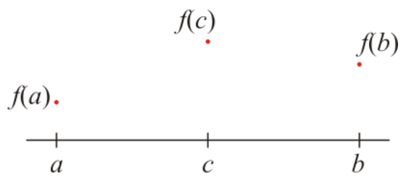
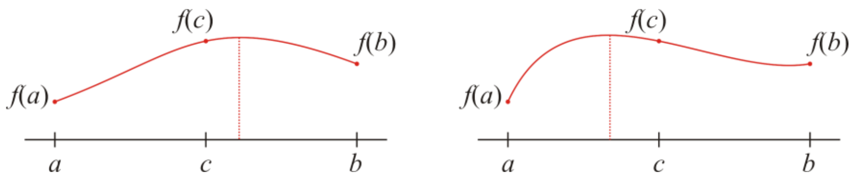
9.1 Golden-ratio search

The problem at hand is finding the maximum of a real-valued function $f(x)$ for a real variable x. Suppose we want to find a maximum on the interval $[a, b]$.

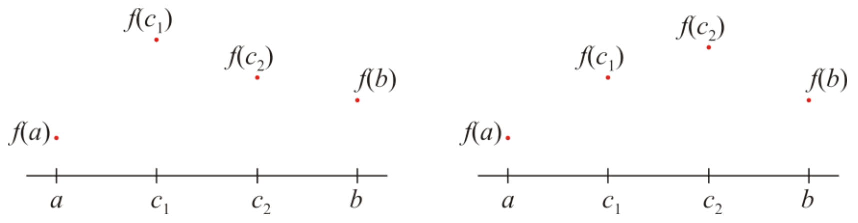
In finding a root, given an interval $[a, b]$ where $f(a)$ and $f(b)$ have opposite signs, we can find a sub-interval by finding the mid-point $c = (a + b)/2$ and choosing a sub-interval based on the sign of $f(c)$. When trying to find a local maximum of a function $f(x)$, evaluating the function at a single point does not allow us to constrain which sub-interval the maximum is on; for example, if we knew $f(a)$, $f(b)$ and $f(c)$ as follows,



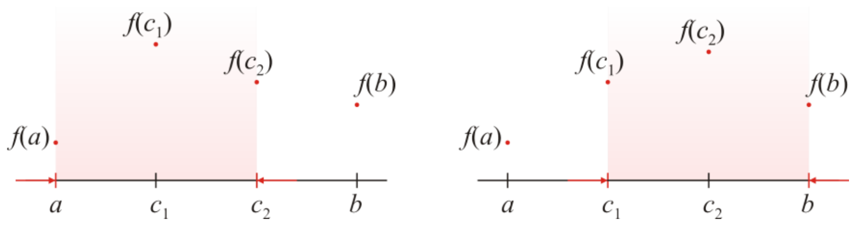
then we know nothing about where a local maximum may exist on the interval: it could be either to the left or the right of the center, as are exemplified here:



If, however, we divide the interval up into four points, then we can get some more information; for example, dividing the interval into three equal sub-intervals and evaluating the function at both interior points, we may have one of two scenarios: either $f(c_1) > f(c_2)$ or $f(c_1) < f(c_2)$, as shown here:



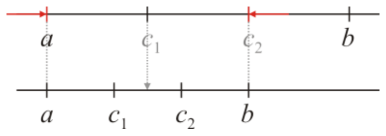
In the first case, we are guaranteed that a local maximum must be on the sub-interval $[a, c_2]$, while in the second case, we are guaranteed that a local maximum must be in the sub-interval $[c_1, b]$:



Thus, a reasonably trivial algorithm is, given [*a*, *b*]:

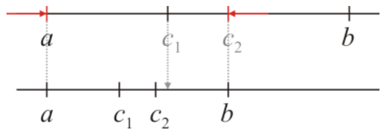
1. While $b - a > \varepsilon_{\text{step}}$,
    a. calculate and set $c_1 \leftarrow a + \frac{1}{3}(b-a)$ and $c_2 \leftarrow a + \frac{2}{3}(b-a)$,
    b. if $f(c_1) > f(c_2)$, set $b \leftarrow c_2$,
       otherwise, set $a \leftarrow c_1$.
2. Return whichever is larger: $f(a)$ or $f(b)$.

There is only one weakness in this algorithm: at each step, we must perform two function evaluations instead of just one, even though the older points overlap the new sub-interval. For example, suppose that with the first step, we find that the maximum is on [a, c_2], in which case, on the new interval, we must find two new points c_1 and c_2 and evaluate the function at those two points:



Note that we have already evaluated the function at the old value of c_1, but we must calculate two new values of c_1 and c_2 using the smaller interval.

Instead, notice what happens if divide the interval into ratios 4:2:4 instead of 1:1:1, we get the following:



Notice that the new c_2 is very close to the older c_1? Ideally, however, we would like to have the two coincide. Suppose that we have a value $\frac{1}{2} < \gamma < 1$ such that

$$c_1 \leftarrow b - \gamma(b-a)$$
$$c_2 \leftarrow a + \gamma(b-a)$$

The width of the interval was originally $b - a$, but after one step, the width of the interval is now $\gamma(b-a)$. Suppose now that we determine that the new interval is [a, c_2]. In this case, the two new interior points are:

$$c_1 \leftarrow \underbrace{\left(a + \gamma(b-a)\right)}_{\text{the old } c_2 \text{ from the first step}} - \gamma^2(b-a)$$
$$c_2 \leftarrow a + \gamma^2(b-a)$$

We want the old c_1 to equal the new c_2, or

$$b - \gamma(b-a) = a + \gamma^2(b-a).$$

Bringing everything to one side, we see that this is:

$$b - a - \gamma(b-a) - \gamma^2(b-a) = 0$$

or

$$(b-a)(1-\gamma-\gamma^2) = 0 .$$

This is zero if and only if γ is a root of the polynomial $1 - x - x^2$, which has the roots

$$\frac{1 \pm \sqrt{1 - 4 \cdot 1 \cdot (-1)}}{2 \cdot (-1)} = \frac{-1 \pm \sqrt{5}}{2} .$$

The root is positive if and only if we choose the $\gamma = \frac{-1+\sqrt{5}}{2} = \frac{\sqrt{5}-1}{2} \approx 0.6180339887498950$. Some of you may recognize this number: it is the inverse of the golden ratio $\varphi = \frac{\sqrt{5}+1}{2}$ so $\varphi^{-1} = \frac{\sqrt{5}-1}{2}$. You can check this yourself by calculating

$$\varphi \cdot \varphi^{-1} = \frac{\sqrt{5}+1}{2} \cdot \frac{\sqrt{5}-1}{2} = \frac{5 - \sqrt{5} + \sqrt{5} - 1}{4} = \frac{4}{4} = 1 .$$

Thus, let

$$\begin{aligned} c_1 &\leftarrow b - \varphi^{-1}(b-a) \\ c_2 &\leftarrow a + \varphi^{-1}(b-a) \end{aligned}$$

If you want to see the approximate numerical values, we have:

$$\begin{aligned} c_1 &\leftarrow b - 0.6180(b-a) = a + 0.3820(b-a) \\ c_2 &\leftarrow a + 0.6180(b-a) \end{aligned}$$

Thus, if you split the interval up into thirds, each additional step reduces the interval size to 66.66% the original width but requires two function evaluations. If you split the interval up using the inverse of the golden ratio as described above, after the initial step, each subsequent step shirks the interval by 61.80% the original width, and so we can perform two steps using two function evaluations to get that the interval has now shrunk to 38.20% of the original size.

As for the algorithm, given [a, b] and a real-valued function $f(x)$, to find a maximum on that interval:

```
1. Calculate and set $c_1 \leftarrow b - \varphi^{-1}(b-a)$ and $c_2 \leftarrow a + \varphi^{-1}(b-a)$.
2. While $b - a > \varepsilon_{\text{step}}$,
    a. if $f(c_1) > f(c_2)$, calculate and set in this order:
            $b \leftarrow c_2$
            $c_2 \leftarrow c_1$
            $c_1 \leftarrow b - \varphi^{-1}(b-a)$
    b. otherwise, $f(c_1) < f(c_2)$, set $b \leftarrow c_2$, so calculate and set in this order:
            $a \leftarrow c_1$
            $c_1 \leftarrow c_2$
            $c_2 \leftarrow a + \varphi^{-1}(b-a)$
3. Return whichever is largest: $f(a)$, $f(c_1)$, $f(c_2)$ or $f(b)$.
```

Implementation in C++

The following is a reasonable implementation of this algorithm in C++:

```
#include <utility>
#include <cassert>
#include <cmath>

std::pair<double, double> golden_ratio_search( double f( double x ),
                            double a, double b,
                            double eps_step ) {
    assert( b > a );
    // Calculate these explicitly to minimize error
    double const inv_phi{(std::sqrt(5.0) - 1.0)/2.0};

    double  fa{f( a )};
    double  c1{b - inv_phi*(b - a)};
    double fc1{f( c1 )};
    double  c2{a + inv_phi*(b - a)};
    double fc2{f( c2 )};
    double  fb{f( b )};

    while ( (b - a) > eps_step ) {
        if ( fc1 > fc2 ) {
              b = c2;
             fb = fc2;
             c2 = c1;
            fc2 = fc1;
             c1 = b - inv_phi*(b - a);
            fc1 = f( c1 );
        } else if ( fc1 < fc2 ) {
              a = c1;
             fa = fc1;
             c1 = c2;
            fc1 = fc2;
             c2 = a + inv_phi*(b - a);
            fc2 = f( c2 );
        } else {
            assert( false );
        }
    }

    // Return the maximum of f(a), f(c ), f(c ) and f(b)
    //                                1      2

    if ( (fa > fc1) && (fa > fc2) && (fa > fb) ) {
        return std::make_pair( a, fa );
    } else if ( (fc1 > fc2) && (fc1 > fb) ) {
        return std::make_pair( c1, fc1 );
    } else if (  fc2 > fb ) {
        return std::make_pair( c2, fc2 );
    } else {
        return std::make_pair( b, fb );
    }
}
```

Acknowledgments