Lecture 7 and 8

Recap

(1.) Complete definition of a function includes the function itself, domain of function, and set in which function values lie.

$$f : X \longrightarrow Y$$

f ↑ function defined for all $x \in X$

X: domain of function

Y: Co domain of function

Examples

(i) $f_1(x) = x^2$, $X = (-\infty, \infty)$, $Y = (-\infty, \infty)$

(ii) $f_2(x) = x^2$, $X = (-\infty, \infty)$, $Y = [0, \infty)$

(iii) $f_3(x) = x^2$, $X = [0, \infty)$, $Y = (-\infty, \infty)$

(iv) $f_4(x) = x^2$, $X = [0, \infty)$, $Y = [0, \infty)$

(v) $f_5(x) = \cos x$, $X = (-\infty, \infty)$, $Y = (-\infty, \infty)$

(vi) $f_6(x) = \cos x$, $X = (-\infty, \infty)$, $Y = [-1, 1]$

(vii) $f_7(x) = \cos^2 x$, $X = (-\infty, \infty)$, $Y = [0, 1]$

(2.) Domain, Codomain, range of a function $f : X \longrightarrow Y$

$\text{Dom}(f) =$ set of all x for which f is defined $= X$

$\text{Codom}(f) =$ set in which all possible values of f lie $= Y$

$\text{Rg}(f) =$ set of $f(x)$ for all $x \in X$

We have $\quad \text{Rg}(f) \subset \text{Codom}(f)$

$\uparrow$ subset

Examples of Range & Codomain

— for functions f_1 & f_2

$$Rg(f) = [0, \infty) \subset Codom(f) = (-\infty, \infty)$$

— for functions f_2 & f_4

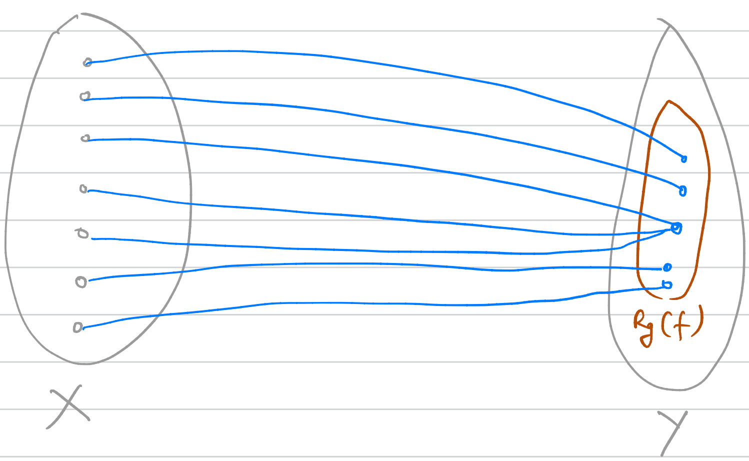
$$Rg(f) = [0, \infty) = Codom(f) = [0, \infty)$$

— for function f_5, f_6, f_7

$$Rg(f_5) = [-1, 1] \subset Codom(f_5) = (-\infty, \infty)$$

$$Rg(f_6) = [-1, 1] = Codom(f_6) = [-1, 1]$$

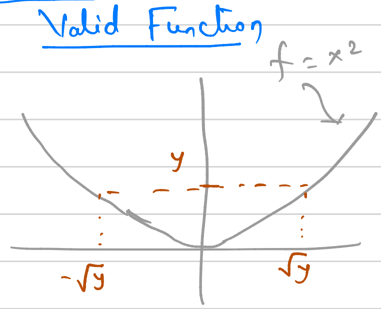
$$Rg(f_7) = [0, 1] = Codom(f_7) = [0, 1]$$



① Two different $x_1, x_2 \in X$ can have $f(x_1) = f(x_2)$

② But one point $x \in X$ can not have two values of $f(x)$

Valid Function



for $x_1 = -\sqrt{y}$, $x_2 = \sqrt{y}$

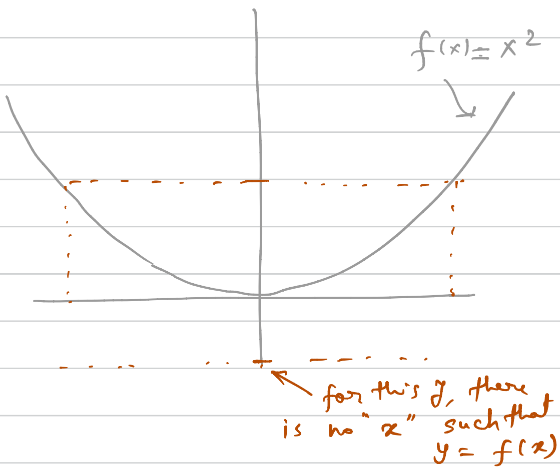
$f(x_1) = f(x_2) = y$

Not a function

x is mapped to two different points on circle

(3.) Types of functions

Surjective : $f : X \longrightarrow Y$ is surjective if for any $y \in Y$, there is a $x \in X$ such that $y = f(x)$



(Case 1) $f(x) = x^2$, $X = (-\infty, \infty)$
$Y = (-\infty, \infty)$

"not surjective"

(Case 2) $f(x) = x^2$, $X = (-\infty, \infty)$
$Y = [0, \infty)$

"surjective"

What changed from Case 1 to Case 2 ?

$\Downarrow$

Co dom (f)

injective : $f : X \longrightarrow Y$ is injective if for any $y \in Rg(f)$, there is a unique $x \in X$ such that

$y = f(x)$

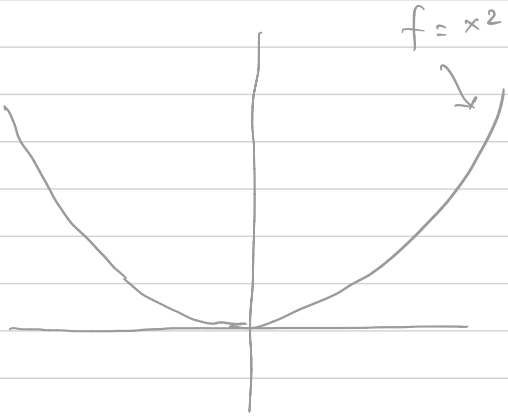
Compare definition of surjective and injective

(i) in surjective, we have "for any $\underline{y \in Y}$"

in injective, we have "for any $\underline{y \in Rg(f)}$"

(ii) in surjective, we have "there is $\underline{a}$ $x \in X$"

in injective, we have "there is $\underline{\text{a unique}}$ $x \in X$"



(Case 1) $f = x^2$, $X = (-\infty, \infty)$
$Y = (-\infty, \infty)$

"not injective", "not surjective"

(Case 2) $f = x^2$, $X = [0, \infty)$
$Y = (-\infty, \infty)$

"injective", "not surjective"

(Case 3) $f = x^2$, $X = [0, \infty)$

$Y = [0, \infty)$

"injective", "surjective"

all that we changed is either dom(f) or codom(f) !!

(4.) Closed and open intervals

(a, b) open on both sides

$x \in (a, b) \Rightarrow a < x < b$

— x can not be equal to a but it can get as close to a as you want

— x can not be equal to b but it can get as close to b as possible

$(a, b]$ open on left side

$[a, b)$ open on right side

$[a, b]$ closed on both sides, $a \leq x \leq b$

It matters in many situations whether the interval is open or closed

Example: $f: X \longrightarrow Y$, $f(x) = x^2$

Consider a problem: find $x \in X$ such that $f(x)=0$

Case 1: $X = (-\infty, \infty)$ Solution: $x = 0 \in (-\infty, \infty)$

Case 2: $X = (0, \infty)$ solution: there is no $x \in X$ such that $f(x) = 0$

"no solution"

Case 3: $X = [0, \infty)$ solution: $x = 0 \in X$

Consider another problem: find $x \in X$ such that $f(x) \leq f(z)$ for any $z \in X$

I.e. find $x \in X$ at which $f(x)$ is minimum

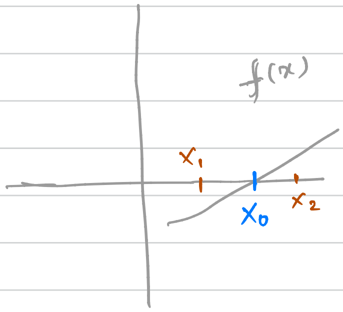
Case 1: $X = [0, \infty)$ solution $x = 0 \in X$

$f(x) = 0 \leq f(z) = z^2$ for any $z \in X$

Case 2: $X = (0, \infty)$ "no solution"

Roots problem: Given a function $f: X \longrightarrow Y$, find $x \in X$ such that $f(x) = 0$

Case 1

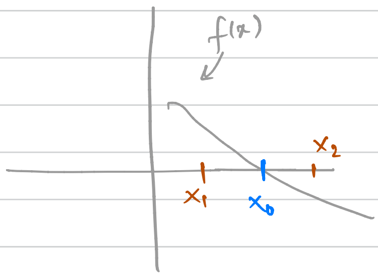


$$\underbrace{f(x_1) < 0, \quad f(x_2) > 0}$$

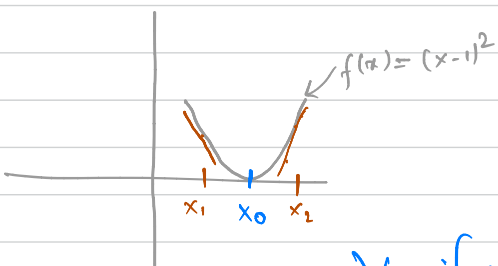
$$\boxed{f(x_1)\, f(x_2) < 0}$$ ⇐ for any functions of Case 1 or Case 2 type

$$\overbrace{f(x_1) > 0, \quad f(x_2) < 0}$$

Case 2



Case 3

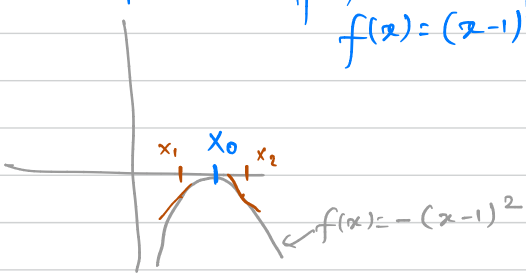


$f(x_1) > 0, \quad f(x_2) > 0$

$f'(x_1) < 0, \quad f'(x_2) > 0$

Verify for $f(x) = (x-1)^2$! $\Longrightarrow$ $\boxed{f'(x_1)\, f'(x_2) < 0}$ ⇐ for any function of Case 3 & 4

Case 4



$f(x_1) < 0, \quad f(x_2) < 0$

$f'(x_1) > 0, \quad f'(x_2) < 0$

Graphical method We already used graphical method in analyzing Case 1, 2, 3, 4 in previous page.

— It is extremely powerful in analyzing the properties of a function near point x_0 such that $f(x_0) = 0$

— The properties we observed will be used in developing numerical method

— But it is inefficient, can not be automated, and accuracy is limited

Bracketing method We consider problems with function that fall in either case 1 or case 2

Given two points $x_1, x_2 \in X$ with $x_1 < x_2$ and $f(x_1) f(x_2) < 0$

then there exist $x_0 \in [x_1, x_2]$ such that

$$f(x_0) = 0$$

Bracketing methods use above property to locate point x_0

There are several bracketing methods

(i) incremental search method

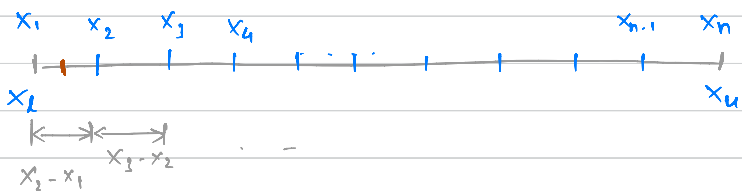
(ii) bisection method

(iii) false position method

Incremental search method Let $f: X \to Y = (-\infty, \infty)$ be a fuction.

Step 0: Suppose we know two numbers $x_u, x_l \in X$ such that $f(x_u)\, f(x_l) < 0$

Step 1: Divide $[a, b]$ in smaller uniformly sized intervals



$X_1 = X_l$

$X_n = X_u$

Such that $x_2 - x_1 = x_3 - x_2 = \cdots = x_n - x_{n-1}$

Step 2: For each $i = 1, 2, \ldots, n-1$

Check $f(x_i)\, f(x_{i+1}) < 0$

$\downarrow$

If true then because $x_0 \in [x_i, x_{i+1}]$ such that $f(x_0) = 0$

we store $\dfrac{x_i + x_{i+1}}{2}$

as one of the solution (we treat midpoint as solution!)

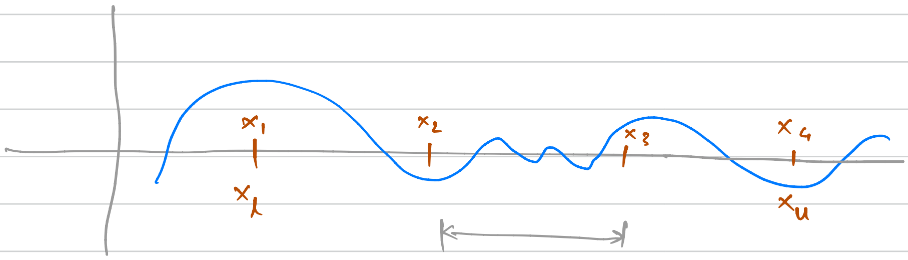
At the end we have list of $\dfrac{x_i + x_{i+1}}{2}$ for those i that satisfy $f(x_i)\, f(x_{i+1}) < 0$.

$\Downarrow$

$\boxed{\text{Our solution}}$

Limitations of incremental search

(i) For cases where multiple x_0 exists and they are close by



in this interval there are 5 x_0 such that $f(x_0)=0$ but the method will return only one solution from this interval $[x_2, x_3]$

$\Uparrow$

Remedy : Take intervals of smaller size

Controlling errors in incremental search method

By taking intervals of smaller and smaller size, we can get arbitrarily close to all the point x_0 such that $f(x_0)=0$

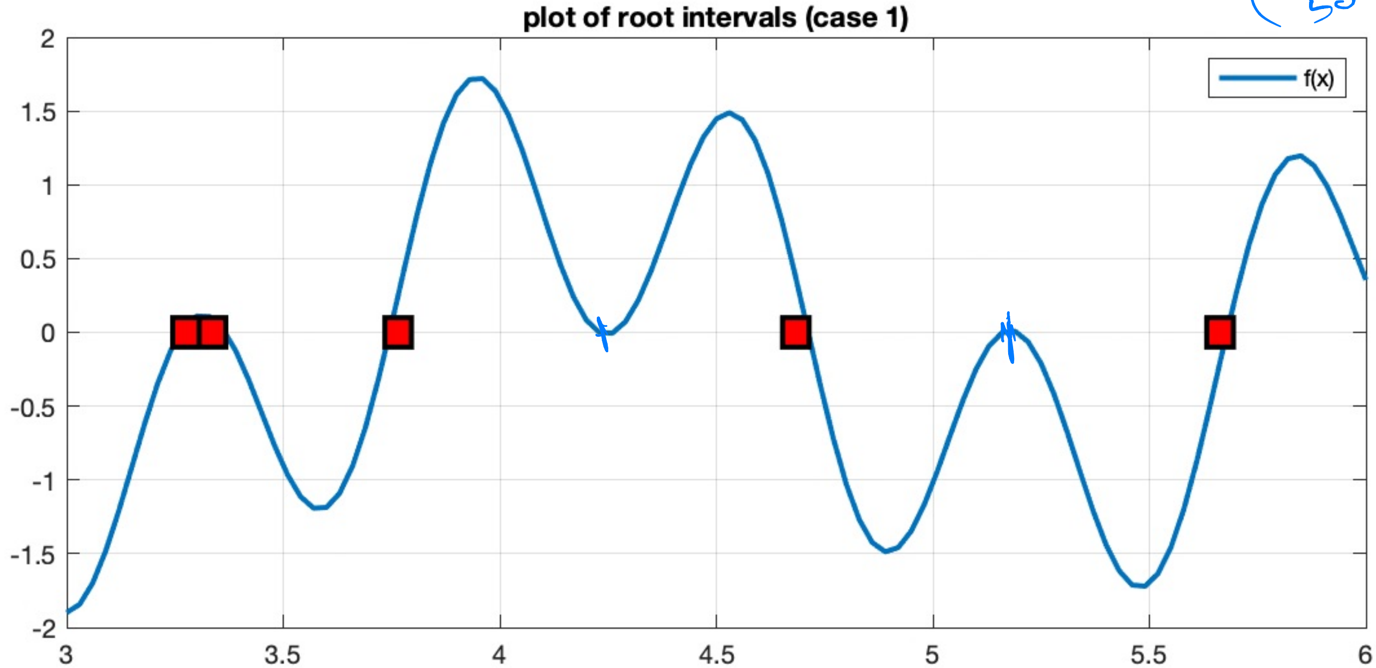
$\Downarrow$

we will look at errors more in next method

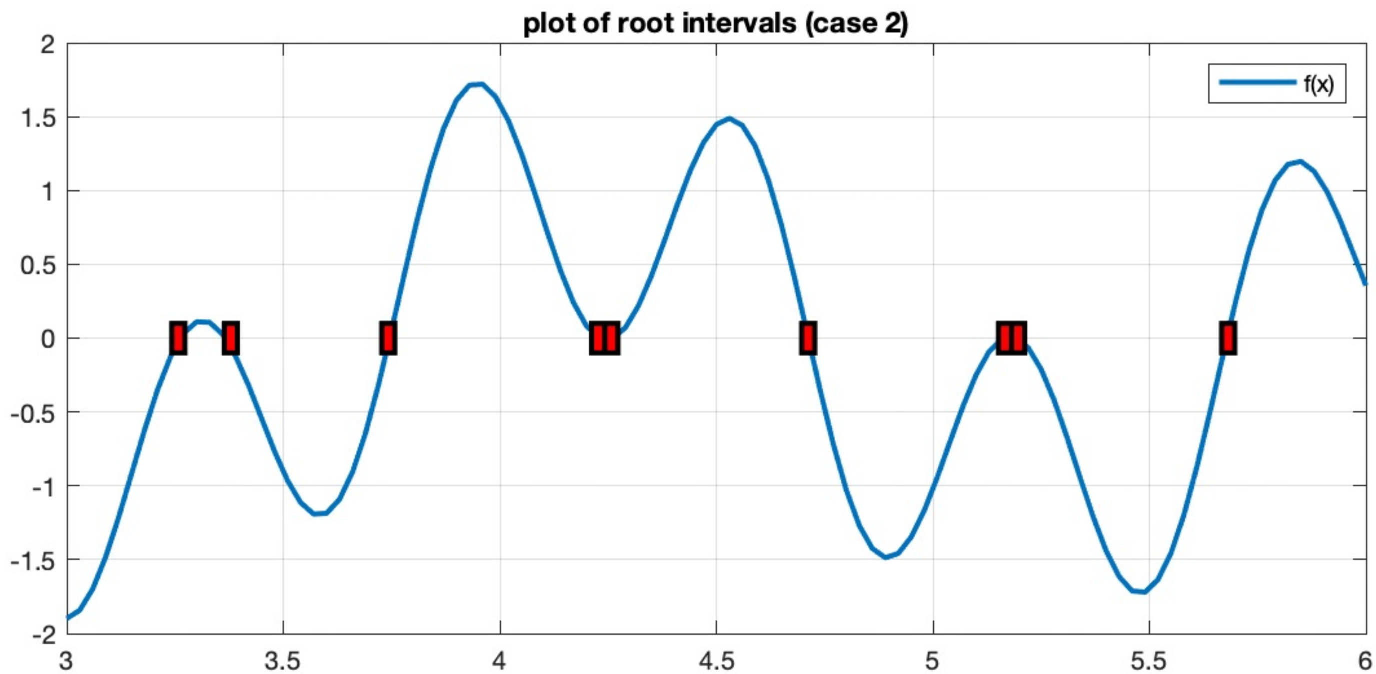
```matlab
function xb = incsearch(func,xmin,xmax,ns)
% incsearch: incremental search root locator
% syntax:
%    xb = incsearch(func,xmin,xmax,ns)
% input:
%    func = name of function
%    xmin, xmax = endpoints of interval
%    ns = number of subintervals (default = 50)
% output:
%    xb(k,1) is the lower bound of the kth sign change
%    xb(k,2) is the upper bound of the kth sign change
%    If no brackets found, xb = [].

% check if sufficient arguments are supplied to this function
if nargin < 3
    error('at least 3 arguments required')
end

%if ns blank set to 50
if nargin < 4
    ns = 50;
end

% Incremental search
x = linspace(xmin,xmax,ns);

% get all values of function
f = func(x);

% start search
nb = 0; xb = []; %xb is null unless sign change detected
for k = 1:length(x)-1
    %check for sign change
    if sign(f(k)) ~= sign(f(k+1))   ⇒   f(k)*f(k+1) < 0
        nb = nb + 1;
        xb(nb,1) = x(k);            ⇒   func(x(k))*func(x(k+1)) < 0
        xb(nb,2) = x(k+1);
    end
end
```

```matlab
% function
f = @(x) sin(10*x)+cos(3*x);

% lower and upper limits of arguments
xl = 3;
xu = 6;
x = xl:(xu-xl)/100:xu;
y = f(x);

% get intervals containing roots
n_intervals_1 = 50;
xb1 = incsearch(f, xl, xu, n_intervals_1);
disp('intervals containing roots')
disp(xb1)

% increase number of intervals (take smaller width intervals)
n_intervals_2 = 100;
xb2 = incsearch(f, xl, xu, n_intervals_2);
disp(' ')
disp('intervals containing roots')
disp(xb2)

% plotting
figure(1)
subplot(2, 1, 1)
plot(x, y, 'DisplayName', 'f(x)', 'LineWidth', 2)
grid on
hold on

for i=1:length(xb1)
    a = xb1(i, 1);
    b = xb1(i, 2);
    rectangle('Position', [a, -0.1, b-a, 0.2], 'LineWidth', 2, ...
        'FaceColor', 'r')
    hold on
end
legend()
title('plot of root intervals (case 1)')

subplot(2, 1, 2)
plot(x, y, 'DisplayName', 'f(x)', 'LineWidth', 2)
grid on
hold on

for i=1:length(xb2)
    a = xb2(i, 1);
    b = xb2(i, 2);
    rectangle('Position', [a, -0.1, b-a, 0.2], 'LineWidth', 2, ...
        'FaceColor', 'r')
    hold on
end
legend()
title('plot of root intervals (case 2)')
```

$\left(\frac{6-3}{50}\right)$



$\left(\frac{6-3}{100}\right)$



Bisection method

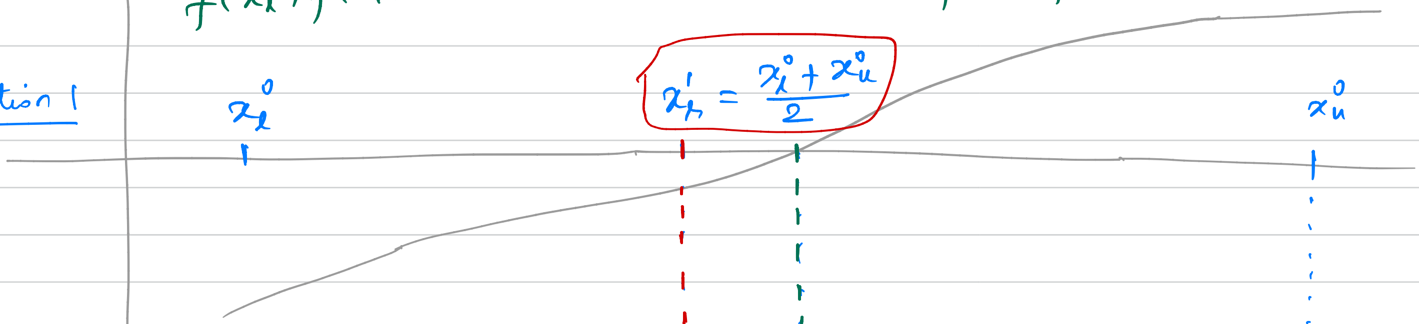
This method allows calculation of roots more rapidly. However, it can only compute one root!

$x_0 \Rightarrow f(x_0) = 0$

x^i

$f(x_l^0)\,f(x_r^1) < 0$ $\quad$ $f(x_r^1)\,f(x_u^0) < 0$

iteration 1 $\quad x_l^0 \quad \boxed{x_r^1 = \dfrac{x_l^0 + x_u^0}{2}} \quad x_u^0$



iteration 2 $\quad x_l^1 = x_r^1 \quad \boxed{x_r^2 = \dfrac{x_l^1 + x_u^1}{2}} \quad x_u^1 = x_u^0$

iteration 3 $\quad x_l^2 = x_l^1 \quad \boxed{x_r^3 = \dfrac{x_l^2 + x_u^2}{2}} \quad x_u^2 = x_r^2$

iteration 4 $\quad x_l^3 = x_l^2 \quad x_u^3 = x_r^3 \quad \boxed{x_r^4 = \dfrac{x_l^3 + x_u^3}{2}}$

iteration 5 $\quad x_l^4 = x_r^4 \quad x_u^4 = x_u^3 \quad \boxed{x_r^5 = \dfrac{x_l^4 + x_u^4}{2}}$

iteration 6 $\quad x_l^5 = x_l^4 \quad x_u^5 = x_r^5 \quad \boxed{x_r^6 = \dfrac{x_l^5 + x_u^5}{2}}$

"getting closer to the root"

error is decreasing with increasing iteration

But how is error defined?

At the end of each iteration "i", bisection method produces improved solution x_r^i

Error in bisection method

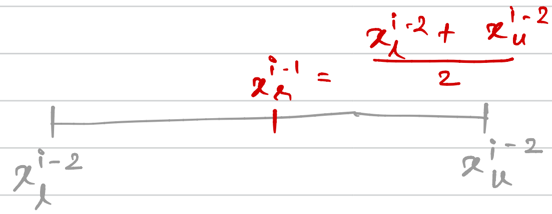
Let x_r^{i-1} root from iteration $i-1$ and x_l^{i-2} and x_u^{i-2}

such that $x_r^{i-1} = \dfrac{x_l^{i-2} + x_u^{i-2}}{2}$

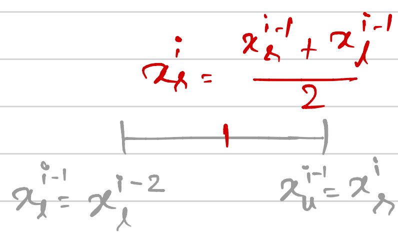
Let x_r^{i} root from iteration i and x_l^{i-1} and x_u^{i-1}

such that $x_r^{i} = \dfrac{x_l^{i-1} + x_u^{i-1}}{2}$

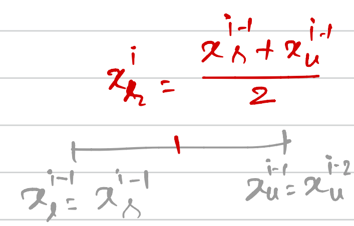
iter $i-1$



iter i



OR



Thus

$$E_i := \left| x_r^{i} - x_r^{i-1} \right|$$

$$= \left| \frac{x_l^{i-1} + x_r^{i-1}}{2} - x_r^{i-1} \right| \quad \text{or} \quad \left| \frac{x_u^{i-1} + x_r^{i-1}}{2} - x_r^{i-1} \right|$$

$$\boxed{E_i = \left| \frac{x_l^{i-1} - x_r^{i-1}}{2} \right| \quad \text{or} \quad \left| \frac{x_u^{i-1} - x_r^{i-1}}{2} \right|}$$

$$= \left| \frac{x_l^{i-2} - \frac{1}{2}(x_l^{i-2} + x_u^{i-2})}{2} \right| \quad \text{or} \quad \left| \frac{x_u^{i-2} - \frac{1}{2}(x_l^{i-2} + x_u^{i-2})}{4} \right|$$

$$= \frac{1}{4} \left| x_u^{i-2} - x_l^{i-2} \right| \quad \text{or} \quad \frac{1}{4} \left| x_u^{i-2} - x_l^{i-2} \right|$$

Next, note that

$$|x_u^i - x_l^i| = \frac{1}{2}\underbrace{|x_u^{i-1} - x_l^{i-1}|} = \frac{1}{2^2}|x_u^{i-2} - x_l^{i-2}| \dots$$

So

$$E_i = \frac{1}{2^2}|x_u^{i-2} - x_l^{i-2}| = \frac{1}{2^3}|x_u^{i-3} - x_l^{i-3}| = \frac{1}{2^4}|x_u^{i-4} - x_l^{i-4}|$$

$\Rightarrow$ $\boxed{E_i = \frac{1}{2^i}|x_u^0 - x_l^0|}$ where x_l^0, x_u^0 are initial interval we started

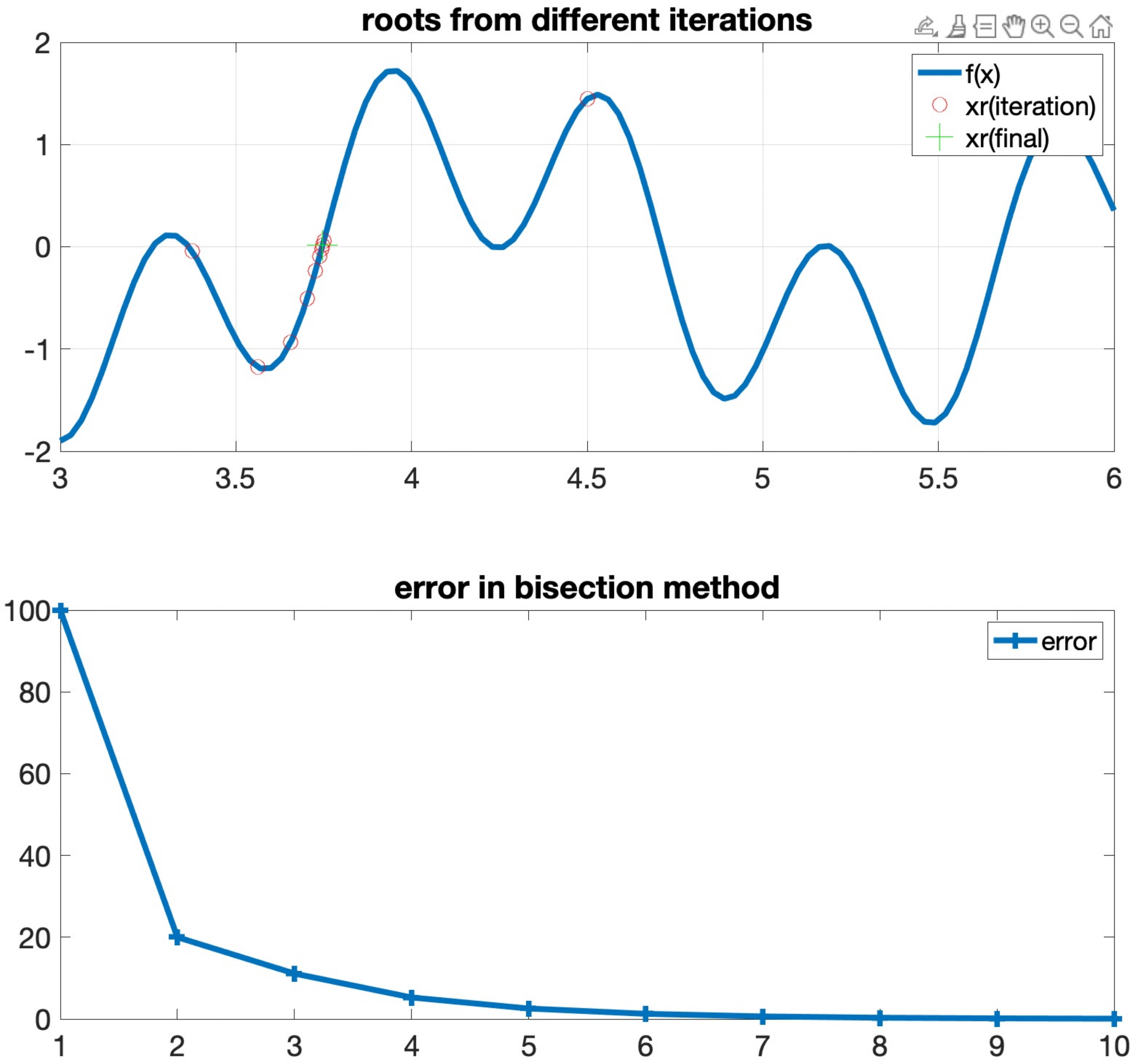
So error at iteration i is simply $\frac{1}{2^i}\Delta$ where Δ is the size of initial interval

Normalized error in bisecting method

$$e_a^i = \frac{E_a^i}{|x_r^i|} \times 100\%.$$

```matlab
function [xr, fxr, ea, iter] = bisect(func,xl,xu,ea_tol,maxit)
if nargin < 3
    error('at least 3 arguments required')
end
if nargin < 4
    maxit = 50;
end

xr = []; fxr = []; ea = [];
iter = 0; xl_new = xl; xu_new = xu; xr_new = 0; ea_new = 100;
while (1) % we terminate inside code
    iter = iter + 1;

    % reset old mid point
    xr_old = xr_new;
    xr_new = 0.5*(xl_new + xu_new);

    % set the new interval end points for next iteration
    xl_old = xl_new; xu_old = xu_new;

    % select either [xl_old, xr_new] or [xr_new, xu_old]
    f_product = func(xl_old)*func(xr_new);
    if f_product < 0 % left interval is selected
        xl_new = xl_old;
        xu_new = xr_new;
    else % right interval is selected
        xl_new = xr_new;
        xu_new = xu_old;
    end

    % compute error
    if iter > 1
        ea_new = abs(xr_new - xr_old) * 100 / abs(xr_new);
    end

    % save
    xr(iter) = xr_new; fxr(iter) = func(xr_new); ea(iter) = ea_new;

    % terminate
    if ea_new <= ea_tol || iter >= maxit
        break;
    end
end
```

```
% function
f = @(x) sin(10*x)+cos(3*x);

% lower and upper limits of arguments
xl = 3;
xu = 6;
x = xl:(xu-xl)/100:xu;
y = f(x);

% get root
maxit = 50;
ea_tol = 0.1;
[xr, fxr, ea, iter] = bisect(f, xl, xu, ea_tol, maxit);
disp('root')
disp(xr(end))

% plotting
ls = 4; ms = 10; ms2 = 20;
figure('DefaultAxesFontSize',20)
subplot(2, 1, 1)
plot(x, y, 'DisplayName', 'f(x)', 'LineWidth', ls)
grid on
hold on

% plot the solution at different iterations
plot(xr, fxr, 'ro', 'LineStyle', 'none', ...
    'DisplayName', 'xr(iteration)', 'MarkerSize', ms)
hold on
plot(xr(end), fxr(end), 'g+', ...
    'DisplayName', 'xr(final)', 'MarkerSize', ms2)
legend()
title('roots from different iterations')

subplot(2, 1, 2)
iter_i = 1:1:iter;
plot(iter_i, ea, '+-', 'DisplayName', 'error', ...
    'LineWidth', ls, 'MarkerSize', ms)
legend()
title('error in bisection method')
```

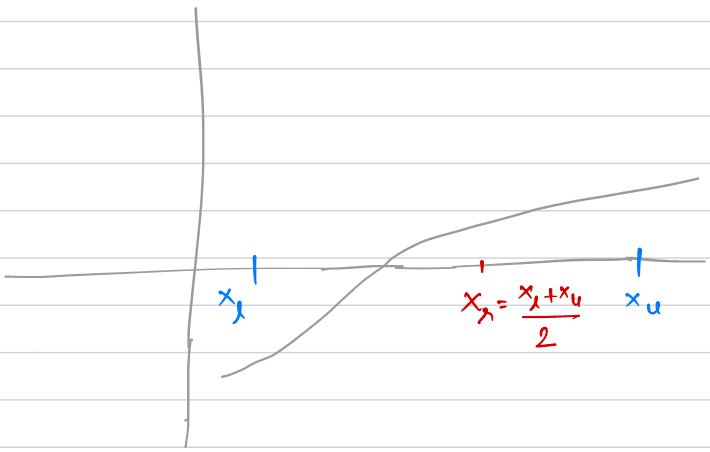


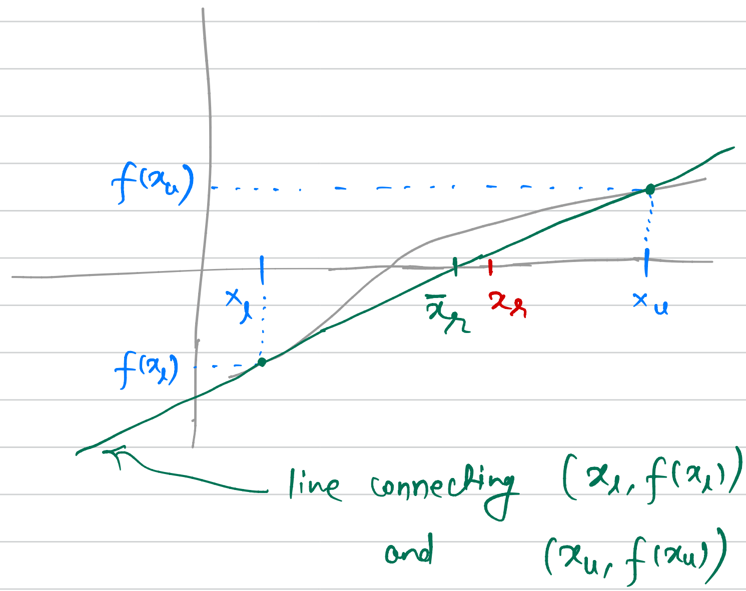
roots from different iterations
f(x)
xr(iteration)
xr(final)
2
1
0
-1
-2
3
3.5
4
4.5
5
5.5
6
error in bisection method
error
100
80
60
40
20
0
1
2
3
4
5
6
7
8
9
10

The false-position method

In bisection method, we take the mid point of interval as approximate root. False-position method instead uses clever way to get the approximate root

⇓





☐ Consider a line connecting $(x_l, f(x_l))$ and $(x_u, f(x_u))$

Equation of this line is

$$\frac{y - f(x_l)}{x - x_l} = \frac{f(x_u) - f(x_l)}{x_u - x_l}$$

$$\boxed{y = f(x_l) + (f(x_u) - f(x_l))\left(\frac{x - x_l}{x_u - x_l}\right)}$$

We can also have

$$\Rightarrow \frac{f(x_u) - y}{x_u - x} = \frac{f(x_u) - f(x_l)}{x_u - x_l}$$

$$y = f(x_u) - (f(x_u) - f(x_l))\left(\frac{x_u - x}{x_u - x_l}\right)$$

☐ $\bar{x}_r$ such that $y = 0$

$$\Rightarrow f(x_l) + (f(x_u) - f(x_l))\left(\frac{\bar{x}_r - x_l}{x_u - x_l}\right) = 0$$

$$\Rightarrow \boxed{\bar{x}_r = x_l + \frac{(x_u - x_l)}{(f(x_u) - f(x_l))}\left(-f(x_l)\right)}$$

OR equivalently

$$\bar{x}_r = x_u - f(x_u)\left(\frac{f(x_u) - f(x_l)}{x_u - x_l}\right)$$

⇓

Compare this with bisection solution

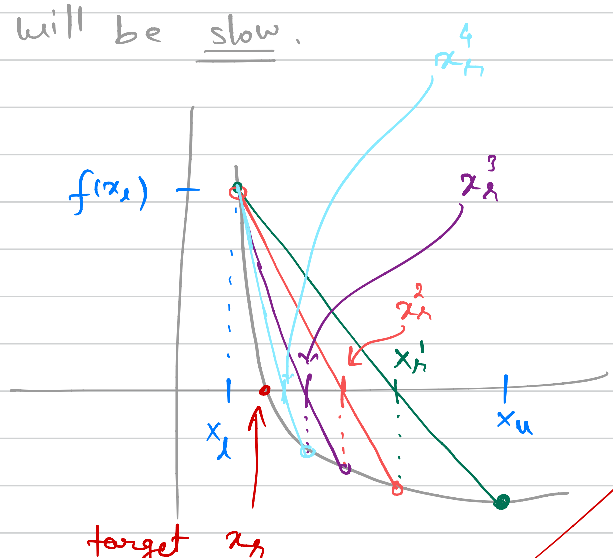
$$\boxed{x_r = \frac{x_l + x_u}{2}}$$

So false-position method uses values of function $f(x_l)$, $f(x_u)$ also to find the root $\bar{x}_r$.

Limitation of false-position method

for fuction with very large slope, the improvement in location of root after each iteration may not be substantial

OR in other words decay of error with iteration will be slow.



with each iteration,

(i) we are moving towards x_r

(ii) but the change from x_r^1 to x_r^2, x_r^2 to x_r^3, x_r^3 to x_r^4, ... is not large and the change is getting

(iii) the closer we get to x_r, the harder it becomes in further improving the solution

due to the fact that $f(x_l)$ is very large resulting in lines with very large slopes