Golden Section Search

Very similar to a bisection search (the key difference is that we are looking for a minimum instead of a zero)

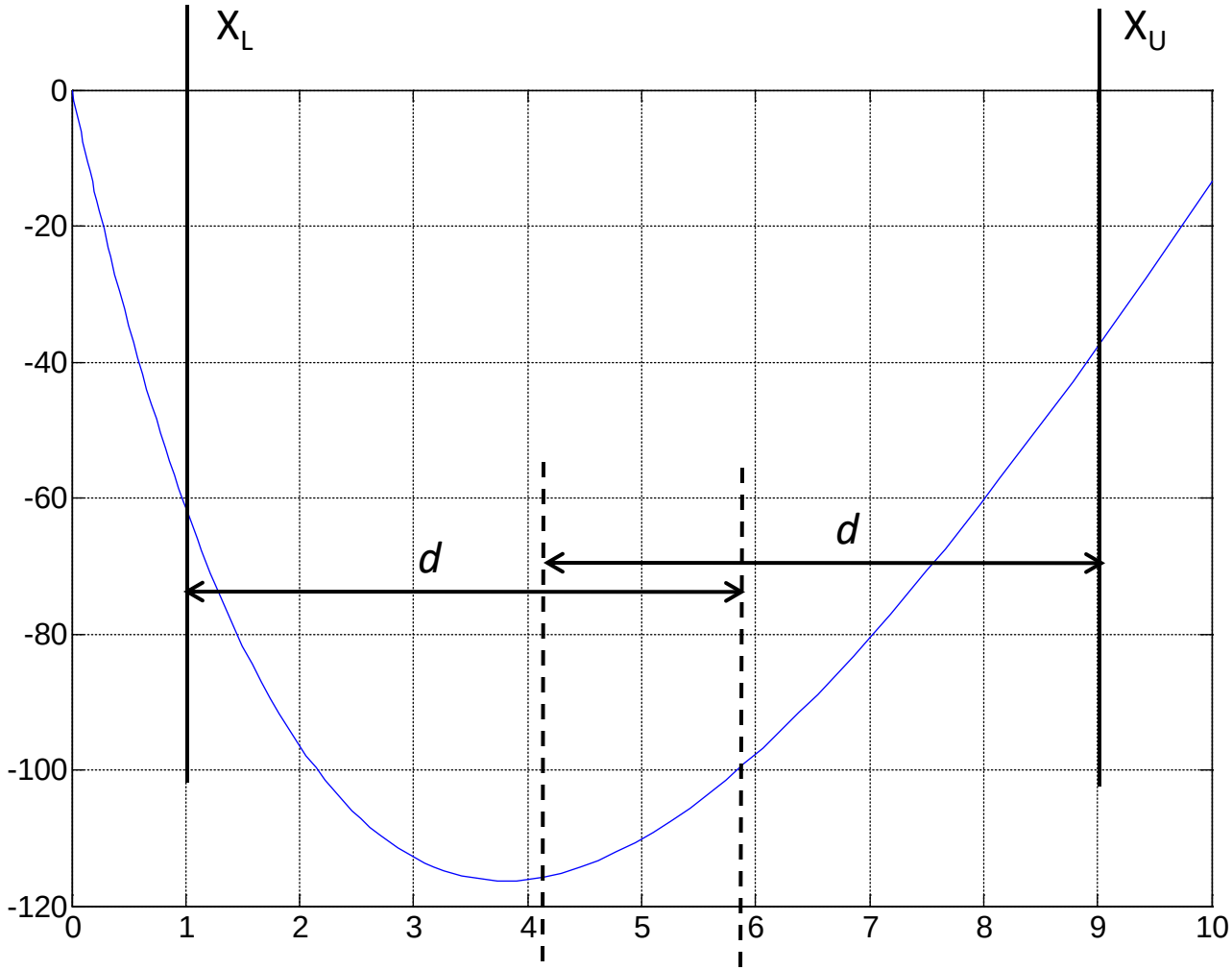
A bracketing method (to get started we need a pair of *x* values that bracket the solution).

At each iteration the interval containing the solution gets smaller and smaller.

By continuing until the interval becomes appropriately small any desired degree of accuracy can be achieved.

Want a method which minimizes the number of function evaluations

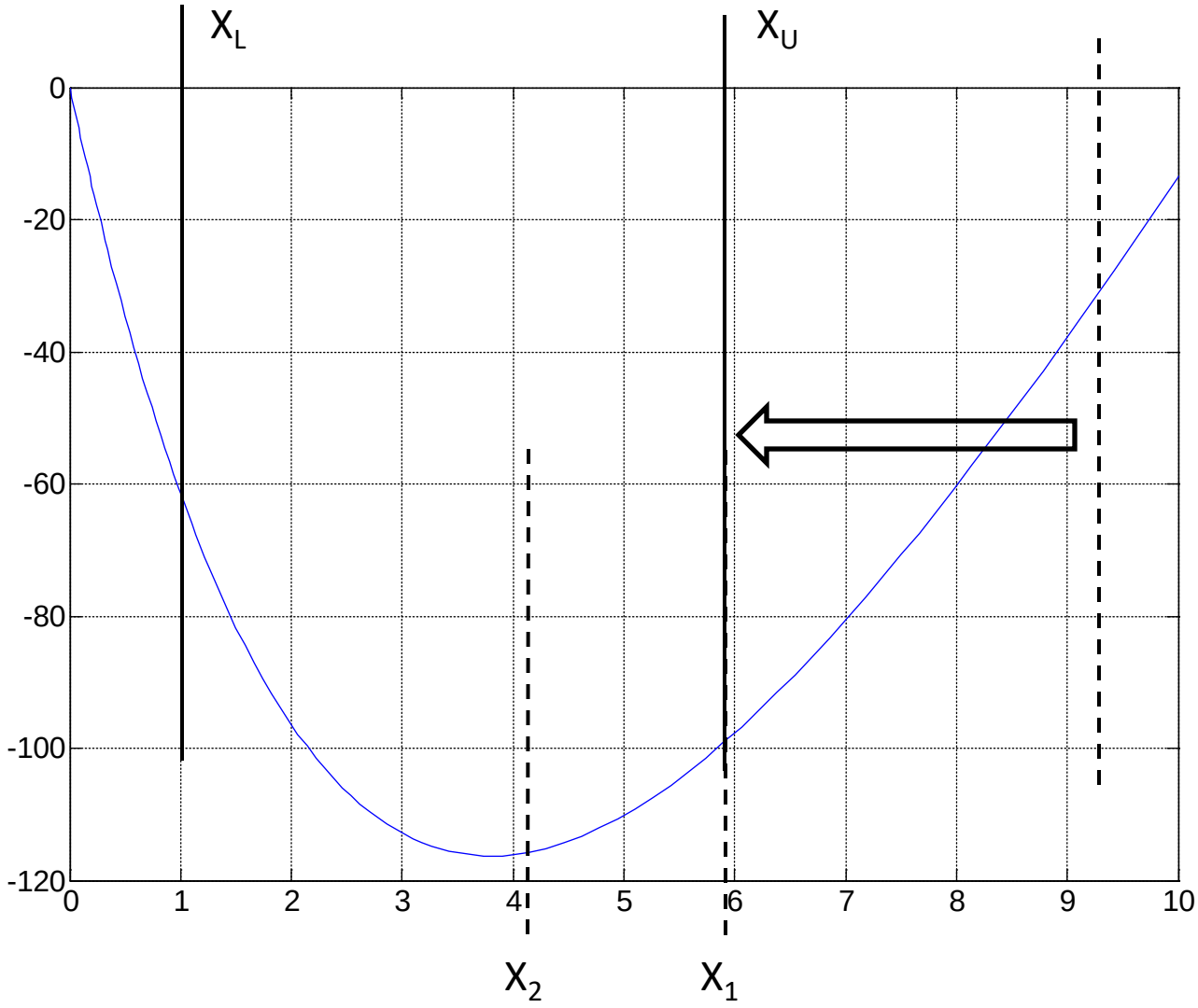
Step 1: Pick two interior points by applying the rules shown



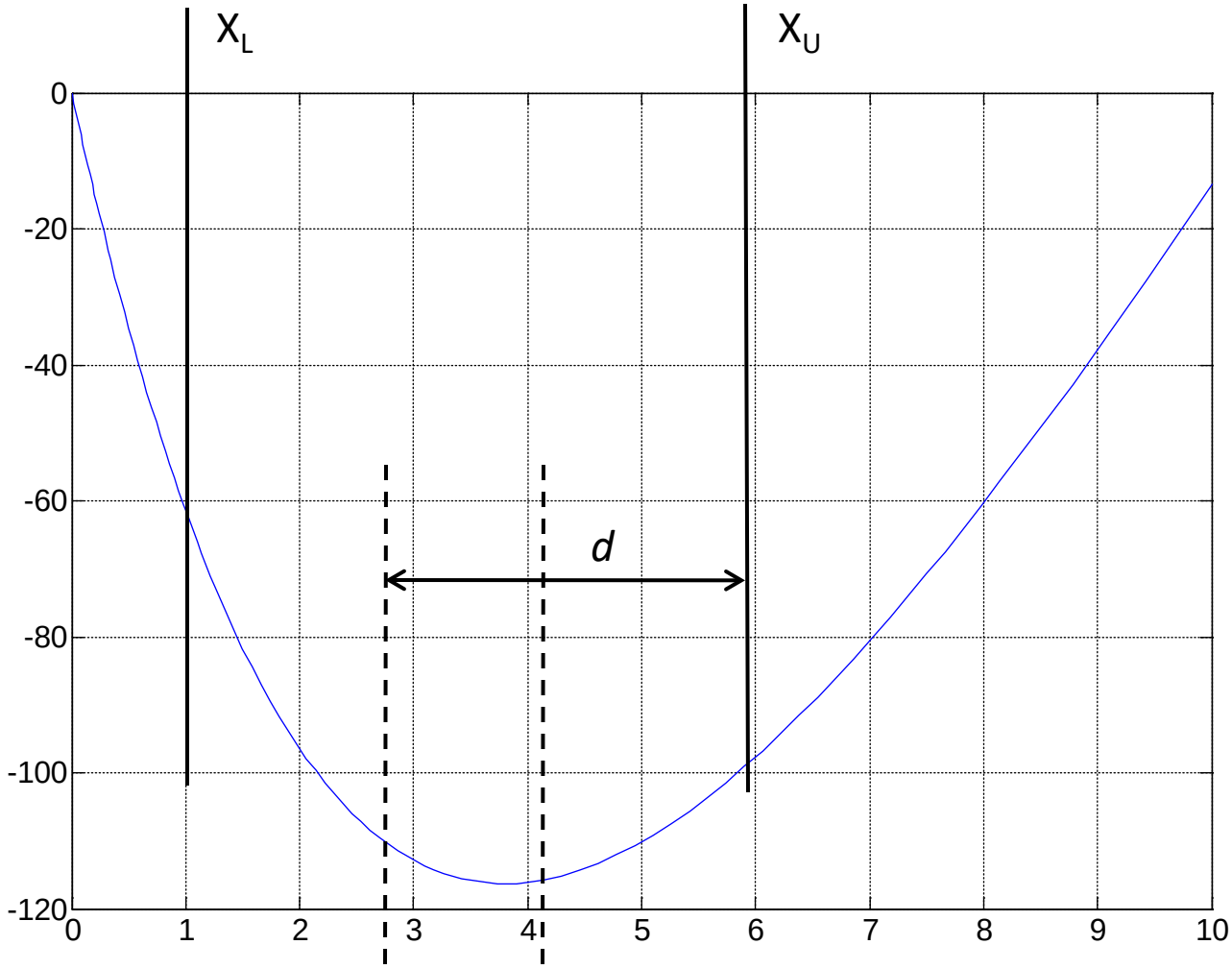
$X_2 = X_U - d \quad X_1 = X_L + d$

$$d = (\phi - 1)(X_U - X_L) \qquad \phi = \frac{1+\sqrt{5}}{2} = 1.6180339887...$$

Step 2: Move one of the two "walls" in. If $f(X_1) > f(X_2)$, X_U gets moved to X_1. Otherwise X_L gets moved to X_2.



Step 3: Renumber the unused interior point (X_2 becomes X_1 and vice versa) and use the formulae from step 1 to create whichever interior point is missing.



$X_2 = X_U - d$ X_1 (was X_2)

$$d = (\phi - 1)(X_U - X_L) \qquad \phi = \frac{1+\sqrt{5}}{2} = 1.6180339887...$$

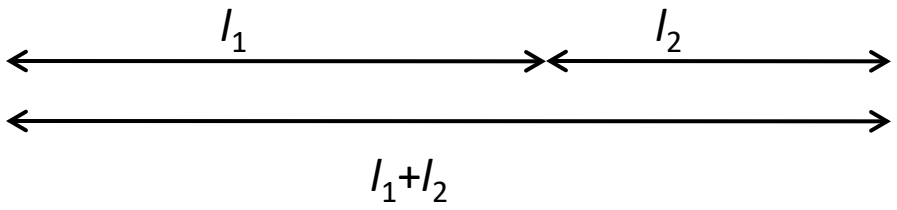
This gets us back to where we were at the end of step 1.

Steps 2 and 3 are now repeated until X_L and X_U become acceptable close to each other (i.e. until the minimum is located to the desired degree of accuracy).

The key feature of the algorithm is that the interior point that does not become a "wall" (step 2) is reused as an interior point on the next iteration.

There is only one new point (and hence only one function evaluation) per iteration.

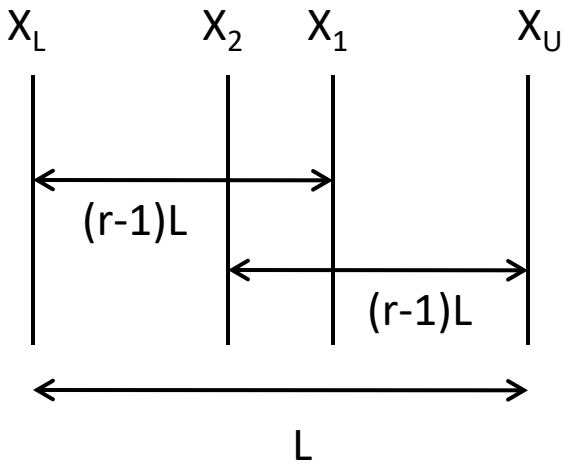
This magic works because ϕ (the *Golden Ratio*) is used in positioning the interior points.



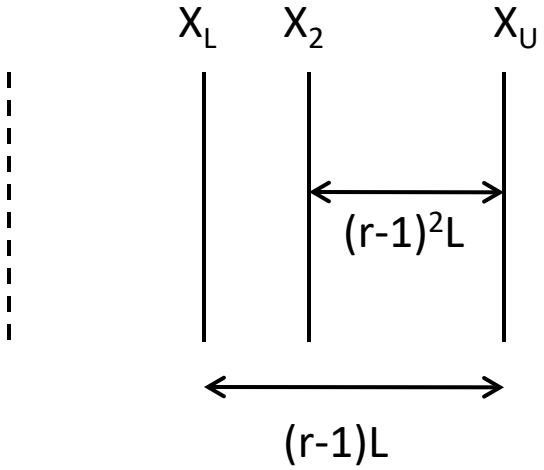
The *Golden Ratio* is the ratio of l_1 to l_2 that makes

$$\frac{l_1}{l_2} = \frac{l_1 + l_2}{l_1}$$

Solving this equation gives: $\frac{l_1}{l_2} = \phi = \frac{1+\sqrt{5}}{2} = 1.6180339887...$



Assume that the original interval is *L*.



Assume that X_L is moved to X_2 (as the situation is symmetrical the other possible assumption gives exactly the same result)

The original X_2 becomes X_1 and must be correctly positioned.

From the diagrams: $(r-1)^2 L + (r-1)L = L$

Dividing through by *L* and solving for the positive root gives: $r = \dfrac{1+\sqrt{5}}{2} = \phi$

Every iteration reduces the interval containing the solution by a factor of (ϕ-1) = 0.6180339887…

Not as good as a bisection search (which halves the interval at every iteration) but still pretty good.

Let Δx^0 be the original interval width.

Interval width after *n* iterations = $\Delta x^0 (\phi - 1)^{n-1}$

If the midpoint of the interval is used as the final answer (a reasonable approach) the maximum error is half of the interval width.

Max error after n iterations = $E_{MAX} = \frac{\Delta x^0 (\phi - 1)^{n-1}}{2}$

Number of iterations required to reduce E_{MAX} to $E_{DESIRED}$ =

$$n = \frac{\ln\left(\frac{2E_{DESIRED}}{\Delta x^0}\right)}{\ln(\phi - 1)} + 1 = \frac{\ln\left(\frac{2E_{DESIRED}}{\Delta x^0}\right)}{-0.481212} + 1 = -2.078087 \ln\left(\frac{2E_{DESIRED}}{\Delta x^0}\right) + 1$$

Notes on text:

The discussion of error in the text makes pretty heavy weather of what is really a pretty simple concept.

Instead of using the interval midpoint as the final answer the text takes whichever internal point is not about to become one of X_L and X_U.

The error formula assumes that X_L and X_U have yet to be updated (i.e. start of iteration values are assumed) but the Matlab code updates them before applying the formula (i.e. end of iteration values are used). This is a mistake.

As usual the text uses relative error (the error is divided by the magnitude of the solution).

The Matlab code in the text completely negates the advantages of the Golden Section search. Both interior points are recalculated on every iteration and the function is evaluated twice per iteration.

If one is going to do this it would make more sense to locate the interior points just on either side of the interval midpoint (and so get rid of nearly half of the interval on every iteration).

```
function x = golden (f, xL, xU, Edes, display)
% GOLDEN  Finds a minimum by performing a golden section search.
% Inputs: f = a function of one variable
%         xL = lower bound of region containing minimum
%         xU = upper bound of region containing minimum
%         Edes = function stops when x within Edes of minimum
%         display = display option (0 = no display (default),  1 = display)
% Outputs: x - estimate of minimum

if nargin < 5; display = 0; end

p1 = ((1 + sqrt(5)) / 2) - 1;  % golden ratio - 1

if display
  fprintf ...
  ('    k     xL          x2          x1          xU          Emax\n');
end

% set up for first iteration
x1 = xL + p1 * (xU - xL); fx1 = f(x1);
x2 = xU - p1 * (xU - xL); fx2 = f(x2);
```

```
for k = 1 : 1000

   Emax = (xU - xL) / 2;
   if display
    fprintf ('%5d %12.6f %12.6f %12.6f %12.6f %12.6f\n', k, xL, x2, x1, xU, Emax);
   end
   if Emax <= Edes
      x = (xL + xU) / 2; return;
   end

   if fx2 < fx1
      xU = x1; x1 = x2; fx1 = fx2; % old x2 becomes new x1
      x2 = xU - p1 * (xU - xL); fx2 = f(x2); % brand new x2 required
   else
      xL = x2; x2 = x1; fx2 = fx1; % old x1 becomes new x2
      x1 = xL + p1 * (xU - xL); fx1 = f(x1); % brand new x1 required
   end

end

error ('Golden section search has not converged.');

end
```

Example Golden Section Search:

```
>> f = @(x) -5 * x^3 + 115.3 * x^2 - 700 * x + 757.5;

>> minx = golden (f, 2, 8, 0.001, 1)
    k      xL           x2           x1           xU           Emax
    1     2.000000     4.291796     5.708204     8.000000     3.000000
    2     2.000000     3.416408     4.291796     5.708204     1.854102
    3     3.416408     4.291796     4.832816     5.708204     1.145898
    4     3.416408     3.957428     4.291796     4.832816     0.708204
    5     3.957428     4.291796     4.498447     4.832816     0.437694
    6     3.957428     4.164079     4.291796     4.498447     0.270510
    7     3.957428     4.085145     4.164079     4.291796     0.167184
    8     4.085145     4.164079     4.212862     4.291796     0.103326
    9     4.085145     4.133929     4.164079     4.212862     0.063859
   10     4.133929     4.164079     4.182712     4.212862     0.039467
   11     4.133929     4.152562     4.164079     4.182712     0.024392
   12     4.152562     4.164079     4.171196     4.182712     0.015075
   13     4.152562     4.159680     4.164079     4.171196     0.009317
   14     4.159680     4.164079     4.166797     4.171196     0.005758
   15     4.159680     4.162398     4.164079     4.166797     0.003559
   16     4.159680     4.161360     4.162398     4.164079     0.002199
   17     4.161360     4.162398     4.163040     4.164079     0.001359
   18     4.162398     4.163040     4.163437     4.164079     0.000840
minx =
    4.1632
```

fminbnd Details:

Options may be specified (details as for *fzero*).
fminbnd uses a combination golden section search and parabolic interpolation (see text)

```
>> opts = optimset('display', 'iter');
>> tstar = fminbnd (g, 2, 6, opts)  % objectHeight example from last lecture

Func-count     x          f(x)         Procedure
    1        3.52786    -115.931       initial
    2        4.47214    -114.266       golden
    3        2.94427    -112.317       golden
    4        3.83069    -116.306       parabolic
    5        3.81155    -116.309       parabolic
    6        3.80199     -116.31       parabolic
    7        3.80136     -116.31       parabolic
    8        3.8014      -116.31       parabolic
    9        3.80143     -116.31       parabolic

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-004
Tstar = 3.8014
```