One-Dimensional Optimization

MA 348
Kurt Bryan

The basic task of one-dimensional optimization is this: we are given a function $f(x)$ and possibly some initial guess x_0 which is "near" a local minimum for f. We might even be given an interval $[x_1, x_2]$ in which the function f is known to have a minimum. We are to locate the minimum with some specified accuracy.

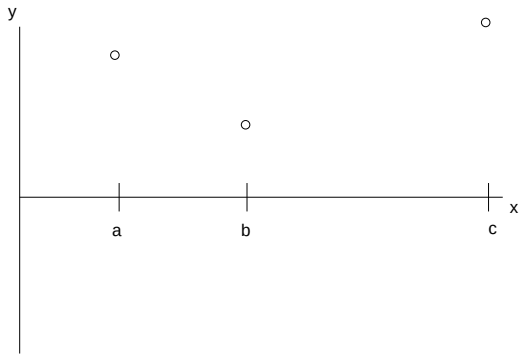
How we do this depends on the information we have about f. One important issue is whether we can compute the derivative of f. Obviously if f isn't differentiable then this is impossible, but even if f is differentiable the derivative might be difficult or computationally costly to compute. In general if the derivative information is easily available then it pays to use it.

It's important that we do a decent job with one-dimensional optimization, for whatever algorithms we come up with will ultimately become part of our n-dimensional optimization algorithms. Most algorithms for minimizing functions of n variables work by doing a sequence of 1D optimizations, possibly BILLIONS of times, so any efficiency in the 1D algorithm is multiplied correspondingly.

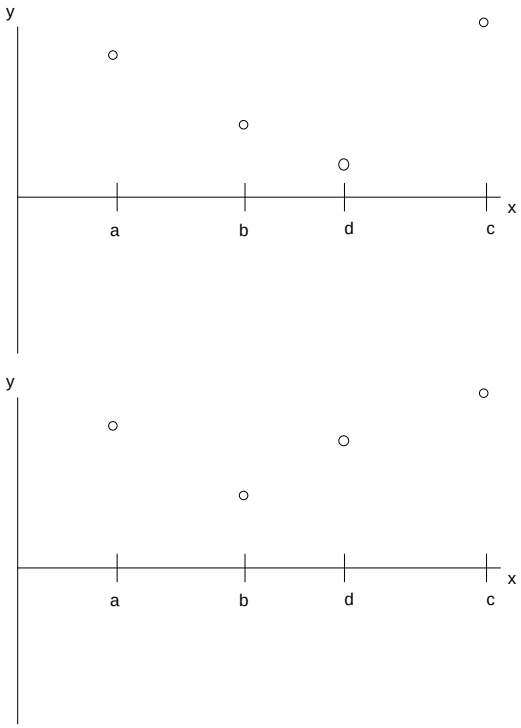
The Golden Section Method

The *golden section* search provides a reasonably efficient way to locate a local minimum for a function $f(x)$, and moreover, you don't even need $f(x)$ to be differentiable for this method to work. The only requirement is that $f(x)$ be continuous.

The basis of the method is the following observation. In the picture below,



the function $f(x)$ has been evaluated at three points, $x = a$, $x = b$, and $x = c$. As shown in the picture $f(b) < f(a)$ and $f(b) < f(c)$. It's pretty obvious that f *must* have a minimum somewhere between $x = a$ and $x = c$. In this situation the minimum value is said to be *bracketed*, that is, trapped in the interval $[a, c]$. Choose a new point $x = d$ in the largest subinterval, $[a, b]$ or $[b, c]$. In this case, we choose d in $[b, c]$. Evaluate $f(d)$. There are two possibilities: either $f(d) > f(c)$ or $f(d) \leq f(c)$, as illustrated below.



If $f(d) \leq f(b)$ then set $a = b$, $b = d$ and repeat the whole process again. Otherwise $f(d) > f(b)$; set $c = d$ and repeat the whole process again. In either case, the interval $[a, c]$ shrinks in length and the minimum is still bracketed. Iterating this process leads to a series of shrinking, nested intervals, which gradually converge down to a minimum. We can terminate the process when the length of $[a, c]$ is less than a predetermined amount; we will then have located a minimum to within that specified tolerance. How one chooses the initial points a, b, and c to bracket the minimum is something we'll discuss in a moment.

Below is an example with the function $f(x) = (x - 1)^2$; the true minimum value is at $x = 1$. The initial points which bracket the minimum are $a = 0.4$, $b = 1.25$ and $c = 1.5$. At each iteration the new "test" point $x = d$ is chosen as the midpoint of the largest of the subintervals $[a, b]$ or $[b, c]$. The table lists the value of a, b, c and $f(b)$ at each iteration. In this case we iterate until $c - a < 10^{-2}$.

a	b	c	$f(b)$
0.400	0.950	1.500	0.003
0.400	0.675	0.950	0.106
0.675	0.950	1.225	0.003
0.675	0.812	0.950	0.035
0.812	0.950	1.087	0.003
0.812	0.881	0.950	0.014
0.881	0.950	1.019	0.003

a	b	c	$f(b)$
0.950	1.019	1.053	0.000
0.950	0.984	1.019	0.000
0.950	0.967	0.984	0.001
0.967	0.984	1.002	0.000
0.984	1.002	1.010	0.000
0.984	0.993	1.002	0.000

There is an important question left unanswered—how should we choose the test point $x = d$? The strategy of taking it as the midpoint of the largest of $[a, b]$ or $[b, c]$ isn't bad, but

it turns out that there is a slightly better strategy. Suppose that at a given iteration, $[b, c]$ is the largest subinterval; then you should choose $d = b + R(c - b)$, where $R = \frac{3-\sqrt{5}}{2} \approx 0.38197$. This is in contrast to the "midpoint" search, which corresponds to $R = 0.5$. Similarly, if $[a, b]$ is the largest subinterval, choose $d = b - R(b - a)$.

What is the reasoning behind such a strange choice for d? Consider the simple figure below, illustrating an interval $[a, c]$ at a certain iteration in the algorithm. The numbers a, b, and c are given and we have to decide at which point d to evaluate the function $f(x)$.

I've drawn the picture so that $[b, c]$ is larger than $[a, b]$. The new test point d is to be chosen between b and c. For convenience, define

$$R = \frac{b - a}{c - a}, \tag{1}$$

so R is the "fractional" distance of b from a to c. Let us require of our search algorithm that at each iteration the newest point will be chosen at fractional distance R from the left end of the largest subinterval ($[b, c]$ above). The choice we make for R determines where we will choose to put b in $[a, c]$, and where we will put d in $[b, c]$. Thus in the above example we will choose d a fractional distance R through the interval $[b, c]$, so that $\frac{d-b}{c-b} = R$, which is equivalent to $d = b + R(c - b)$. Now from equation (1) we have $b = a + R(c - a)$ and hence we can write

$$d = b + R(c - b) = a + R(c - a) + R(c - a - R(c - a)) = a + (c - a)(2R - R^2).$$

Now consider the situation after we've chosen d. We evaluate $f(d)$ and depending on the outcome the next subinterval will be either $[a, d]$ or $[b, c]$. We will choose R in such a way as to minimize the "worst case" outcome, that is, choose R to minimize the quantity

$$\max(d - a, c - b).$$

This is equivalent to minimizing

$$g(R) = (c - a) \max(2R - R^2, 1 - R).$$

Note that the $c - a$ in front is irrelevant for finding the optimal R. You can check that g is as small as possible when $2R - R^2 = 1 - R$, i.e., when $R^2 - 3R + 1 = 0$. This occurs when $R = (3 - \sqrt{5})/2 \approx 0.382$. Thus each new test point b should be chosen a proportion R of the distance from a to c. In this case the ratio of the distance $b - a$ to the distance $c - b$ is $(1 - R)/R = (1 + \sqrt{5})/2$. This is why the method is called the golden section method; the number $(1 - R)/R$ is the "famous" golden mean, $(1 + \sqrt{5})/2$.

The point is that this method of choosing the new test points should minimize the worst case outcome at each iteration. It generally leads to faster convergence. The following table

illustrates the same example used above, but using the modified strategy for choosing the next test point:

a	b	c	$f(b)$
0.400	1.080	1.500	0.006
0.400	0.820	1.080	0.032
0.820	1.080	1.240	0.006
0.820	0.981	1.080	0.000
0.820	0.919	0.981	0.007
0.919	0.981	1.019	0.000
0.981	1.019	1.042	0.000
0.981	1.004	1.019	0.000
0.981	0.995	1.004	0.000
0.995	1.004	1.010	0.000

The new strategy converges to the required accuracy in 10 iterations, instead of 13 as required by the midpoint strategy.

Suppose we start out with the minimum trapped in the interval $[a, b]$ and we desire to locate the minimum to within a tolerance δ. With the Golden Section Search, at each stage of the algorithm the length of the bracketing interval is at most a fraction $1 - R = \frac{\sqrt{5}-1}{2}$ of the length of the previous bracketing interval, and so after n iterations we will have trapped the minimum to within $(b-a)(\frac{\sqrt{5}-1}{2})^n$. We can terminate when $(b-a)\left(\frac{\sqrt{5}-1}{2}\right)^n \leq \delta$, that is, when

$$n \geq \frac{\log(\delta) - \log(b-a)}{\log(\frac{\sqrt{5}-1}{2})} \approx 2.08(\log(b-a) - \log(\delta)).$$

Bracketing a Minimum

The final issue is how to actually go about initially bracketing a minimum of a function $f(x)$. There isn't a lot to say here. You should use whatever special knowledge you have of the function you're trying to minimize, in order to make an educated guess at the location of a minimum. It is usually the case (especially when doing 1D optimization as part of an n-dimensional problem) that you have SOME information about where to start the search.

If you have no information about where a minimum might occur, you could simply try the following strategy: choose two points $x = a$ and $x = b$. I'll assume that $f(b) \leq f(a)$. Compute $f(c)$ for $c = a + p_k(b-a)$ and $k = 1, 2, 3 \ldots$, where p_k is some increasing sequence that tends to infinity, for example, $p_k = 2^k$, or $p_k = k$. Continue until $f(c) > f(b)$, if ever. You will then have a bracketing triple a, b, c.

Tolerance Issues

Computers only perform computations to a finite number of digits. Typically single precision computations are accurate to around 7 significant (decimal) figures, double precision to around 15 significant figures. One way to quantify this is to consider the smallest number $\epsilon > 0$ such that $1 + \epsilon \neq 1$ in the computer's floating point arithmetic. This number ϵ is called the *unit round* or *machine epsilon*, and it varies from computer to computer, and also depends on the implementation of floating point arithmetic on any particular machine. For most computers using IEEE floating point arithmetic $\epsilon \approx 10^{-7}$ in single precision and $\epsilon \approx 10^{-16}$. In general $x + y \neq x$ (in floating point math) only if $|y| \geq |x|\epsilon$.

The fact that the computer does computations to finite precision has implications for choosing the stopping tolerance of numerical algorithms like the Golden Section Method. To analyze the situation we need to recall Taylor's Theorem, which in the "second order" case says that if we have a function $f(x)$ which is twice differentiable and a "base point" a then for any x

$$f(x) = f(a) + f'(a)(x-a) + \frac{1}{2}f''(y)(x-a)^2,$$

where y is between x and a. Now suppose that a is a local minimum for f, so $f'(a) = 0$. Also, suppose that x is close to a, so y is close a, and so $f''(y) \approx f''(a)$. Then we have

$$f(x) \approx f(a) + \frac{1}{2}f''(a)(x-a)^2. \tag{2}$$

Note that $f(a) + \frac{1}{2}f''(a)(x-a)^2$ is quadratic as a function of x. Taylor's theorem shows that near a local minimum f looks parabolic, with the minimum $x = a$ at the bottom of the parabola.

Now consider what happens when a minimization algorithm starts zeroing in on a minimum for $f(x)$ at $x = a$. If x is close to a then $x - a$ is small and in equation (2) we find that $(x-a)^2$ is VERY small, and hence so is $f''(a)(x-a)^2$. In fact, if $x - a$ is small enough then so far as the computer is concerned we find that

$$f(x) \approx f(a) + \frac{1}{2}f''(a)(x-a)^2 = f(a). \tag{3}$$

In the computer's view there's no difference between $f(x)$ and $f(a)$, and so with such finite precision we may as accept x as the minimum, since it's indistinguishable from a.

This can be quantified: from equation (3) we'll have $f(x) = f(a)$ if

$$\frac{1}{2}|f''(a)|(x-a)^2 < \epsilon|f(a)|,$$

or

$$|x-a| < \sqrt{\frac{2\epsilon|f(a)|}{|f''(a)|}}.$$

If $f(a)$ and $f''(a)$ are about the same size (so $|f(a)|/|f''(a)| \approx 1$, not uncommon) then we find that $f(x)$ and $f(a)$ are indistinguishable for $|x-a| < \sqrt{2\epsilon}$. The implication for optimization is that we should only iterate until $|x-a| \approx \sqrt{\epsilon}$, for any further precision is of no value. In single precision this means that locating a minimum to within 10^{-4} is the best possible, or 10^{-8} in double precision. For Maple doing ten significant figure math, $\epsilon = 5 \times 10^{-10}$ and the best tolerance we should aim for is thus about 2.2×10^{-5}.

Fun Problem

- Use Maple to plot the function $f(x) = 2 - \cos(x)$ on intervals of the form $(-\epsilon, \epsilon)$ for various values of ϵ. This function has a minimum at $x = 0$. How small can ϵ get before the minimum is invisible (the graph goes flat)? Try the same experiment on the interval $(1-\epsilon, 1+\epsilon)$ (in which f does not have a minimum). What's going on?

Other 1D Optimization Algorithms

Another approach to minimizing a function $f(x)$ is based on interpolation. Suppose we have a bracketing triple $[a, b, c]$. Construct a quadratic polynomial $p(x) = mx^2 + nx + r$ (a parabola) through the points $(a, f(a))$, $(b, f(b))$, $(c, f(c))$. It's easy to write out m, n, and r in terms of a, b, c, and f. Now find the minimum of this polynomial (why must it be a minimum and not a maximum?) The minimum occurs when $p'(x) = 2mx + n = 0$, so $x = -n/(2m)$. This x value becomes our new "test point" d, just as in the Golden Section algorithm. We evaluate $f(d)$ and based on that construct a new bracketing triple. Note that this approach does not require derivatives of f.

The idea behind quadratic interpolation is that any function which is twice differentiable can be well approximated by a polynomial—in this case we use a quadratic polynomial—near a point. This is just Taylor's Theorem. Thus if the points in our bracketing triple $[a, b, c]$ are all close together then in the vicinity of these points $f(x) \approx p(x)$, and so the minimum of p should approximate the minimum of f.

As it stands this interpolation approach leaves something to be desired. First, when a, b, and c are very close together the process of fitting a parabola becomes numerically rather unstable due to round-off error. It has to be programmed rather carefully. Actually, even if the length of the bracketing interval $c - a$ is large, the new test point $x = d$ may be very close to either a or c, the ends of the bracketing interval. This makes it hard to fit a parabola through the three points at the next iteration of the algorithm, again due to numerical instability and round-off error. Also, if the new test point at $x = d$ is close to $x = b$ and both are close to the minimum then due to round off error we will not be able to accurately compare $f(b)$ and $f(d)$—they'll look about the same. As a result we can actually make the wrong choice for the bracketing interval as "lose" our bracket.

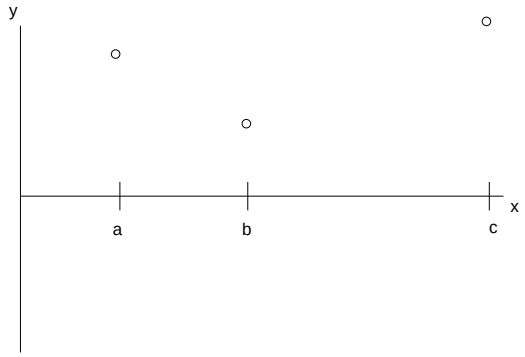
Finally, with the parabolic method as it stands it is difficult to know when to terminate, for it may be the case that although a and b get close to the minimum, c may not budge from iteration to iteration. As a result $c - a$ will not get small and we won't know with what accuracy we've trapped the minimum.

A better algorithm might implement a refined strategy: Use parabolic interpolation, but if the new test point at $x = d$ is too close (as quantified by some tolerance) to a, b, or c, take a Golden Section step instead; we never evaluate f with $\sqrt{\epsilon}$ distance of a point at which we've already evaluated. Also, if $c - a$ is not getting small with each iteration, again go to tried-and-true Golden Section for an iteration. This is one of the idea's behind Brent's algorithm, described below.

Using Derivative Information

So far the techniques we've looked at do not use $f'(x)$. Let's consider how that information could be incorporated into an optimization algorithm.

Here is a variation on the Golden Section which makes limited use of derivative information. Consider the situation below, in which we have a bracketing triple $[a, b, c]$



Compute $f'(b)$. It is rather obvious that if $f'(b) > 0$ then f must have a minimum somewhere in $[a, b]$, whereas if $f'(b) < 0$ then there is a minimum in $[b, c]$. Based on $f'(b)$ we will choose our new test point $x = d$ using the Golden Section strategy, but instead of picking the largest of $[a, b]$ or $[b, c]$, we'll choose the interval in which to place d based on the sign of $f'(b)$. This ought to provide some small improvement on the straight Golden Section method.

Another variation would be to make use of derivative information in parabolic interpolation. Given a bracket $[a, b, c]$, compute $f'(b)$. If $f'(b) > 0$ then there is a minimum in $[a, b]$; construct a quadratic polynomial $p(x) = mx^2 + nx + r$ which passes through $(a, f(a))$, $(b, f(b))$, and such that $p'(b) = f'(b)$ (so p agrees with f at two points and p' agrees with f' at one point.) We hope p is a good approximation to f, and hence $d = -n/(2m)$ (the minimum of p) should be a good approximation to the minimum of f. We compute $f(d)$ and choose a new bracketing triple, either $[a, d, b]$ or $[d, b, c]$. If $f'(b) < 0$ above then we do the same on the interval $[b, c]$.

Yet another approach uses cubic polynomials and derivative information to interpolate. Let's say that an interval $[a, b]$ brackets a minimum of f if $f'(a) < 0$ and $f'(b) > 0$ (draw a picture). There must be a minimum somewhere in $[a, b]$. Construct a cubic polynomial $p(x)$ with $p(a) = f(a), p(b) = f(b), p'(a) = f'(a), p'(b) = f'(b)$. We hope this polynomial is a good approximation to f, and $p(x)$ must have a minimum (in fact, only one) in $[a, b]$. Find this minimum, which is easy to do in closed form—it's the root of a quadratic, and use it as a

new test point d. Evaluate $f'(d)$. If $f'(d) < 0$ then choose $[a, d]$ as a new bracket, otherwise choose $[d, b]$.

One algorithm that is widely used for 1D optimization is Brent's algorithm. We won't go into all the gory details, but the main idea is to use parabolic interpolation to locate a test point at which to evaluate f; the algorithm always maintains a bracket on the minimum. If the test point constructed by interpolation turns out to be unsuitable (e.g., too close to a point at which f has already been evaluated) then we take a Golden Section step instead. The algorithm is optimized to make as few functions evaluations per iteration as possible, and wring as much information out of previous function evaluations as possible.

Orders of Convergence

Consider starting the Golden Section method on a function f with a bracketing triple $[a, b, c]$. Suppose f has a minimum at $x = x^*$. The analysis we did above for this method shows that (asymptotically, anyway) at the nth iteration of the algorithm we have trapped the minimum in an interval of length $(c-a)Z^n$, where $Z = (\sqrt{5}-1)/2 \approx 0.618$. If we let z_n be the midpoint of the bracketing interval at the nth iteration then we're guaranteed that

$$|z_n - x^*| \leq (c-a)Z^n. \tag{4}$$

This kind of convergence is called *linear* convergence, for we always have

$$|z_{n+1} - x^*| \leq Z|z_n - x^*|.$$

The error at the $n+1$ iteration is at most 62 percent of the error at the nth iteration.

However, it can be shown that (under ideal circumstances) the parabolic interpolation approach has order of convergence about 1.324, which means that (asymptotically in n)

$$|z_{n+1} - x^*| \leq C|z_n - x^*|^{1.324}$$

where C is some constant. For the cubic interpolation approach the convergence is quadratic, meaning

$$|z_{n+1} - x^*| \leq C|z_n - x^*|^2$$

for some C. It's easy to see that higher order convergence leads to the minimum in much fewer iterations. The drawback with the higher order methods is that they are finicky (as alluded to above) and so are usually combined with a slower but more robust algorithm like Golden Section.