Minimization of a function.

Our goal is to find a minimum of a function with maximum efficiency. Maximum efficiency means a minimum number of function evaluations (and a minimal number of evaluations of the derivatives). Surprisingly, even the very simple bisection algorithm below is quite efficient…

Minimization in one dimension
Consider a function \( f(x) \) of the scalar variable \( x \) that has exactly one minimum in the interval \( a < x < b \). We wish to determine the minimum of \( f(x) \) in that interval. The simplest possible method is to just plot the function and to find the minimum by inspection. To determine the minimum with an accuracy of \( \Delta = (b - a) / N \), we are required to do \( N \) function evaluations. This is not efficient, but possible and can be done in one dimension, providing a global view of the function.

One of the simplest methods (that is still reasonably efficient) is based on a recursive division of the interval by a factor of two reducing the size of an interval that includes a minimum by a factor of two.

Here is an algorithm for bisection
\begin{itemize}
  \item Initiate by computing the function value at the edges of the interval \( f_L \equiv f(x_a) \) and \( f_R \equiv f(x_b) \) and \( x_L \equiv x_a \) \( x_R \equiv x_b \)
  \item Compute the center of the interval \( x_c = 0.5(x_R - x_L) \) and the function value at the center \( f(x_c) \)
  \item Consider the two new interval \( (x_L, x_c) \) and \( (x_c, x_R) \). One of the intervals contains the minimum. Calculate the function value in the center of one of the two nearby interval (say \( (x_L, x_c) \)). If the function value at \( f(0.5(x_L + x_c)) \) is smaller than the function values at the interval edges, \( f(x_c) \) and \( f(x_L) \), then this half contains the minimum, otherwise it is the second half -- \( (x_c, x_R) \). Note that this procedure is still ok if we hit a minimum when the interval is bisected.
  \item Repeat dividing the interval that contains the minimum until desired accuracy \( \Delta \) is reached
\end{itemize}

The number of step in the above algorithm is MUCH smaller than plotting the function, since \( \Delta = (a - b) / 2^n \) where \( n \) is the current number of function evaluations
\[
(n = \log(N) / \log(2))
\]

We can speed up the convergence to a minimum if we can use derivative information. Here is a variant on the Newton Raphson algorithm that uses first and second derivatives. At a minimum the first derivative is zero. If we expand the function in the neighborhood of the minimum we have
\[
\frac{df}{dx}_{x=x_c} = 0 = \left. \frac{df}{dx} \right|_{x=x_c} + \left. \frac{d^2 f}{dx^2} \right|_{x=x_c} (x_m - x_c)
\]

\[
x_m = x_c - \left( \frac{df}{dx}_{x=x_c} \left/ \frac{d^2 f}{dx^2}_{x=x_c} \right. \right)
\]

where \( x_c \) means the current guess for the minimum. In this algorithm we keep only one position and evaluate the first and second derivative of the function at that point. We use the derivative information to determine our next step. If the function is exactly parabolic then the process converged in just one step (!). Sufficiently close to the minimum we always expect the function to be parabolic. Therefore the above approach is especially useful in refining close guesses to exact values. For functions with more than one variables, even if the function is not parabolic, this process converged very quickly. If you have first and the second derivative this approach makes a lot of sense.

- Is it possible using the algorithm described above NOT to converge to a minimum?