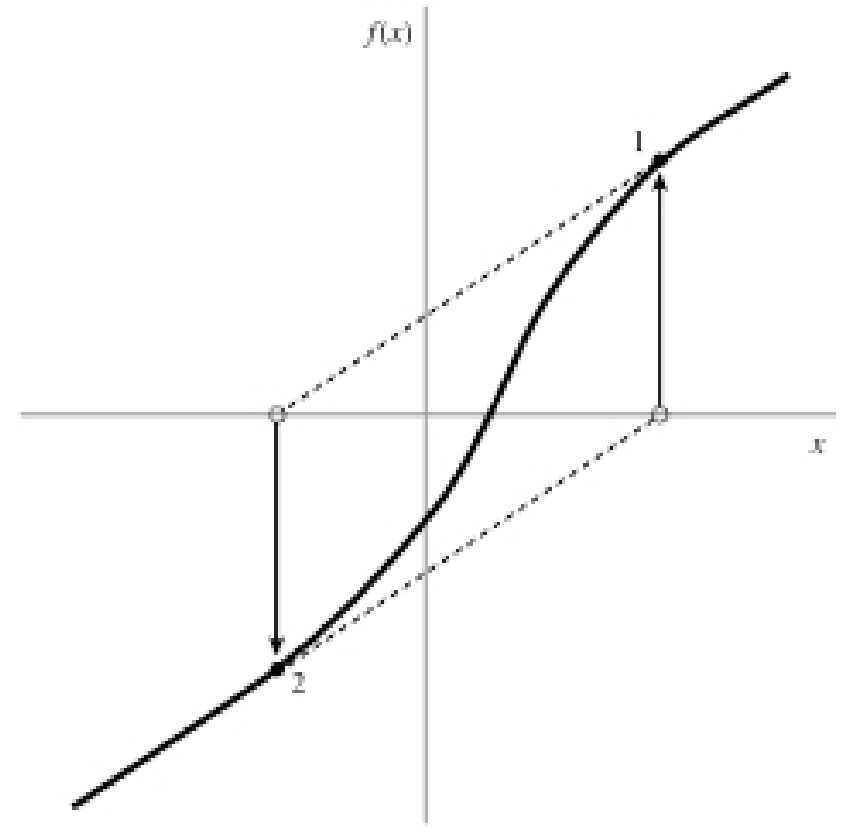


This iterative procedure converges quadratically, so

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = \text{const.} \qquad (2.11)$$

While having quadratic converge is a great property, Newton's method is not guaranteed to converge, and only works under certain conditions.





To minimize a function using Newton's method, we simply substitute the function for its first derivative and the first derivative by the second derivative,

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}. \tag{2.12}$$

Example 2.2: Function Minimization Using Newton's Method

Consider the following single-variable optimization problem

$$\begin{aligned} \text{minimize} \quad & f(x) = (x-3)x^3(x-6)^4 \\ \text{w.r.t.} \quad & x \end{aligned}$$

Solve this using Newton's method.

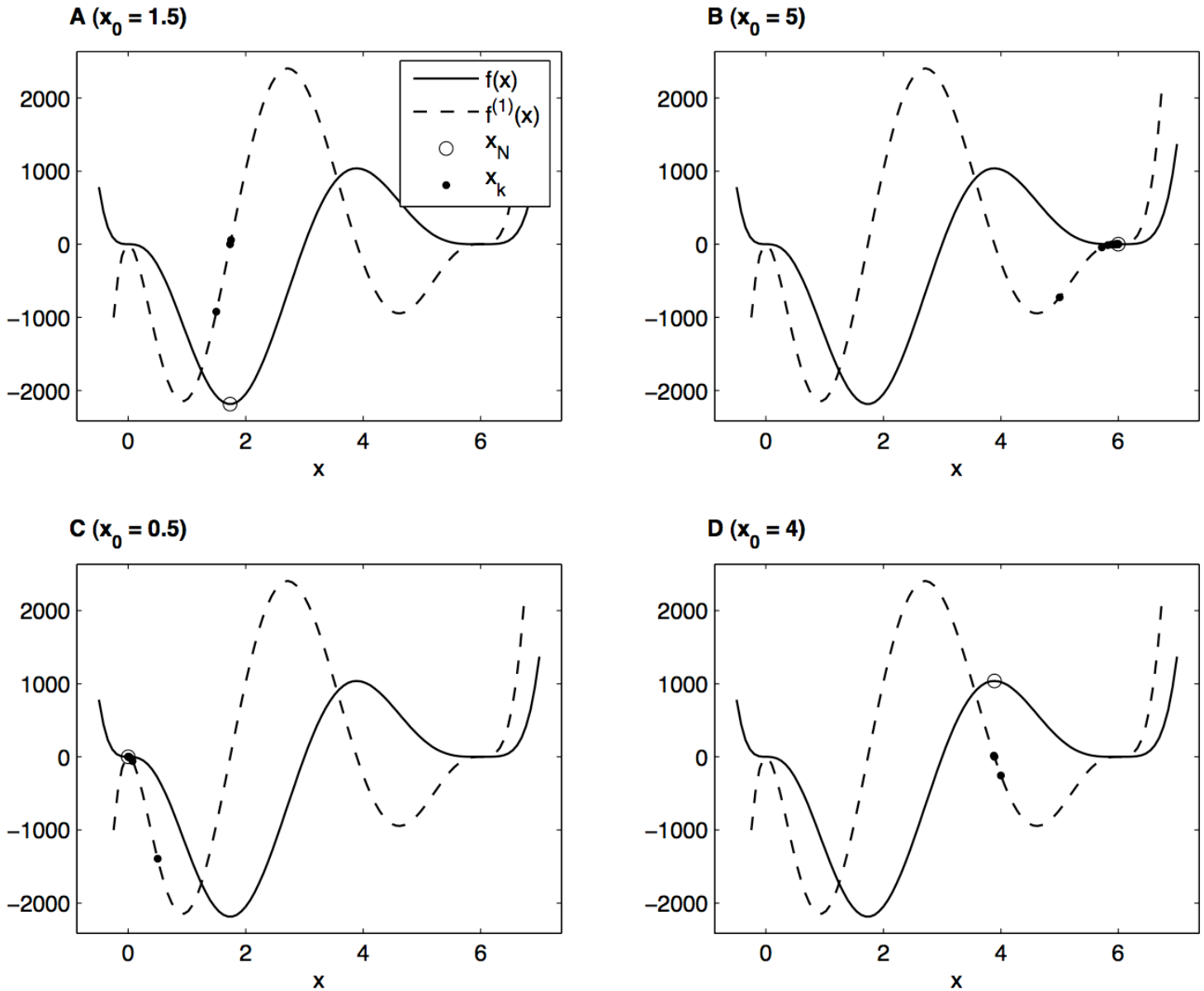


Figure 2.1: Newton's method with several different initial guesses. The x_k are Newton iterates. x_N is the converged solution.

2.3.3 Secant Method

Newton's method requires the first derivative for each iteration. In some practical application, it might not be possible to obtain this derivative analytically or it might just be troublesome.

If we use a forward-difference approximation for $f'(x_k)$ in Newton's method we obtain

$$x_{k+1} = x_k - f(x_k)\left(\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}\right). \tag{2.13}$$

which is the secant method ("the poor-man's Newton method").

Under favorable conditions, this method has superlinear convergence ($1 < r < 2$), with $r \approx 1.6180$.

2.3.4 Fibonacci Search

This is a function minimization method, as opposed to a root finding algorithm.

The Fibonacci search is the strategy that yields the maximum reduction in the interval of uncertainty for a given number of evaluations.

The Fibonacci numbers are defined as

$$F_0 = F_1 = 1 \tag{2.14}$$

$$F_k = F_{k-1} + F_{k-2} \tag{2.15}$$

The first few numbers of this sequence are $1, 1, 2, 3, 5, 8, 13, \ldots$. For a given number of function evaluations, N, we start with an interval of uncertainty $[0, F_N]$.

Say we have an interval of uncertainty and the function has been evaluated at the boundaries. In order to reduce the interval of uncertainty, we have to evaluate two new points inside the interval. Then (assuming the function is *unimodal*) the interval that has lower function evaluation inside its boundaries is the new interval of uncertainty.

The most efficient way of reducing the size of the interval would be to: (1) ensure that the two possible intervals of uncertainty are the same size, and (2) once the new interval is chosen, the point inside the domain can be used and only one more function evaluation is required for selecting a new interval of uncertainty.

The interval sizes, I_k are such that

$$\begin{aligned} I_1 &= I_2 + I_3 \\ I_2 &= I_3 + I_4 \\ &\vdots \\ I_k &= I_{k+1} + I_{k+2} \\ &\vdots \\ I_{N-2} &= I_{N-1} + I_N \\ I_{N-1} &= 2I_N \end{aligned}$$

To find the successive interval sizes, we need to start from the last interval in reverse order, only after this can we start the search.

When using the Fibonacci search, we have to decide on the number of function evaluations *a priori*. This is not always convenient, as the termination criteria is often the variation of the function values in the interval of uncertainty. Furthermore, this method requires that the sequence be stored.

Fibonacci search is the optimum because in addition to yielding two intervals that are the same size and reusing one point, the interval between the two interior points converges to zero, and in the final iteration, the interval is divided into two (almost exactly), which is the optimum strategy for the last iteration.

2.3.5 Golden Section Search

Suppose we chose interval sizes, I_k are such that

$$\begin{aligned} I_1 &= I_2 + I_3 \\ I_2 &= I_3 + I_4 \\ &\vdots \\ \frac{I_2}{I_1} &= \frac{I_3}{I_2} = \frac{I_4}{I_3} = \cdots = \tau \end{aligned}$$

Then,

$$\tau^2 + \tau - 1 = 0 \tag{2.16}$$

The positive solution of this equation is the golden section ratio,

$$\lim_{k\to\infty} \frac{F_{k-1}}{F_k} = \frac{2}{1+\sqrt{5}} \equiv \tau \approx 0.6180 \tag{2.17}$$

Say we start the search with an uncertainty interval $[0, 1]$. If we evaluate two points such that the two intervals that can be chosen are the same size and one of the points is reused, we have a relatively efficient method (but not as efficient as Fibonacci's).

Evaluating the function at $1 - \tau$ and τ achieves this. The two possible intervals are $[0, \tau]$ and $[1 - \tau, 1]$, and they are of the same size. If, say $[0, \tau]$ is selected, then the next two interior points would be $\tau(1 - \tau)$ and $\tau\tau$. But $\tau^2 = 1 - \tau$ from the definition (2.16), and we already have this point!

Example 2.3: Line Search Using Golden Section
Consider the following single-variable optimization problem

$$\begin{aligned} \text{minimize} \quad & f(x) = (x-3)x^3(x-6)^4 \\ \text{w.r.t.} \quad & x \end{aligned}$$

Solve this using the golden section method.

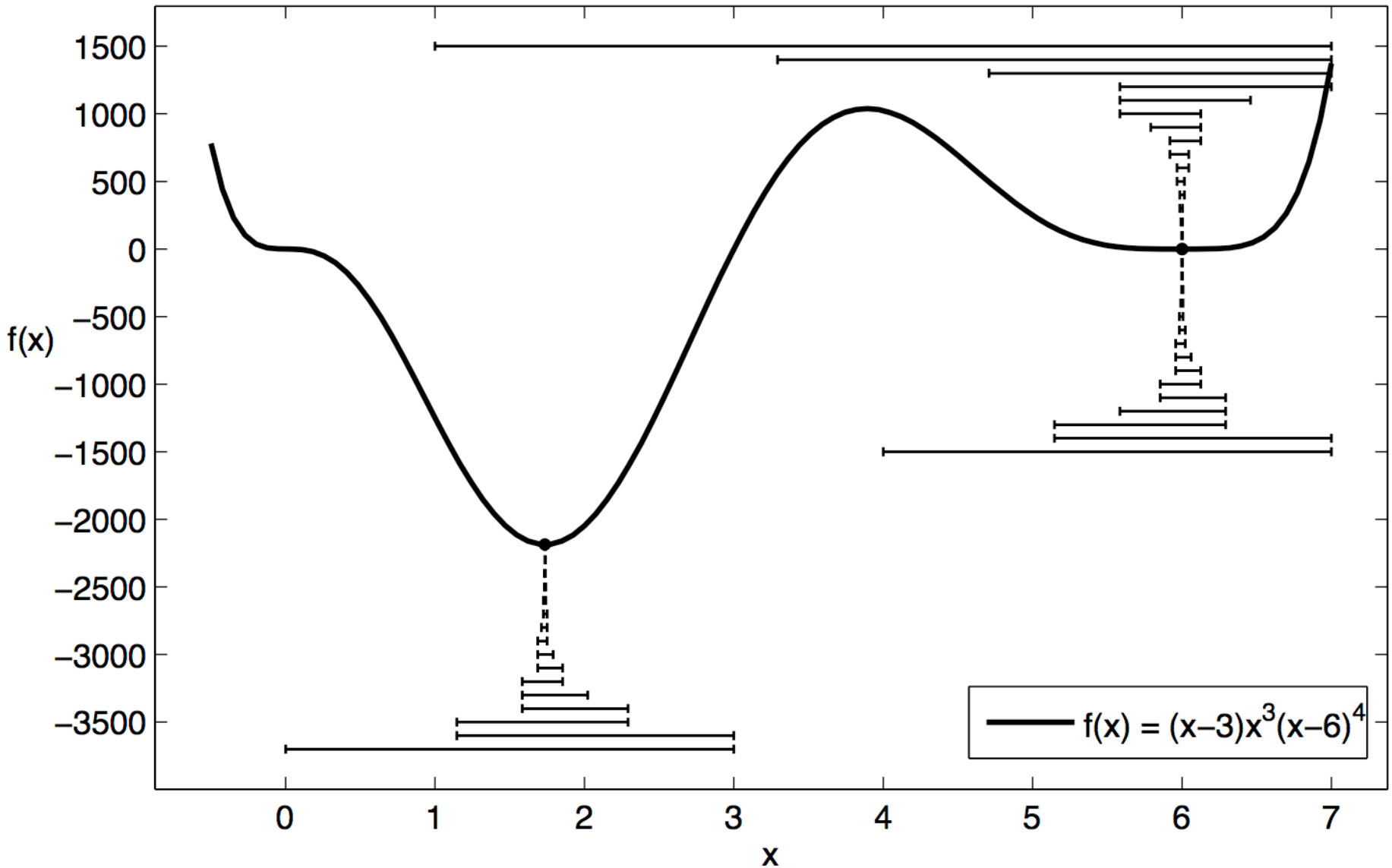


Figure 2.2: The golden section method with initial intervals $[0, 3]$, $[4, 7]$, and $[1, 7]$. The horizontal lines represent the sequences of the intervals of uncertainty.

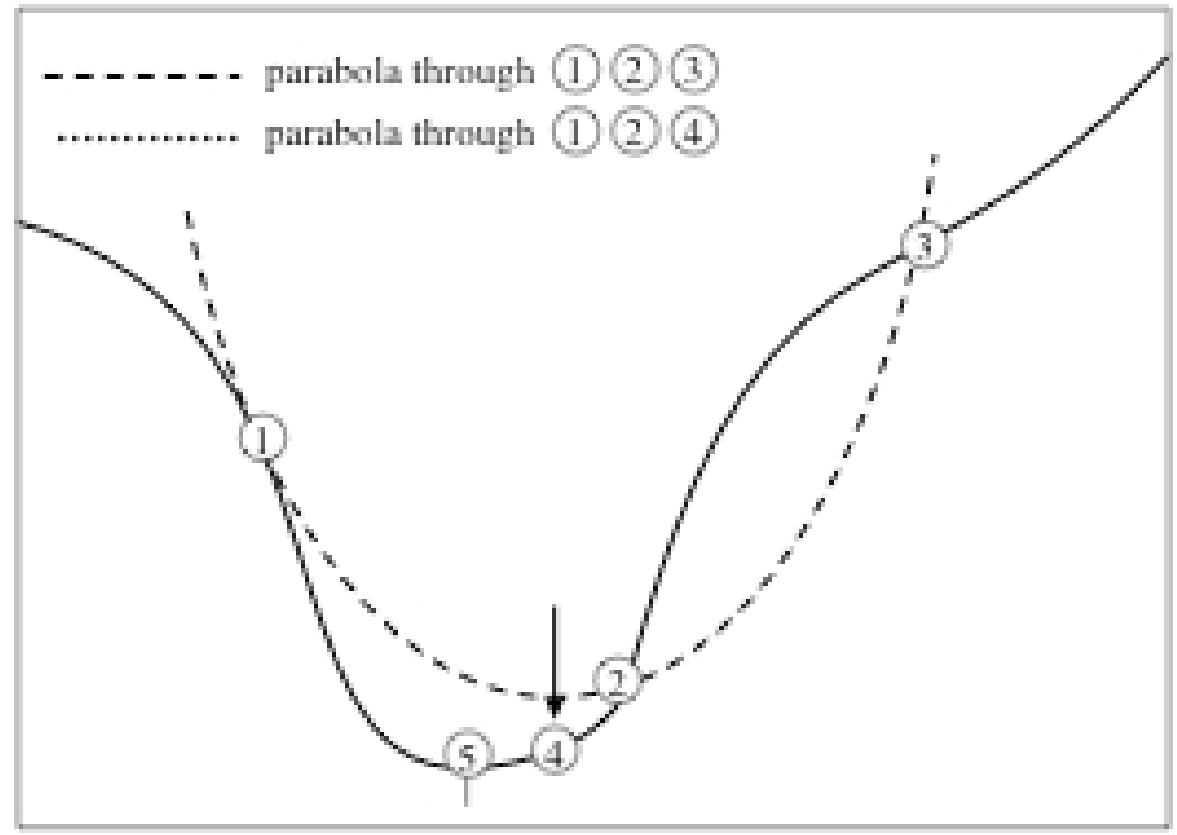
2.3.6 Polynomial Interpolation

More efficient procedures use information about f gathered during iteration. One way of using this information is to produce an estimate of the function which we can easily minimize.

The lowest order function that we can use for this purpose is a quadratic, since a linear function does not have a minimum.
Suppose we approximate f by

$$\tilde{f} = \frac{1}{2}ax^2 + bx + c. \quad (2.18)$$

If $a > 0$, the minimum of this function is $x^* = -b/a$.



To generala quadratic approximation, three independent pieces of information are needed. For example, if we have the value of the function and its first and second derivatives at point x_k, we can write a quadratic approximation of the function value at x as the first three terms of a Taylor series

$$\tilde{f}(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2 \tag{2.19}$$

If $f''(x_k)$ is not zero, and setting $x = x_{k+1}$ this yields

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}. \tag{2.20}$$

This is Newton's method used to find a zero of the first derivative.

Brent's Quadratic Fit-Sectioning Algorithm

Fits a quadratic polynomial and accepts the quadratic minimum when the function is cooperative, and uses the golden section method otherwise.

At any particular stage, Brent's algorithm keeps track of six points (not necessarily all distinct), a, b, u, v, w and x, defined as follows:

- The minimum is bracketed between a and b.
- x is the point with the least function value found so far (or the most recent one in case of a tie).
- w is the point with the second least function value.
- v is the previous value of w.
- u is the point at which the function was evaluated most recently.

The general idea is the following: parabolic interpolation is attempted, fitting through the points x, v, and w. To be acceptable, the parabolic step must(1) fall within the bounding interval (a, b), and (2) imply a movement from the best current value x that is less than half the movement of the step before last. This second criterion insures that the parabolic steps are converging, rather than, say, bouncing around in some non-convergent limit cycle.

The algorithm is as follows:

1. Start with 3 points, a, b and x.
2. w and v assume the value of x.
3. If the points x, w and v are all distinct, then go to 5.
4. Calculate u using the golden section of the larger of the two intervals $[a, x]$ or $[x, b]$. Go to 7.
5. Try quadratic fit for x, w and v. If the quadratic minimum falls inside $[a, b]$ then determine the minimum point u.
6. If the point u is close to a, b or x, then adjust u into the larger of $[a, x]$ or $[x, b]$ such that u is away from x by a minimum distance δ_{tol} (chosen according to the machine zero).
7. Evaluate $f(u)$.
8. From (a, b, x, w, v, u) determine the new (a, b, x, w, v).
9. If the larger of the intervals $[a, x]$ or $[x, b]$ is smaller than $2\delta_{\text{tol}}$, then the algorithm has converged and we can stop. Otherwise, repeat the process from 3.

The minimum of the quadratic that fits $f(x)$, $f(v)$ and $f(w)$ is

$$q = x - \frac{1}{2}\frac{(x-w)^2[f(x)-f(v)] - (x-v)^2[f(x)-f(w)]}{(x-w)[f(x)-f(v)] - (x-v)[f(x)-f(w)]}. \tag{2.21}$$

2.4 Line Search Techniques

Line search methods are related to single-variable optimization methods as they address the problem of minimizing a multivariable function along a line, which is a subproblem in any gradient-based optimization method.

After a gradient-based optimizer has computed a search direction p_k, it must decide how far to move along that direction. The step can be written as

$$x_{k+1} = x_k + \alpha_k p_k \tag{2.22}$$

where the positive scalar α_k is the *step length*.

Most algorithms require that p_k be a *descent direction*, i.e., that $p_k^T g_k < 0$, since this guarantees that f can be reduced by stepping along this direction.

We want to compute a step length α_k that yields a substantial reduction in f, but we do not want to spend too much computational effort in making the choice. Ideally, we would find the global minimum of $f(x_k + \alpha_k p_k)$ with respect to α_k but in general, it is to expensive to compute this value. Even to find a local minimizer usually requires too many evaluations of the objective function f and possibly its gradient g. More practical methods perform an *inexact* line search that achieves adequate reductions of f at reasonable cost.

2.4.1 Wolfe Conditions

A typical line search involves trying a sequence of step lengths, accepting the first that satisfies certain conditions.

A common condition requires that α_k should yield a *sufficient decrease* of f, as given by the inequality

$$f(x_k + \alpha p_k) \leq f(x_k) + \mu_1 \alpha g_k^T p_k \tag{2.23}$$

for a constant $0 \leq \mu_1 \leq 1$. In practice, this constant is small, say $\mu_1 = 10^{-4}$.

Any sufficiently small step can satisfy the sufficient decrease condition, so in order to prevent steps that are too small we need a second requirement called the *curvature condition*, which can be stated as

$$g(x_k + \alpha p_k)^T p_k \geq \mu_2 g_k^T p_k \tag{2.24}$$

where $\mu_1 \leq \mu_2 \leq 1$. Note that $g(x_k + \alpha p_k)^T p_k$ is the derivative of $f(x_k + \alpha p_k)$ with respect to α_k. This condition requires that the slope of the univariate function at the new point be greater. Since we start with a negative slope, the gradient at the new point must be either less negative or positive. Typical values of μ_2 are 0.9 when using a Newton type method and 0.1 when a conjugate gradient methods is used.

The sufficient decrease and curvature conditions are known collectively as the *Wolfe conditions*.

We can also modify the curvature condition to force α_k to lie in a broad neighborhood of a local minimizer or stationary point and obtain the *strong Wolfe conditions*

$$f(x_k + \alpha p_k) \leq f(x_k) + \mu_1 \alpha g_k^T p_k. \tag{2.25}$$

$$\left| g(x_k + \alpha p_k)^T p_k \right| \leq \mu_2 \left| g_k^T p_k \right|, \tag{2.26}$$

where $0 < \mu_1 < \mu_2 < 1$. The only difference when comparing with the Wolfe conditions is that we do not allow points where the derivative has a positive value that is too large and therefore exclude points that are far from the stationary points.

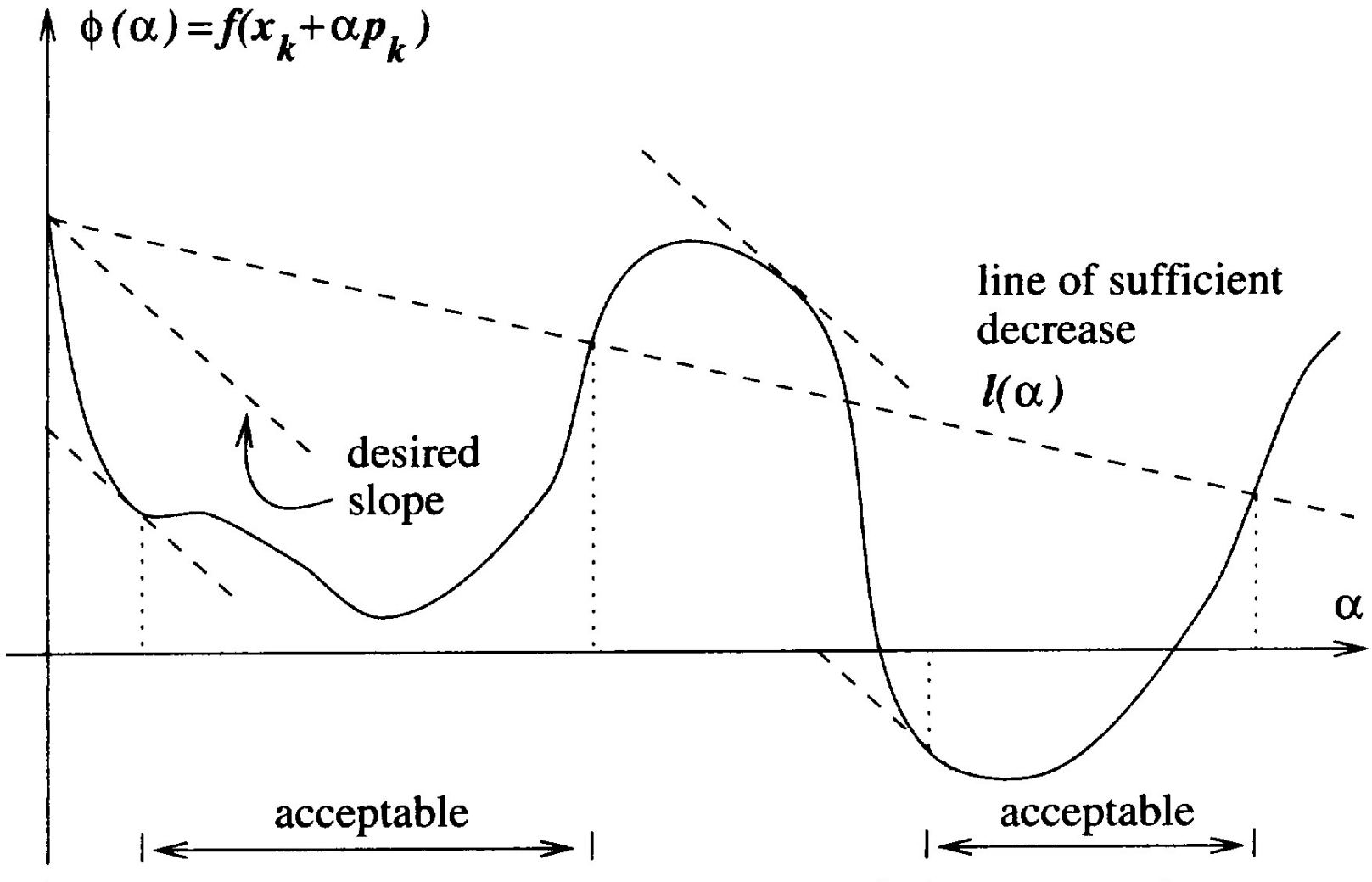


Figure 2.3: Acceptable steps for the Wolfe conditions

2.4.2 Sufficient Decrease and Backtracking

We can ignore the curvature condition by performing *backtracking*, i.e., by executing the following algorithm.

1. Choose a starting step length $0 < \bar{\alpha} < 1$.
2. If $f(x_k + \alpha p_k) \leq f(x_k) + \mu_1 \alpha g_k^T p_k$ then set $\alpha_k = \alpha$ and stop.
3. $\alpha = \rho\alpha$.
4. Return to 2.

2.4.3 Line Search Algorithm Using the Strong Wolfe Conditions

This procedure has two stages:

1. Begins with trial α_1, and keeps increasing it until it finds either an acceptable step length or an interval that brackets the desired step lengths.
2. In the latter case, a second stage (the `zoom` algorithm below) is performed that decreases the size of the interval until an acceptable step length is found.

Define the univariate function $\phi(\alpha) = f(x_k + \alpha p_k)$, so that $\phi(0) = f(x_k)$.

The first stage is as follows:

1. Set $\alpha_0 = 0$, choose $\alpha_1 > 0$ and α_{max}. Set $i = 1$.
2. Evaluate $\phi(\alpha_i)$.
3. If $[\phi(\alpha_i) > \phi(0) + \mu_1 \alpha_i \phi'(0)]$ or $[\phi(\alpha_i) > \phi(\alpha_{i-1})$ and $i > 1]$ then, set $\alpha_* = \texttt{zoom}(\alpha_{i-1}, \alpha_i)$ and stop.
4. Evaluate $\phi'(\alpha_i)$.
5. If $|\phi'(\alpha_i)| \leq - \mu_2 \phi'(0)$, set $\alpha_* = \alpha_i$ and stop.
6. If $\phi'(\alpha_i) \geq 0$, set $\alpha_* = \texttt{zoom}(\alpha_i, \alpha_{i-1})$ and stop.
7. Choose α_{i+1} such that $\alpha_i < \alpha_{i+1} < \alpha_{\text{max}}$.
8. Set $i = i + 1$.
9. Return to 2.

The second stage, the `zoom` function:

1. Interpolate (using quadratic, cubic, or bisection) to find a trial step length α_j between α_{lo} and α_{hi}.
2. Evaluate $\phi(\alpha_j)$.
3. If $\phi(\alpha_j) > \phi(0) + \mu_1 \alpha_j \phi'(0)$ or $\phi(\alpha_j) > \phi(\alpha_{\text{lo}})$, set $\alpha_{\text{hi}} = \alpha_j$.
4. Else:
   4.1 Evaluate $\phi'(\alpha_j)$.
   4.2 If $|\phi'(\alpha_j)| \leq - \mu_2 \phi'(0)$, set $\alpha_* = \alpha_j$ and stop.
   4.3 If $\phi'(\alpha_j)(\alpha_{\text{hi}} - \alpha_{\text{lo}}) \geq 0$, set $\alpha_{\text{hi}} = \alpha_{\text{lo}}$.
   4.4 $\alpha_{\text{lo}} = \alpha_j$.

Example 2.4: Line Search Algorithm Using Strong Wolfe Conditions

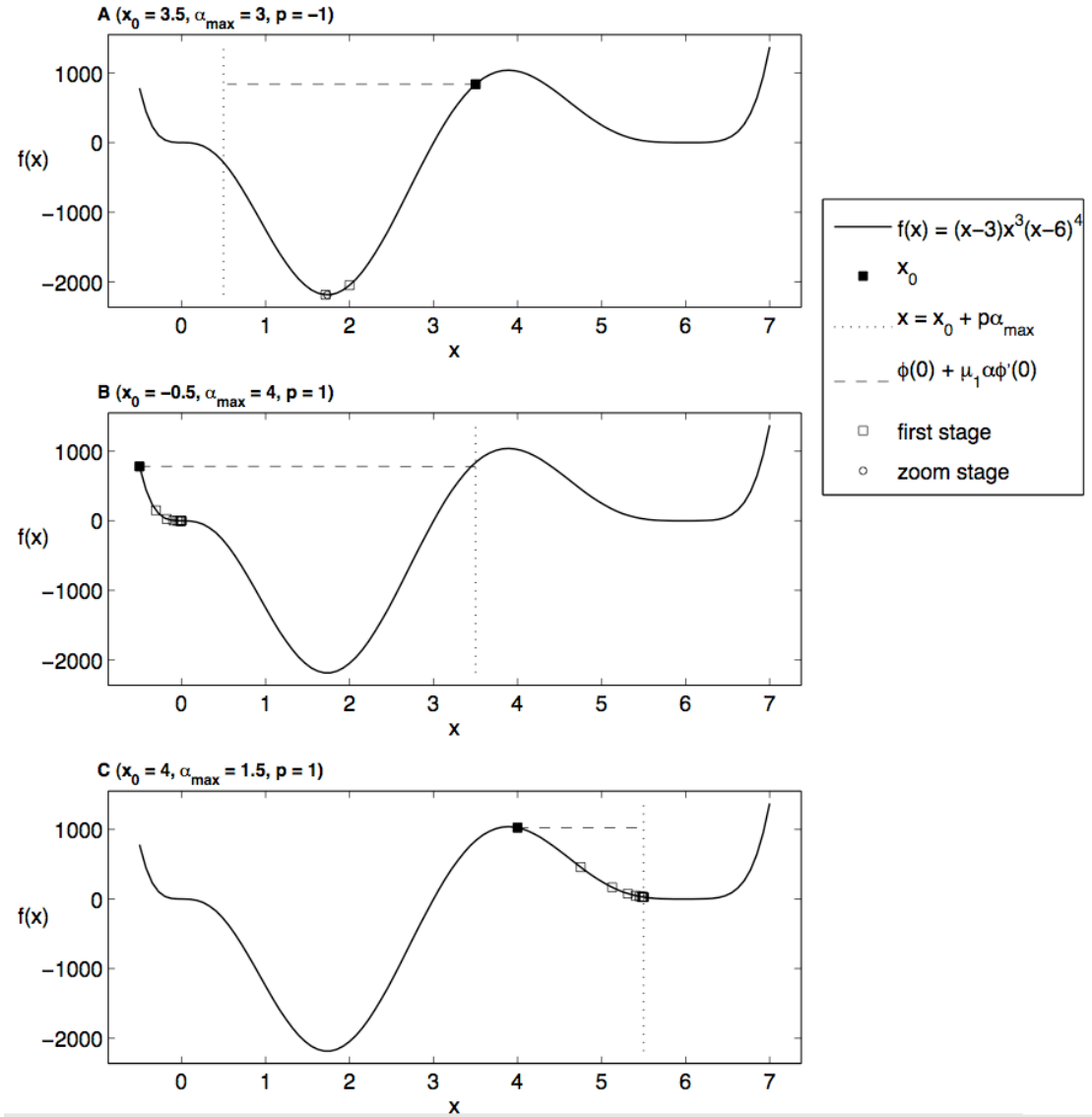


Figure 2.4: The line search algorithm iterations. The first stage is marked with square labels and the zoom stage is marked with circles.

References

[1] A. D. Belegundu and T. R. Chandrupatla. *Optimization Concepts and Applications in Engineering*, chapter 2. Prentice Hall, 1999.

[2] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*, chapter 4. Academic Press, 1981.

[3] J. Nocedal and S. J. Wright. *Numerical Optimization*, chapter 3. Springer, 2nd edition, 2006.