The Golden Section Search Method

The purpose of this document is to show the derivation of the golden section search method to find the minimum of a unimodal continuous function over an interval without using derivatives. You can find the algorithm on page 412 of our textbook [2].

1 Derivation of the Method of Golden Section Search

Consider a function f over the interval $[a, b]$. We assume that $f(x)$ is continuous over $[a, b]$; and that $f(x)$ is *unimodal* over $[a, b]$, i.e.: $f(x)$ has only one minimum in $[a, b]$. Note that the method applies as well to finding the maximum.

The conditions above remind us to the bisection method and we will apply a similar idea: narrow the interval that contains the minimum comparing function values. In designing the method we seek to satisfy two goals. We want an optimal reduction factor for the search interval and minimal number of function calls. With these goals in mind we are going to determine the location to evaluate the function.

One choice inspired by the bisection method would be to compute the midpoint $m = (a + b)/2$ and to evaluate at x_1 and x_2, defined by $x_1 = m - \delta/2$ and $x_2 = m + \delta/2$, for some small value δ for which $f(x_1) \neq f(x_2)$. If $f(x_1) < f(x_2)$, then we are left with $[a, x_1]$, otherwise $[x_2, b]$ is our new search interval. While this halves the search interval in each step, we must take two new function evaluation in each step. This is not optimal.

So we only want to perform one new function evaluation in each step. Furthermore, we want a constant reduction factor, say c, for the size of the interval.
For x_1 and x_2 somewhere in $[a, b]$, there are two cases:

1. If $f(x_1) < f(x_2)$, then $[a, b] := [a, x_2]$, with interval size reduction

$$x_2 - a = c(b - a) \Rightarrow x_2 = a + cb - ca \Rightarrow x_2 = (1 - c)a + cb. \tag{1}$$

2. If $f(x_1) > f(x_2)$, then $[a, b] := [x_1, b]$, with interval size reduction

$$b - x_1 = c(b - a) \Rightarrow -x_1 = cb - ca - b \Rightarrow x_1 = ca + (1 - c)b. \tag{2}$$

Thus, once we know c, we know the location of x_1 and x_2. Without loss of generality, we focus on the case $f(x_1) < f(x_2)$. For the ease of calculation, take $[a, b] = [0, 1]$.

If $f(x_1) < f(x_2)$, then we recycle $x_1 = 1 - c$ and have to determine where to evaluate next, either at the left, or at the right of $1 - c$.

1. Suppose we place a new function evaluation at the left of $x_1 = 1 - c$, then x_1 is the right point of the interval $[0, c]$, and we write x_1 in two ways (using the formula for x_2 derived above with $a = 0, b = c$):

$$1 - c = (1 - c)0 + cc \Rightarrow c^2 + c - 1 = 0 \tag{3}$$

   The positive root leads to $c = (-1 + \sqrt{5})/2$, which equals approximately 0.6180.

2. Suppose we place a new function evaluation at the right of $x_1 = 1 - c$, then x_1 is the left point of the interval $[0, c]$, and we write x_1 in two ways (using the formula for x_1 derived above with $a = 0, b = c$):

$$1 - c = c0 + (1 - c)c \Rightarrow (1 - c)^2 = 0 \tag{4}$$

   The (double) root of this equation is 1, which gives no reduction, so we exclude this possibility.

Thus we have two rules. If $f(x_1) < f(x_2)$, we keep x_1, which becomes x_2, and compute a new x_1 using (2). If $f(x_1) > f(x_2)$, we keep x_2, which becomes x_1, and compute a new x_2 using (1).

2 Executing the method

Below is a simple function (save as gss.m) to run the golden section search method:

```
function [a,b] = gss(f,a,b,eps,N)
%
%  Performs golden section search on the function f.
%  Assumptions: f is continuous on [a,b]; and
%               f has only one minimum in [a,b].
%  No more than N function evaluations are done.
%  When b-a < eps, the iteration stops.
%
%  Example: [a,b] = gss('myfun',0,1,0.01,20)
%
c = (-1+sqrt(5))/2;
x1 = c*a + (1-c)*b;
fx1 = feval(f,x1);
x2 = (1-c)*a + c*b;
fx2 = feval(f,x2);
fprintf('-----------------------------------------------------\n');
fprintf('    x1        x2        f(x1)      f(x2)      b - a\n');
fprintf('-----------------------------------------------------\n');
fprintf('%.4e %.4e %.4e %.4e %.4e\n', x1, x2, fx1, fx2, b-a);
for i = 1:N-2
   if fx1 < fx2
      b = x2;
      x2 = x1;
      fx2 = fx1;
      x1 = c*a + (1-c)*b;
      fx1 = feval(f,x1);
   else
      a = x1;
      x1 = x2;
      fx1 = fx2;
      x2 = (1-c)*a + c*b;
      fx2 = feval(f,x2);
   end;
   fprintf('%.4e %.4e %.4e %.4e %.4e\n', x1, x2, fx1, fx2, b-a);
   if (abs(b-a) < eps)
      fprintf('succeeded after %d steps\n', i);
      return;
   end;
end;
fprintf('failed requirements after %d steps\n', N);
```

To run it, we need a function (save as myfun.m):

```
function y = myfun(x)
%
%  returns y = e^x - 2*x
%
y = exp(x) - 2*x;
```

Then we use gss.m in a session with MATLAB or Octave like

```
>> [a,b] = gss('myfun',0,1,0.01,20)
-----------------------------------------------------
    x1        x2        f(x1)      f(x2)      b - a
-----------------------------------------------------
3.8197e-01 6.1803e-01 7.0123e-01 6.1921e-01 1.0000e+00
6.1803e-01 7.6393e-01 6.1921e-01 6.1884e-01 6.1803e-01
7.6393e-01 8.5410e-01 6.1884e-01 6.4106e-01 3.8197e-01
7.0820e-01 7.6393e-01 6.1393e-01 6.1884e-01 2.3607e-01
6.7376e-01 7.0820e-01 6.1408e-01 6.1393e-01 1.4590e-01
7.0820e-01 7.2949e-01 6.1393e-01 6.1504e-01 9.0170e-02
6.9505e-01 7.0820e-01 6.1371e-01 6.1393e-01 5.5728e-02
6.8692e-01 6.9505e-01 6.1374e-01 6.1371e-01 3.4442e-02
6.9505e-01 7.0007e-01 6.1371e-01 6.1375e-01 2.1286e-02
6.9194e-01 6.9505e-01 6.1371e-01 6.1371e-01 1.3156e-02
6.9002e-01 6.9194e-01 6.1372e-01 6.1371e-01 8.1306e-03
succeeded after 10 steps

a =

    0.6869


b =

    0.6950

>>
```

References

[1] Ward Cheney and David Kincaid. *Numerical Mathematics and Computing.* Third Edition, Brooks/Cole Publishing Company, 1994.

[2] Curtis F. Gerald and Patrick O. Wheatley. *Applied Numerical Analysis.* Seventh Edition, Addison-Wesley, 2004.

[3] Floyd Hanson. MCS 471 Class Optimization Notes: Method of Golden Section Search. Available at http://www.math.uic.edu/~hanson/mcs471/classnotes.html.