Incremental Search (incsearch.m)

```
function [ brackets ] = incsearch ( f, min, max, points )
% INCSEARCH: locates roots by incremental search
% Inputs: f = a function of one variable (need not be able to deal with vectors)
%           min = lower bound of range to be searched
%           max = upper bound of range to be searched
%           points = number of search steps
% Outputs: brackets(i, 1) = lower bound for ith bracket
%              brackets(i, 2) = upper bound for ith bracket
%              **** if no brackets found, brackets = [] ****

nb = 0; brackets = [];   % brackets is initially 0 by 0
x = linspace (min, max, points);
flo = f(x(1));
for i = 2: points
   fhi = f(x(i));
   if sign(flo) ~= sign(fhi)
      nb = nb + 1;
      brackets(nb, 1) = x(i - 1);  brackets(nb, 2) = x(i);
   end
   flo = fhi;
end
end
```

The script below finds and outputs all roots of *f* between 0 and 20.

```
f = @(x) 50 * sin(0.5 * x) - x.^2 - 17 * x + 60;  % from last problem

brackets = incsearch ( f, 0, 20, 100 );

[n m] = size(brackets); % note vector on left hand side

if n == 0
   fprintf ('No roots found.\n');
else
   for k = 1 : n
      x = fzero(f, brackets(k, :));  % select whole row
      fprintf ('There is a root at x = %f\n', x);
   end
end
```

Output:

```
There is a root at x = 1.583780
There is a root at x = 6.636039
There is a root at x = 12.825489
There is a root at x = 16.266248
```

Bisection Search (basic idea)

start with x_{LOW} (less than root) and x_{HIGH} (greater than root)

while true

    pick x_{ROOT} half way between x_{LOW} and x_{HIGH}

    if termination conditions satisfied
        stop
    endif

    if $f(x_{\mathrm{MID}})$ has same sign as $f(\mathrm{x}_{\mathrm{LOW}})$
        $x_{\mathrm{LOW}} = x_{\mathrm{ROOT}}$
    else
        $x_{\mathrm{HIGH}} = x_{\mathrm{ROOT}}$
    endif

endwhile

Possible Termination Conditions:

1/. The error in x_{ROOT} has become acceptably small (i.e. we have got close enough to the actual root).

2/. The function at x_{ROOT} is zero or acceptably close to zero.

3/. Some maximum number of iterations has been reached.

Condition (1) is the normal termination condition.

Condition (2) makes senses in situations in which accuracy in x is unimportant as long as $f(x)$ is close enough to zero.

Condition (3) is really more applicable to iterative processes that can go wrong. As the bisection process is essentially guaranteed to succeed there is no real need to guard against disaster.

Max possible error = $E_{\text{MAX}} = (x_{\text{HIGH}} - x_{\text{LOW}}) / 2$

Reduces by a factor of two on every iteration.

Let Δx^0 be the original interval width.

Max error after n iterations $= \dfrac{\Delta x^0}{2^n}$

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

$$\log_2\left(\frac{\Delta x_0}{E_{\text{DESIRED}}}\right) = \log\left(\frac{\Delta x_0}{E_{\text{DESIRED}}}\right) / \log 2 = \ln\left(\frac{\Delta x_0}{E_{\text{DESIRED}}}\right) / \ln 2$$

Bisection search unusual in that the number of iterations required to achieve a desired accuracy can be predicted in advance.

Also unusual in that x_{ROOT} can be guaranteed to be within some amount of the correct answer.

Text Notes (1)

Text uses "approximate error" = $E_A = |x_{ROOT}^{NEW} - x_{ROOT}^{OLD}|$

Small changes in x_{ROOT} are assumed to mean that error is correspondingly small

For a bisection search E_A and E_{MAX} are in fact exactly the same thing.

$|x_{ROOT}^{NEW} - x_{ROOT}^{OLD}|$ is always equal to $(x_{HIGH} - x_{LOW})$ / 2

Assume $x_{ROOT}^{OLD} = x_{LOW}$

Then $\left|x_{ROOT}^{NEW} - x_{ROOT}^{OLD}\right| = \left|x_{LOW} - \dfrac{x_{HIGH} + x_{LOW}}{2}\right| = \left|\dfrac{x_{LOW} - x_{HIGH}}{2}\right| = \dfrac{x_{HIGH} - x_{LOW}}{2}$

Assume $x_{ROOT}^{OLD} = x_{HIGH}$

Then $\left|x_{ROOT}^{NEW} - x_{ROOT}^{OLD}\right| = \left|x_{HIGH} - \dfrac{x_{HIGH} + x_{LOW}}{2}\right| = \dfrac{x_{HIGH} - x_{LOW}}{2}$

Text Notes (2):

Text also works with “relative” errors.

Relative error = absolute error / magnitude of x_{ROOT}

The idea is that a given error is less significant when dealing with larger numbers.

An error of 0.01 is insignificant if x_{ROOT} is 100000 but matters if x_{ROOT} is 1.

Relative error undefined when x_{ROOT} is zero, problematic when x_{ROOT} is small.

Bisection Search (bisect.m)

```
function [xr] = bisect (f, xl, xh, Edes, display)
% BISECT  Finds a root by performing a bisection search.
% Inputs: f = a function of one variable
%         xl = lower bound of range to be searched
%         xh = upper bound of range to be searched
%         **** f(a) and f(b) must have different signs ****
%         Edes = tolerance in x (function stops when xr is guaranteed to
%                          be within Edes of a root)
%         display = display option (0 = no output, 1 = output, defaults to 0)
% Outputs: xr = estimate of root

if nargin < 5; display = 0; end

yl = f(xl);  yh = f(xh);
if sign(yl) == sign(yh), error ('no sign change'), end

if display
  fprintf ('  step       xl          xh          xr          yr          Emax\n');
end

signOfYl = sign(yl);  % keep track of sign of function at xl
```

```
for k = 1:1000        % 1000 max iterations

    xr = (xl + xh) /2; yr = f(xr);  Emax = (xh - xl) / 2;

    if display
        fprintf ('%5d   %12.6f %12.6f %12.6f %12.6f %12.6f\n', k, xl, xh, xr, yr, Emax);
    end

    if Emax <= Edes || yr == 0  % error acceptably small or direct hit
        return;
    end

    if sign(yr) == signOfYl
        xl = xr;
    else
        xh = xr;
    end

end

error ('Bisection search has not converged');   % most unlikely

end
```

Bisection with a Casio Calculator

Example: root of $f(x) = x^3-5$ between 1 and 2

Store low limit in memory "A" (1 SHIFT STO A)
Store high limit in memory "B" (2 SHIFT STO B)
Enter function using M as *x* (ALPHA M SHIFT x^2 – 5)
Evaluate at low limit (CALC 1 =), remember sign at low limit
Set memory M = (A + B) /2 ((ALPHA A + ALPHA B) / 2) SHIFT STO M)
while true
  Use up arrow to recall function
  Evaluate function at M (CALC =)
  if sign of result same as sign at low limit
    Store M in A (ALPHA M SHIFT STO A)
  else
    Store M in B (ALPHA M SHIFT STO B)
  endif
  Use up arrow to recall (A+B)/2 -> M
  Update M (=)
endwhile

varargin:

Bisection code in text uses *varargin*.

```
function [root, ea, iter] = bisect (func, xl, xh, es, maxit, varargin)
```

Allows for functions that require parameters in addition to the independent variable.

```
f = @(x, p1, p2) …….
```

Parameter values are tacked on to end of argument list when using *bisec*t.

```
root = bisect (f, 0, 10, 0.0001, 1000, 15, 2);  % 15 and 2 are values for p1 and p2
```

bisect passes additional values on to the function when function called

```
test = func (xl, varargin{:}) * func (xr, varargin{:});
```

For details see text section 3.5.3

False Position (aka Regula Falsi)

Similar to bisection but uses different rule for picking x_{ROOT}

$$x_{ROOT} = x_{HIGH} - \frac{f(x_{HIGH})(x_{LOW} - x_{HIGH})}{f(x_{LOW}) - f(x_{HIGH})}$$

instead of

$$x_{ROOT} = \frac{(x_{LOW} + x_{HIGH})}{2}$$

Advantage: Typically (but not always) faster than bisection

Disadvantage: Does not allow maximum possible error to be reduced to some chosen amount. Termination must be based on approximate error (stop when changes in x_{ROOT} from one iteration to the next become acceptably small).