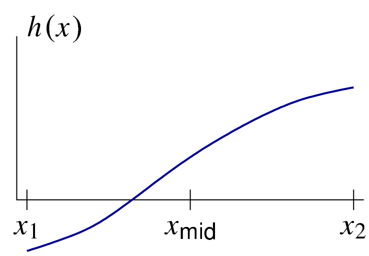
Bisection search

- To use full cell model to find $i_{\max,n}^{\text{dis,volt}}$, seek u_n to solve
$$0 = h\big(x_n[k + k_{\Delta T}], u_n\big) - v_{\min}$$
- To use full cell model to find $i_{\min,n}^{\text{chg,volt}}$, or seek u_n to solve
$$0 = h\big(x_n[k + k_{\Delta T}], u_n\big) - v_{\max}$$
- That is, we require a method to solve for a root of a nonlinear equation
- Here, we use the bisection search algorithm to do so

Bisection search

- Bisection search algorithm looks for a root of $h(x)$ (*i.e,* value of x such that $h(x) = 0$) where it is known *a priori* that at least one root lies between values $x_1 <$ root $< x_2$
  - Can know root lies in interval if sign of $h(x_1)$ different from sign of $h(x_2)$
- Each iteration of the bisection algorithm evaluates the function at the midpoint $x_{\text{mid}} = (x_1 + x_2)/2$
- Based on the sign of the evaluation, either x_1 or x_2 is replaced by x_{mid} to retain different signs on $h(x_1)$ and $h(x_2)$
- The root-location uncertainty is halved by this algorithmic step



Bisection search

- Bisection iteration repeated until interval between x_1 and x_2 as small as desired: if ε is desired resolution, algorithm requires at most $\lceil \log_2(|x_2 - x_1|/\varepsilon) \rceil$ iterations
- The following code segment is beginning of bisect.m function, ensures that root is between x_1 and $x_1 + \Delta x$

```
% Search interval x1...x2 in fn h(.) for root, with tolerance tol
function x = bisect(h,x1,x2,tol)
  jmax = ceil(log2(abs(x2-x1)/tol));
  dx = x2 - x1; % set the search interval dx = x2 - x1
  if( h(x1) >= 0 )
    dx = -dx; x1 = x2;     % root now b/w (x1,x1 + dx), and h(x1) < 0
  end
```

Bisection search

- Remaining code loops at most `jmax` times, dividing search interval in half each iteration

```
  for jj = 1:jmax
    dx = 0.5 * dx; xmid = x1 + dx;
    if h(xmid) <= 0,
      x1 = xmid;
    end
  end
  x = x1 + 0.5*dx;
end
```

- Special case: if $h(x_1)$ and $h(x_2)$ have same sign initially, bisection returns $x \approx x_2$
- An example of how to run this algorithm is (returns `-9.5367e-07`):

```
h = @(x) x^3;
bisect(h,-1,2,1e-5)
```


Summary

- Need a nonlinear search algorithm to find root to nonlinear equation to solve for voltage-based current limits
- The cell model is “linear enough” that a simple bisection search works well
- You have learned how the bisection algorithm works
- You have also seen how to write a bisection search in Octave