1D Optimization Algorithms

This type of algorithms is needed for two purposes:

- Finding the minimum of some objective function $F(x)$. This objective function may be a stand-alone physical function, or it may be the merit function in a multivariate optimization analysis conducted by a directional line search algorithm. Reliability analysis by FORM is one example. In that case it is often required to find the optimal step size along each search direction; hence, $F(x)$ is the merit function that determines what constitutes an optimal step size.
- Finding the root of a function $f(x)$, i.e., determining the value(s) of x that yields f=0.

The following notation applies:

$F(x)$ = objective function, or merit function, relevant in optimization, not in root-finding

$f(x) = \mathrm{d}F/\mathrm{d}x$ = function whose root is sought

$h(x) = \mathrm{d}f/\mathrm{d}x = \mathrm{d}^2F/\mathrm{d}x^2$ = Hessian, i.e., second-order derivative of the objective function

1D Newton-Raphson Line Search

The one-parameter version of the algorithm named after Isaac Newton (1643-1727) and Joseph Raphson (1648–1715), implemented in the class *RNewtonLineSearchAlgorithm*, can be derived from a Tayor expansion of the function

$$f(x) = \frac{dF(x)}{dx} = 0 \tag{1}$$

which is the function whose root is sought and $F(x)$ is the objective function in a one-parameter optimization problem

$$x^* = \arg\min\{F(x)\} \tag{2}$$

First-order Taylor approximation of $f(x)$ yields the following version of Eq. (1):

$$f(x) \approx f(x_0) + f'(x_0)\cdot(x - x_0) = 0 \tag{3}$$

Solving for x yields

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{4}$$

The linearization of $f(x)$ in Eq. (3) implies that x solves $f(x)$=0 when $f(x)$ is linear. When $f(x)$ is nonlinear Eq. (4) is used as a first approximation in a search that repeatedly evaluates Eq. (4) at the previous x-value:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{5}$$

Bisection Line Search

This algorithm is implemented in C++ in the class *RBisectionLineSearchAlgorithm*. A simple but somewhat computationally inefficient method to solve the 1D optimization problem

$$x^* = \arg\min\{F(x)\} \tag{6}$$

and the 1D root-finding problem

$$f(x) = \frac{dF(x)}{dx} = 0 \tag{7}$$

is to repeatedly cut in half intervals along the x-axis where the solution is known to exist. The algorithm starts with a user-given guess for the interval in which the solution is assumed to exist.

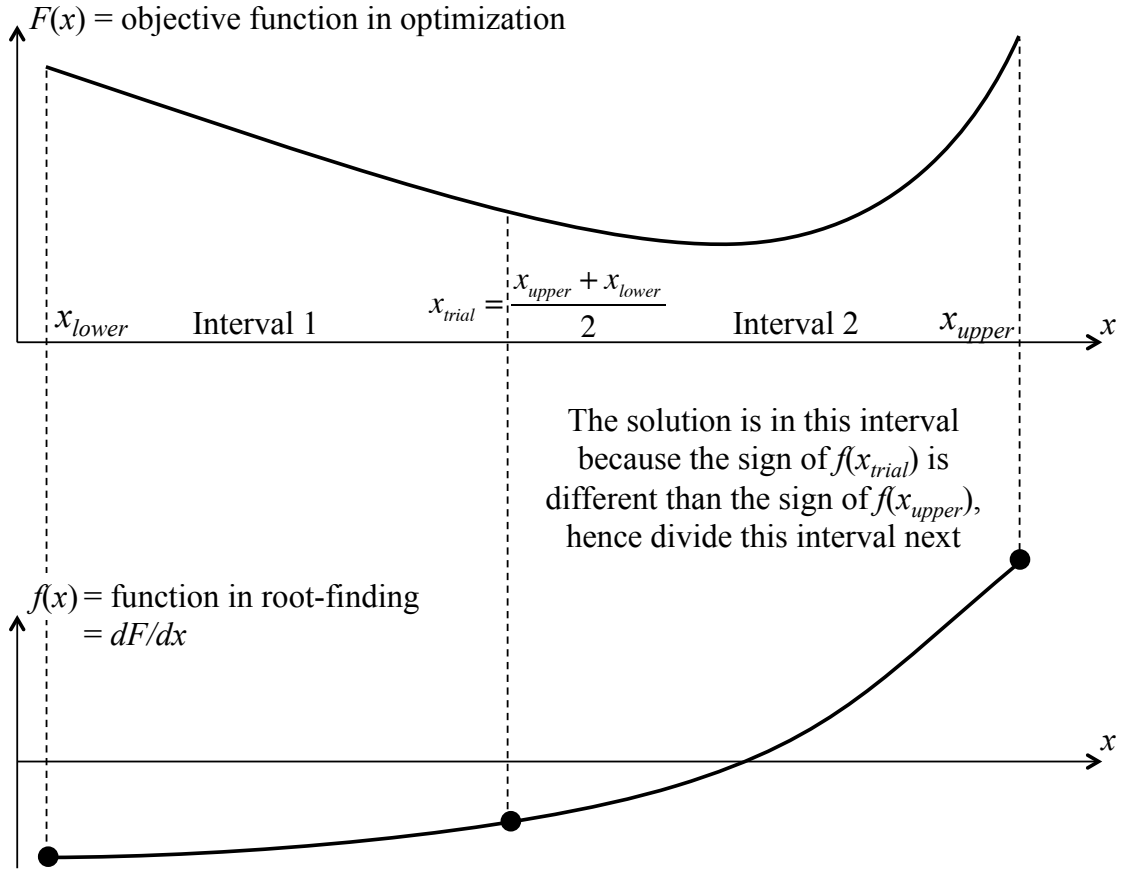

Figure 1: Dividing of intervals in the bisection algorithm.

Starting with the initial bounds x_{lower} and x_{upper}, and the calculation of the corresponding function values $f(x_{lower})$ and $f(x_{upper})$ the interval midpoint

$$x_{trial} = \frac{x_{upper} + x_{lower}}{2} \tag{8}$$

and the corresponding value $f(x_{trial})$ is computed. If $f(x_{trial})$ and $f(x_{lower})$ has a different signs then the solution is known to be in the left-most interval, and vice versa.

Golden Section Line Search

This algorithm is implemented in Rts in the class *RGoldenSectionLineSearchAlgorithm*. It is an interesting algorithm to determine the solution to the minimization problem

$$x^* = \arg\min\{F(x)\} \tag{9}$$

based on the golden section ratio, which also appears also in aesthetical studies. At every iteration uses the value of $F(x)$ at three x-values and tests the value of $F(x)$ at a fourth. In Figure 1 the four points are labelled x_i, i=1,2,3,4. Suppose we have evaluated $F(x_1)$, $F(x_2)$, and $F(x_4)$ but not $F(x_3)$. Then $F(x_3)$ is evaluated and its value is below $F(x_2)$, which implies that the solution must lie somewhere in the range x_2 to x_4. Conversely, if $F(x_2) < F(x_3)$ then the solution must lie in the range x_1 to x_3. The interval that contains the solution gets an added point, located according to the golden section ratio as in the initial four points.

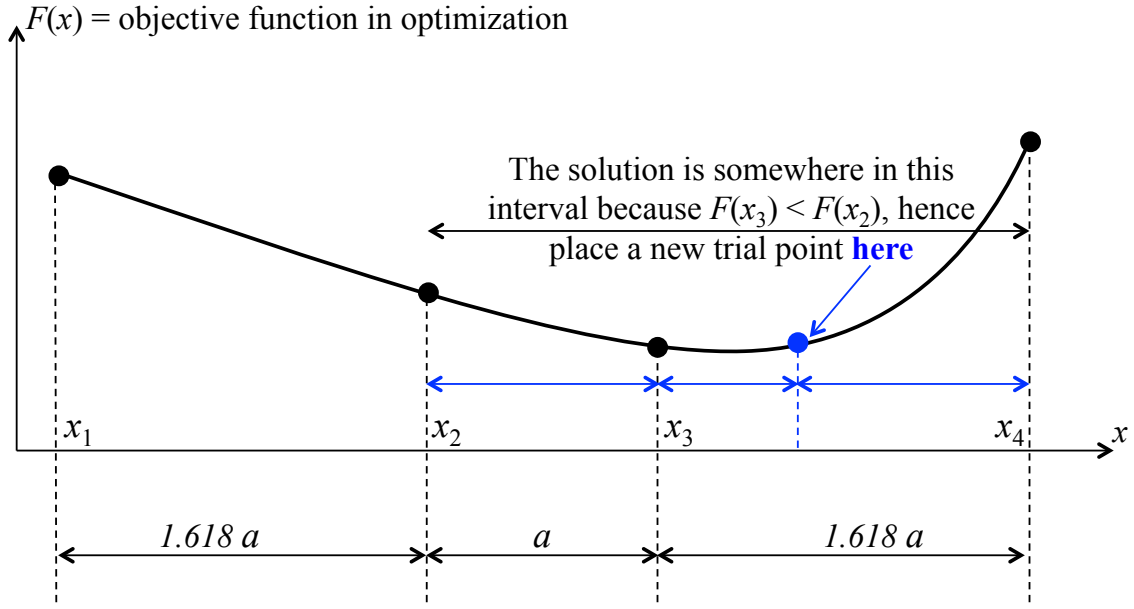

Figure 2: Golden section algorithm.

The ratio of the "large intervals" (say, a) to the "small intervals" (say, b) is determined by the two equations

$$\frac{a}{b} = \frac{a+b}{a} = \varphi \tag{10}$$

where φ is a constant. One approach for determining φ is to write the second equation as

$$\frac{a+b}{a} = 1 + \frac{b}{a} = 1 + \frac{1}{\varphi} = \varphi \tag{11}$$

where the first equation was used in the second equality. The resulting equation is a second-order equation

$$1+\frac{1}{\varphi}=\varphi \qquad \Rightarrow \qquad \varphi^2-\varphi-1=0 \tag{12}$$

which has the solutions

$$\varphi=\frac{1\pm\sqrt{5}}{2} \tag{13}$$

The negative root is discarded and the positive root is $\varphi \approx 1.618$.

Secant Line Search

This algorithm is implemented in C++ in Rts in the class *RSecantLineSearchAlgorithm*. One approach to find the root of the function

$$f(x)=\frac{dF(x)}{dx}=0 \tag{14}$$

or the solution to the minimization problem

$$x^*=\arg\min\{F(x)\} \tag{15}$$

is to repeatedly determine the root of the secant ("chord line") between values x_{lower} and x_{upper} that the solution is assumed to lie within.

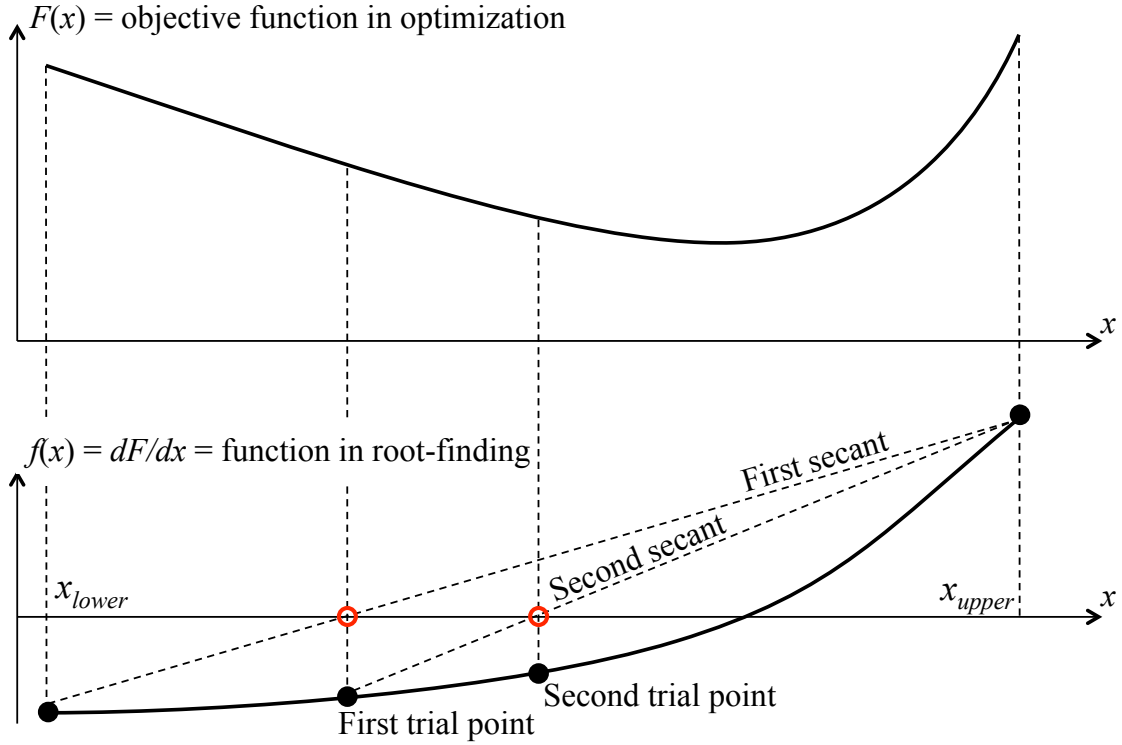

Figure 3: Sequential trial roots in the secant algorithm.

Given the value of $f(x)$ at those two x-values the line of the secant is defined by the triangle equality

$$\frac{x_{upper} - x_{lower}}{f_{upper} - f_{lower}} = \frac{x_{trial} - x_{lower}}{0 - f_{lower}} \tag{16}$$

Solving for x_{trial} yields the formula

$$x_{trial} = x_{lower} - f_{lower} \cdot \left(\frac{x_{upper} - x_{lower}}{f_{upper} - f_{lower}} \right) \tag{17}$$

which is recursive because at each step either x_{lower} or x_{upper} is replaced by x_{trial}, depending on the where the solution lies; if $f(x_{lower})$ and $f(x_{trial})$ have different signs then the solution is to the left of x_{trial}, and vice versa.