Roots of Equations

Open Methods

(Part 2)

Edited by: Dr. wafaa El-Haweet

The following root finding methods will be introduced:

A. Bracketing Methods

A.1. Bisection Method

A.2. Regula Falsi

B. Open Methods

B.1. Fixed Point Iteration

B.2. Newton Raphson's Method

B.3. Secant Method

B.2. Secant Method

Newton-Raphson method needs to compute the derivatives.

The secant method approximate the derivatives by finite divided difference.

$$f'(x_i) \cong \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$

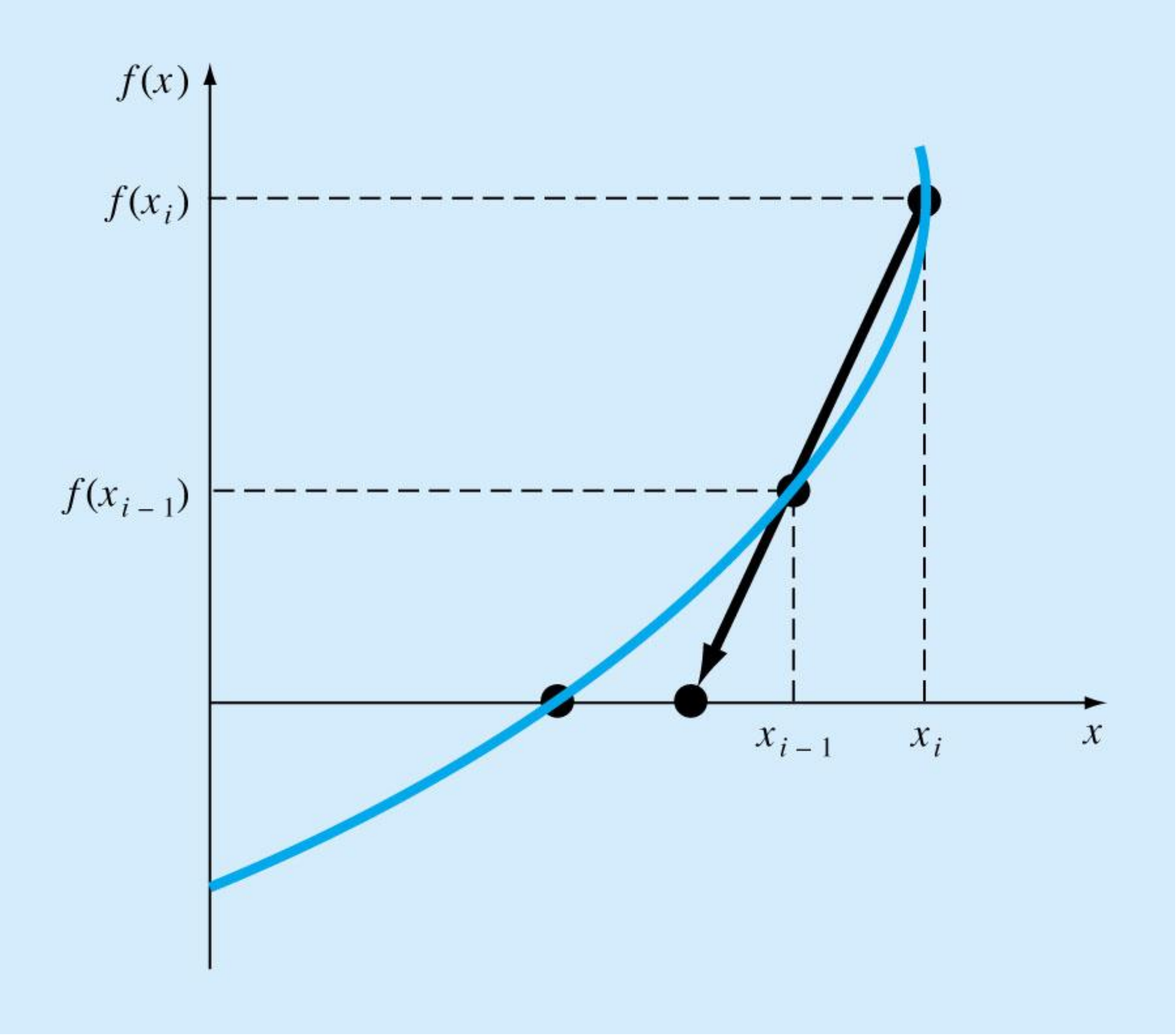
From Newton-Raphson method

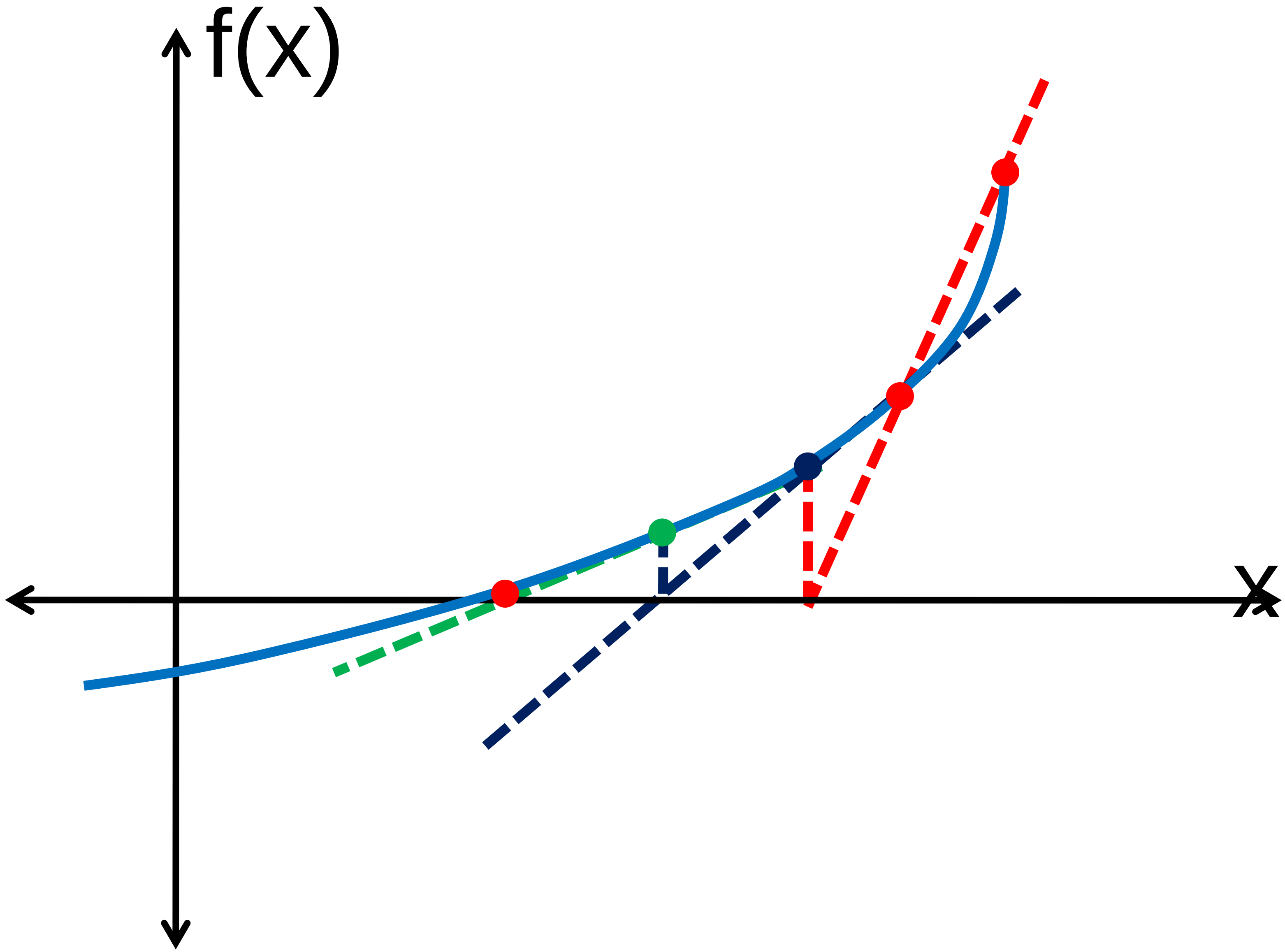
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

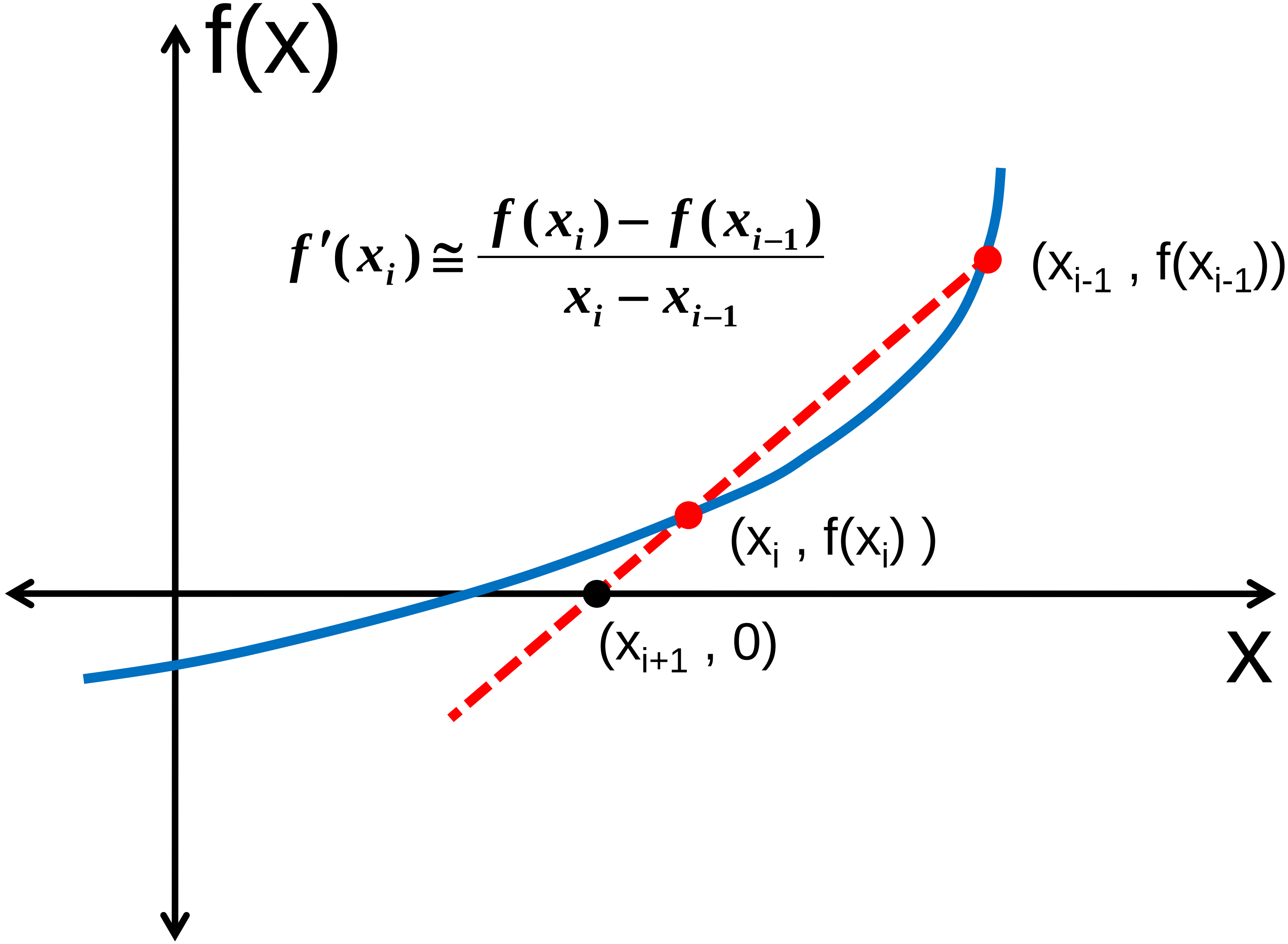
$$\cong x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

$$\boxed{x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}}$$

Secant Method




f(x)
x



f(x)
$f'(x_i) \cong \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$
$(x_{i-1}, f(x_{i-1}))$
$(x_i, f(x_i))$
$(x_{i+1}, 0)$
x

The Secant Method

$$x_{i-1} \Rightarrow given$$

$$x_i \Rightarrow given$$

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

Example

Solve: $Y=X^2 - 2$ Using Secant method

$\varepsilon_s \to \text{The Stop criterion} = 5\%$

Iteration 1

$$f_x = x^2 - 2$$

$$x_{i-1} = 0.5 \qquad f(x_{i-1}) = -1.75$$

$$x_i = 1.0 \qquad f(x_i) = -1$$

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} = 1 - (-1)\frac{1-0.5}{-1-(-1.75)} = 1.66$$

$$\varepsilon_a = \left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| \times 100\% = \left|\frac{1.66-1.0}{1.66}\right| 100\% = 39.8\%$$

Iteration 2

$$f_x = x^2 - 2$$

$$x_{i-1} = 1.0 \qquad f(x_{i-1}) = -1.0$$

$$x_i = 1.66 \qquad f(x_i) = 0.756$$

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} = 1.66 - (0.756)\frac{1.66 - 1}{0.756 - (-1.0)} = 1.376$$

$$\varepsilon_a = \left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| \times 100\% = \left|\frac{1.376 - 1.66}{1.376}\right| 100\% = 20.6\%$$

Iteration 3

$$f_x = x^2 - 2$$

$$x_{i-1} = 1.66 \qquad f(x_{i-1}) = 0.756$$

$$x_i = 1.376 \qquad f(x_i) = -0.11$$

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} = 1.376 - (-0.11)\frac{1.376 - 1.66}{-0.11 - \left(0.756\right)} = 1.41$$

$$\varepsilon_{\mathrm{a}} = \left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| \times 100\% = \left|\frac{1.41 - 1.376}{1.41}\right| 100\% = 2.4\%$$

Secant Method – Example

Find root of $f(x) = e^{-x} - x = 0$ with initial estimate of $x_{-1} = 0$ and $x_0 = 1.0$. (Answer: $\alpha = 0.56714329$)

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

i	x_{i-1}	x_i	$f(x_{i-1})$	$f(x_i)$	x_{i+1}	ε_t
0	0	1	1.00000	-0.63212	0.61270	8.0 %
1	1	0.61270	-0.63212	-0.07081	0.56384	0.58 %
2	0.61270	0.56384	-0.07081	0.00518	0.56717	0.0048 %

Again, compare this results obtained by the Newton-Raphson method and simple fixed point iteration method.

Comparison of the Secant and False-position method

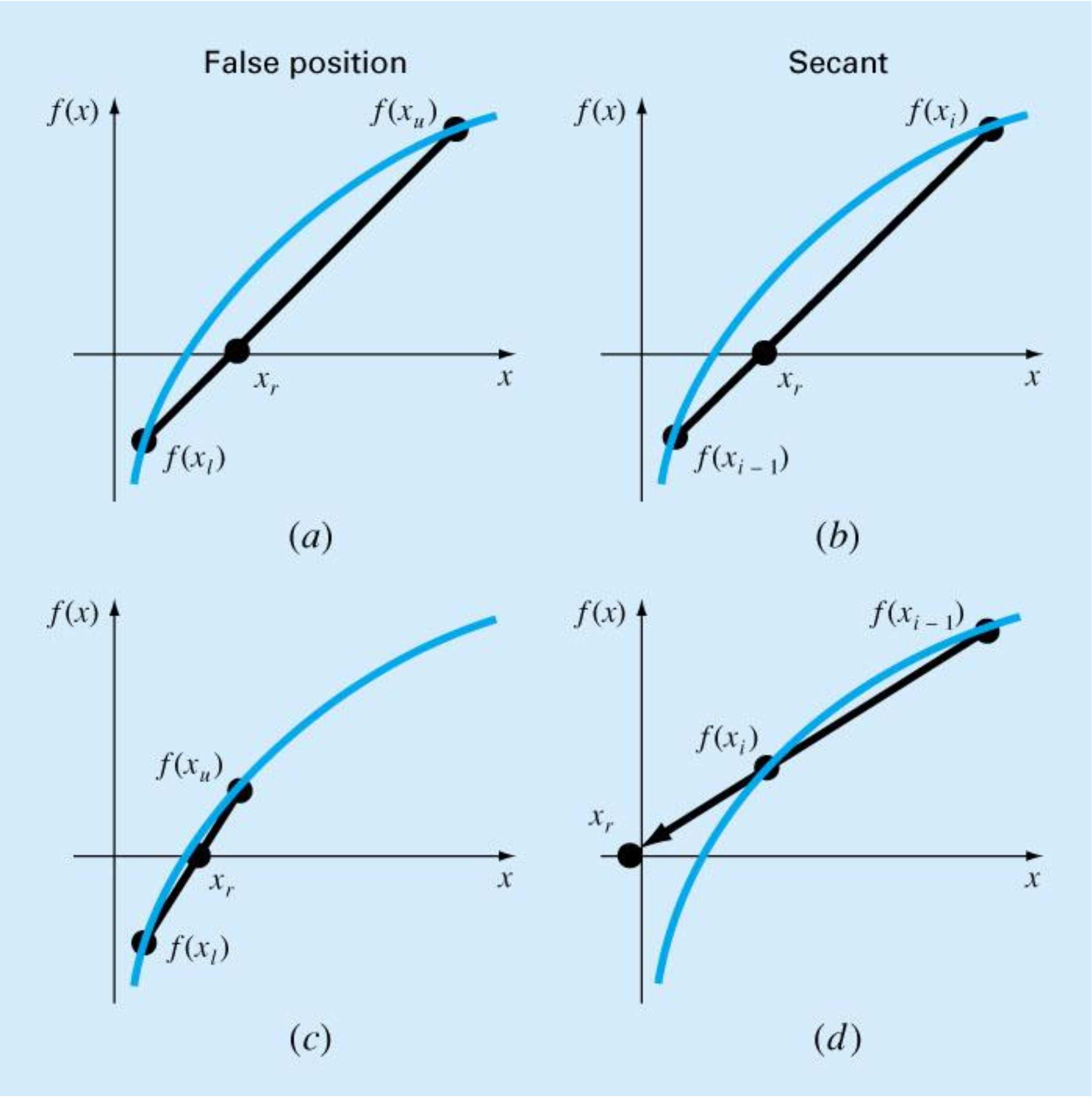
- Both methods use the same expression to compute x_r.

$$\text{Secant}: \qquad x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

$$\text{False position}: \quad x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

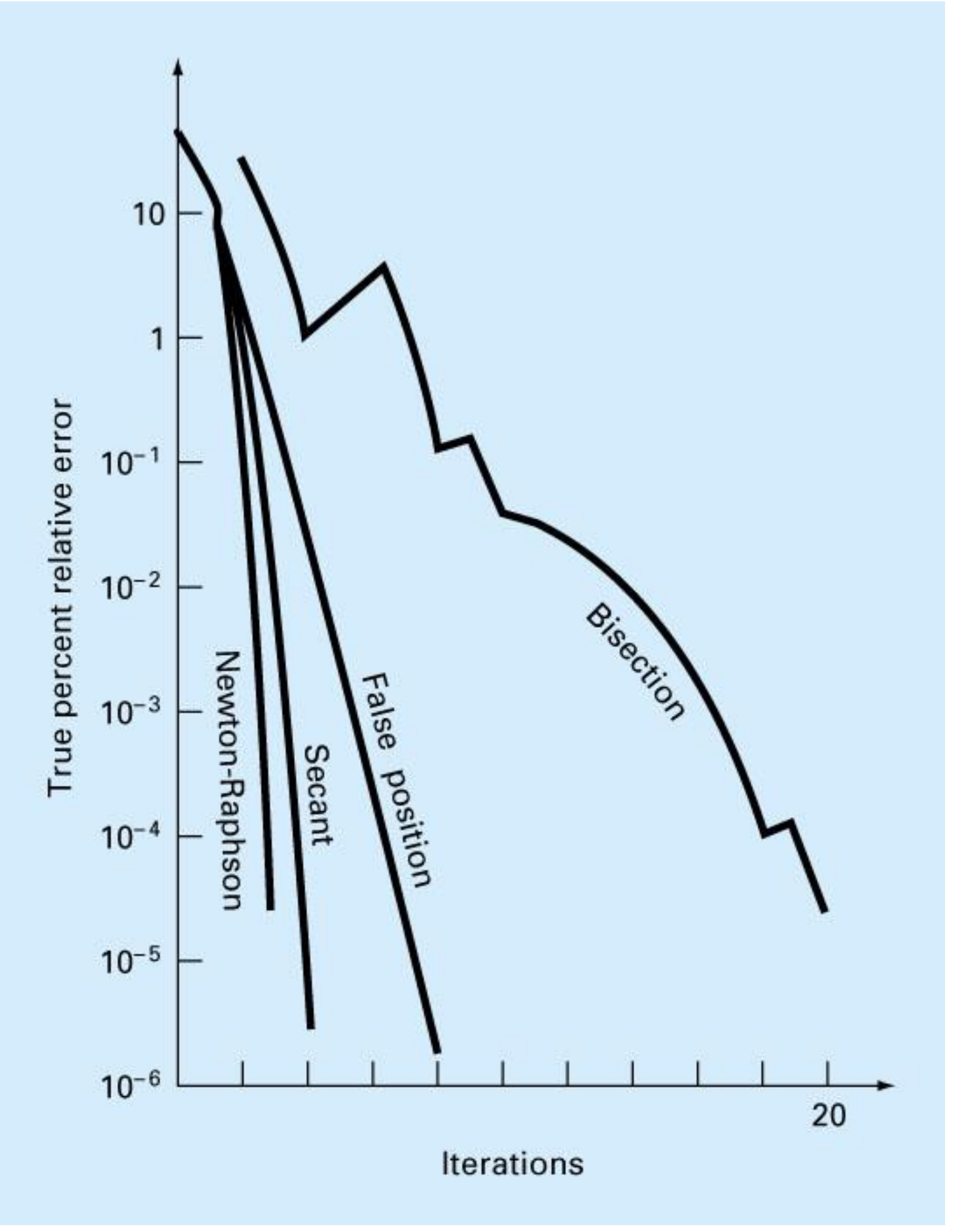
- They have different methods for the replacement of the initial values by the new estimate. (see next page)

Comparison of the Secant and False-position method



Comparison of the Secant and False-position method

$$f(x) = e^{-x} - x$$



Modified Secant Method

- Needs only one instead of two initial guess points
- Replace $x_{i-1} - x_i$ by δx_i and approximate $f'(x)$ as

$$f'(x_i) = \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$

- From Newton-Raphson method,

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$\Rightarrow \boxed{x_{i+1} = x_i - \frac{\delta x_i f(x_i)}{f(x_i + \delta x_i) - f(x_i)}}$$

Modified Secant Method

Find root of $f(x) = e^{-x} - x = 0$ with initial estimate of $x_0 = 1.0$ and δ=0.01. (Answer: α= 0.56714329)

i	x_i	$x_i+\delta x_i$	$f(x_i)$	$f(x_i+\delta x_i)$	x_{i+1}
0	1	1.01	-0.63212	-0.64578	0.537263
1	0.537263	0.542635	0.047083	0.038579	0.56701
2	0.56701	0.567143	0.000209	-0.00867	0.567143

Compared with the Secant method

i	x_{i-1}	x_i	$f(x_{i-1})$	$f(x_i)$	x_{i+1}	ε_t
0	0	1	1.00000	-0.63212	0.61270	8.0 %
1	1	0.61270	-0.63212	-0.07081	0.56384	0.58 %
2	0.61270	0.56384	-0.07081	0.00518	0.56717	0.0048 %

Modified Secant Method – About δ

If δ is too small, the method can be swamped by round-off error caused by subtractive cancellation in the denominator of

$$x_{i+1} = x_i - \frac{\delta x_i f(x_i)}{f(x_i + \delta x_i) - f(x_i)}$$

If δ is too big, this technique can become inefficient and even divergent.

If δ is selected properly, this method provides a good alternative for cases when developing two initial guess is inconvenient.

The following root finding methods will be introduced:

A. Bracketing Methods

- A.1. Bisection Method
- A.2. Regula Falsi

B. Open Methods

- B.1. Fixed Point Iteration
- B.2. Newton Raphson's Method
- B.3. Secant Method

Can they handle multiple roots?

Facts

- Any n^{th} order polynomial has exactly n zeros (counting real and complex zeros with their multiplicities).
- Any polynomial with an odd order has at least one real zero.
- If a function has a zero at $\boldsymbol{x=r}$ with multiplicity $\boldsymbol{m}$ then the function and its first $\boldsymbol{(m-1)}$ derivatives are zero at $\boldsymbol{x=r}$ *and the* $\boldsymbol{m}^{th}$ *derivative at* $\boldsymbol{r}$ *is not zero*.

Multiple Roots

- A **multiple root** corresponds to a point where a function is tangent to the x axis.

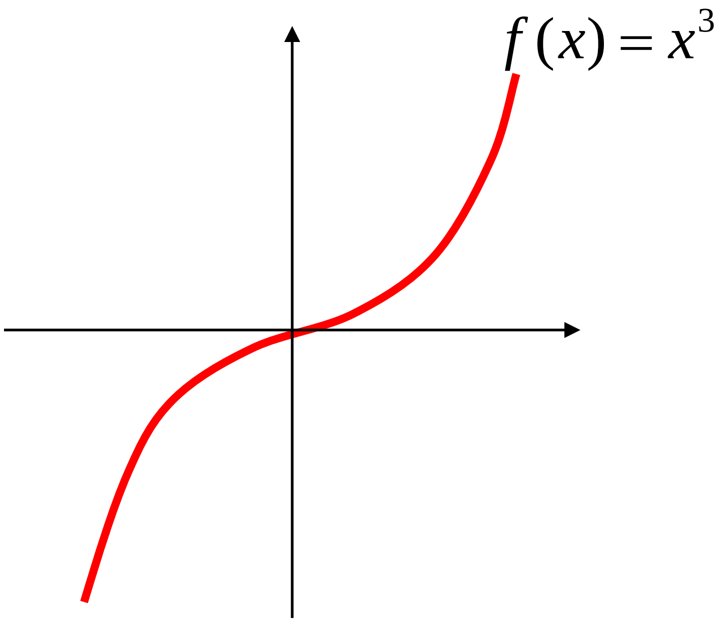
- For example, this function has a double root.

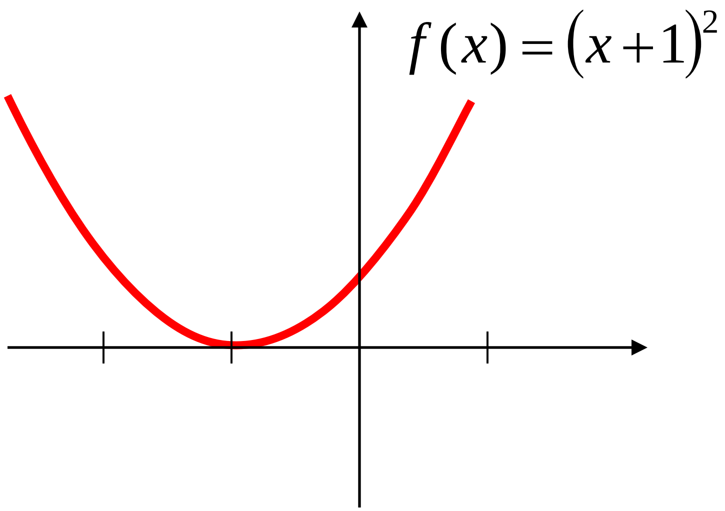
$$f(x) = (x-3)(x-1)(x-1)$$
$$= x^3 - 5x^2 + 7x - 3$$

- For example, this function has a triple root.

$$f(x) = (x-3)(x-1)(x-1)\ (x-1)$$
$$= x^4 - 6x^3 + 12x^2 - 10x + 3$$

Multiple Roots





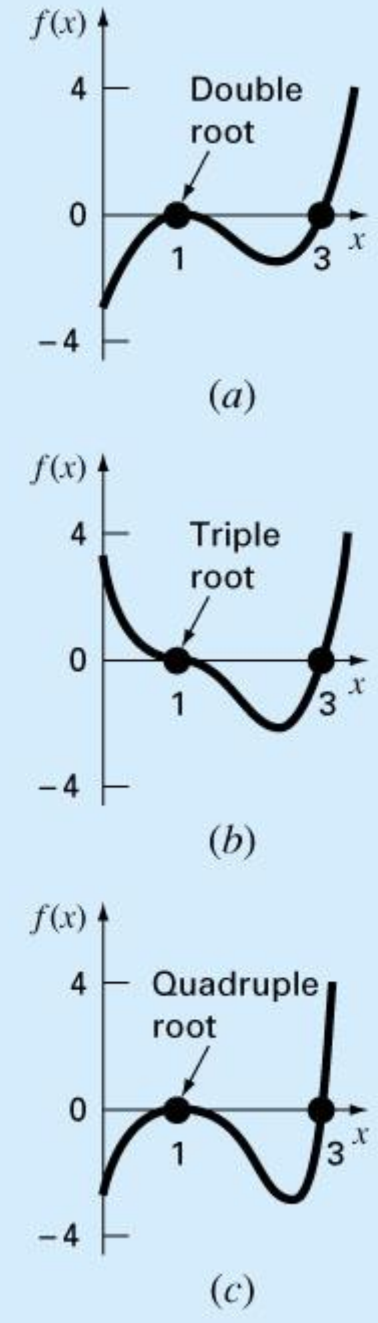
$f(x)$ has three zeros at x = 0

$f(x)$ has two zeros at x = -1

Multiple Roots

- Odd multiple roots cross the axis. (Figure (b))

- Even multiple roots do not cross the axis. (Figure (a) and (c))



Difficulties when we have multiple roots

- Bracketing methods do not work for even multiple roots.

- $f(\alpha) = f'(\alpha) = 0$, so both $f(x_i)$ and $f'(x_i)$ approach zero near the root. This could result in division by zero. A zero check for $f(x)$ should be incorporated so that the computation stops before $f'(x)$ reaches zero.

- For multiple roots, Newton-Raphson and Secant methods converge linearly, rather than quadratic convergence.

Modified Newton-Raphson Methods for Multiple Roots

- **Suggested Solution 1**:

Define $\tilde{f} = f^{1/m}$, m is the multiplicity of the root

$\tilde{f}(\alpha) = 0$ and α is a single root.

$$
\begin{aligned}
x_{i+1} &= x_i - \frac{\tilde{f}(x_i)}{\tilde{f}'(x_i)} \\
&= x_i - \frac{f^{1/m}(x_i)}{\frac{1}{m} f^{\frac{1}{m}-1}(x_i) f'(x_i)} \\
&= x_i - m \frac{f(x_i)}{f'(x_i)}
\end{aligned}
$$

Disadvantage:

work only when ***m*** is known.

Modified Newton's Method

Can be used for both single and multiple roots

(m = 1: original Newton's method)

m =1: single root

m=2, double root

m=3 triple root

etc.

```
function [x, f] = multiple1(func, dfunc)

% Find multiple root near xguess using the modified Newton's method
% Multiplicity m of the root is given -- m = 1: single root
% m = 2: double root; m = 3: triple root, etc.
% Input:
%       func      string containing name of function
%       dfunc     name of derivative of function
%       xguess    starting estimate
%       es        allowable tolerance in computed root
%       maxit     maximum number of iterations
% Output:
%       x         row vector of approximations to root

m = input('enter multiplicity of the root = ');
xguess = input('enter initial guess: xguess = ');
es = input('allowable tolerance: es = ');
maxit = input('maximum number of iterations: maxit = ');

iter = 1;
x(1) = xguess;
f(1) = feval(func, x(1));
dfdx(1) = feval(dfunc, x(1));
for i = 2 : maxit
     x(i) = x(i-1) - m * f(i-1) / dfdx(i-1);
     f(i) = feval(func, x(i));
     dfdx(i) = feval(dfunc, x(i));
     if abs(x(i) - x(i-1)) < es
        disp('Newton method has converged'); break;
     end
     iter = i;
end
if (iter >= maxit)
        disp('zero not found to desired tolerance');
end
n = length(x);   k = 1:n;
disp('   step                x                    f                    df/dx')
out = [k; x; f; dfdx];
fprintf('%5.0f  %20.14f  %21.15f  %21.15f\n',out)
```

Original Newton's method

m = 1

```
» multiple1('multi_func','multi_dfunc');
enter multiplicity of the root = 1
enter initial guess x1 = 1.3
allowable tolerance tol = 1.e-6
maximum number of iterations max = 100
Newton method has converged
```

step	x	y
1	1.30000000000000	-0.44217000000004
2	1.09600000000000	-0.06361262220902
3	1.04407272727272	-0.01453442847741
4	1.02126549372889	-0.00350359197248
5	1.01045853297516	-0.00086139138942
6	1.00518770530932	-0.00021362727675
7	1.00258369467652	-0.00005319712394
8	1.00128933592285	-0.00001327339304
9	1.00064404356011	-0.00000331513217
10	1.00032186610620	-0.00000082838226
11	1.00016089418619	-0.00000020704553
12	1.00008043738571	-0.00000005175515
13	1.00004021625682	-0.00000001293800
14	1.00002010751461	-0.00000000323440
15	1.00001005358967	-0.00000000080860
16	1.00000502663502	-0.00000000020213
17	1.00000251330500	-0.00000000005052
18	1.00000125681753	-0.00000000001262
19	1.00000062892307	-0.00000000000316

Modified Newton's Method

m = 2

```
» multiple1('multi_func','multi_dfunc');
enter multiplicity of the root = 2
enter initial guess x1 = 1.3
allowable tolerance tol = 1.e-6
maximum number of iterations max = 100
Newton method has converged
```

step	x	y
1	1.30000000000000	-0.44217000000004
2	0.89199999999999	-0.10925953065677
3	0.99229251101321	-0.00048075868939
4	0.99995587111371	-0.00000001557990
5	0.99999999853944	-0.00000000000000
6	1.00000060664549	-0.00000000000293

Double root : m = 2

$$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27 = 0$$

Modified Newton-Raphson Methods for Multiple Roots

- **Suggested Solution 2**:

Define $\tilde{f}(x) = \dfrac{f(x)}{f'(x)}$ (1)

$\tilde{f}(x)$ has roots at all the same locations as $f(x)$.

$$x_{i+1} = x_i - \frac{\tilde{f}(x_i)}{\tilde{f}'(x_i)} \qquad (2)$$

Differentiate (1) $\Rightarrow \tilde{f}'(x) = \dfrac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2}$ (3)

Sub (1) and (3) into (2) $\Rightarrow x_{i+1} = x_i - \dfrac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - f(x_i)f''(x_i)}$

Example of the Modified Newton-Raphson Method for Multiple Roots

- Original Newton Raphson method

$$f(x) = (x-3)(x-1)(x-1)$$
$$= x^3 - 5x^2 + 7x - 3$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$= x_i - \frac{x_i^3 - 5x_i^2 + 7x_i - 3}{3x_i^2 - 10x_i + 7}$$

i	x_i	ε_t (%)
0	0	100
1	0.4285714	57
2	0.6857143	31
3	0.8328654	17
4	0.9133290	8.7
5	0.9557833	4.4
6	0.9776551	2.2

The method is linearly convergent toward the true value of 1.0.

Example of the Modified Newton-Raphson Method for Multiple Roots

- For the modified algorithm

i	x_i	ε_t (%)
0	0	100
1	1.105263	11
2	1.003082	0.31
3	1.000002	0.00024

$$f(x) = (x-3)(x-1)(x-1)$$

$$= x^3 - 5x^2 + 7x - 3$$

$$x_{i+1} = x_i - \frac{f(x_i)\,f'(x_i)}{[f'(x_i)]^2 - f(x_i)\,f''(x_i)}$$

$$= x_i - \frac{(x_i^{\,3} - 5x_i^{\,2} + 7x_i - 3)(3x_i^{\,2} - 10x_i + 7)}{(3x_i^{\,2} - 10x_i + 7) - (x_i^{\,3} - 5x_i^{\,2} + 7x_i - 3)(6x_i - 10)}$$

Example of the Modified Newton-Raphson Method for Multiple Roots

- How about their performance on finding the single root?

i	Standard	ε_t (%)	Modified	ε_t (%)
0	4	33	4	33
1	3.4	13	2.636364	12
2	3.1	3.3	2.820225	6.0
3	3.008696	0.29	2.961728	1.3
4	3.000075	0.0025	2.998479	0.05
5	3.000000	2×10^{-7}	2.999998	7.7×10^{-5}

```matlab
function [x, f] = multiple2(func, dfunc, ddfunc)

% Find multiple root near xguess using the modified Newton's method.
% Multiplicity of the root is not known a priori
% Input:
%       func      string containing name of function
%       dfunc     name of derivative of function
%       ddfunc    name of second derivative of function
%       xguess    starting estimate
%       es        allowable tolerance in computed root
%       maxit     maximum number of iterations
% Output:
%       x         row vector of approximations to root

xguess = input('enter initial guess: xguess = ');
es = input('allowable tolerance: es = ');
maxit = input('maximum number of iterations: maxit = ');

iter = 1;
x(1) = xguess;
f(1) = feval(func, x(1));
dfdx(1) = feval(dfunc, x(1));
d2fdx2(1) = feval(ddfunc, x(1));
for i = 2 : maxit
     x(i) = x(i-1)-f(i-1)*dfdx(i-1)/(dfdx(i-1)^2-f(i-1)*d2fdx2(i-1));
     f(i) = feval(func, x(i));
     dfdx(i) = feval(dfunc, x(i));
     d2fdx2(i) = feval(ddfunc, x(i));
     if abs(x(i)-x(i-1)) < es
        disp('Newton method has converged'); break;
     end
     iter = i;
end
if (iter >= maxit)
        disp('zero not found to desired tolerance');
end
n = length(x);   k = 1 : n;
disp('   step        x              f            df/dx            d2f/dx2')
out=[k; x; f; dfdx; d2fdx2];
fprintf('%5.0f  %17.14f  %20.15f %20.15f %20.15f\n',out)
```

Modified Newton's method with $u = f / f'$

```
function f = multi_func(x)
% Exact solutions: x = 1 (double) and 3 (triple)
f = x.^5 - 11*x.^4 + 46*x.^3 - 90*x.^2 + 81*x - 27;
```

```
function f_pr = multi_pr(x)
% First derivative f'(x)
f_pr = 5*x.^4 - 44*x.^3 + 138*x.^2 - 180*x + 81;
```

```
function f_pp = multi_pp(x)
% Second-derivative f''(x)
f_pp = 20*x.^3 - 132*x.^2 + 276*x - 180;
```

Double root at x = 1

```
>> [x, f] = multiple2('multi_func','multi_dfunc','multi_ddfunc');
enter initial guess: xguess = 0
allowable tolerance: es = 1.e-6
maximum number of iterations: maxit = 100
Newton method has converged
```

step	x	f	df/dx	d2f/dx2
1	0.00000000000000	-27.00000000000000	81.00000000000000	-180.00000000000000
2	1.28571428571429	-0.41125721425594	-2.15910037484383	-0.83965014577259
3	1.08000000000002	-0.04529848320001	-1.06168320000175	-10.69055999999067
4	1.00519480519482	-0.00021421012955	-0.08214874792781	-15.62791421430577
5	1.00002034484531	-0.00000000331120	-0.00032550262434	-15.99853520093893
6	1.00000000031772	0.00000000000000	-0.00000000508359	-15.99999997712384
7	1.00000000031772	0.00000000000000	-0.00000000508359	-15.99999997712384

Original Newton's method (m = 1)

$$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27 = 0$$

```
>> [x,f] = multiple1('multi_func','multi_dfunc');
enter multiplicity of the root = 1
enter initial guess: xguess = 10
allowable tolerance: es = 1.e-6
maximum number of iterations: maxit = 200
Newton method has converged
```

Triple Root at x = 3

step	x	f	df/dx
1	10.00000000000000	27783.00000000000000	18081.00000000000000
2	8.46341463414634	9083.80126898861090	7422.20141618487380
3	7.23954576295397	2966.63373682804470	3050.17156837070520
4	6.26693367529599	967.24535263768371	1255.50368906350470
5	5.49652944545325	314.60452268468470	517.98239760637011
6	4.88916416791005	101.98155988768616	214.39105831808899
7	4.41348406871311	32.90550152144181	89.11885079830165
8	4.04425240530314	10.55304447740986	37.25060494810270
9	3.76095379868689	3.35886962312816	15.67519975524624
10	3.54667457573766	1.05957946995756	6.64680914767666
…	……..	……	…..
130	2.99988506446967	-0.00000000000617	0.00000015849787
131	2.99992397673381	-0.00000000000176	0.00000006934738
132	2.99994938715307	-0.00000000000043	0.00000003073785
133	2.99996325688118	-0.00000000000009	0.00000001619992
134	2.99996852018682	0.00000000000000	0.00000001189108
135	2.99996852018682	0.00000000000000	0.00000001189108

Modified Newton's method

$$f(x) = x^5 - 11x^4 + 46x^3 - 90x^2 + 81x - 27 = 0$$

```
>> [x,f] = multiple2('multi_func','multi_dfunc','multi_ddfunc');
enter initial guess: xguess = 10
allowable tolerance: es = 1.e-6
maximum number of iterations: maxit = 100
Newton method has converged
 step         x                    f                      df/dx                   d2f/dx2
   1  10.00000000000000  27783.00000000000000  18081.00000000000000  9380.00000000000000
   2   2.42521994134897      -0.38571706869916 5      1.47193319869160 2     -1.73468593031321 9
   3   2.80435435817775      -0.02438115076461 1      0.34683200123009 8     -3.00796439424448 2
   4   2.98444590681717      -0.00001481878575 8      0.00284324244478 3     -0.36176086525802 0
   5   2.99991809093254      -0.00000000000218 8      0.00000008050028 6     -0.00196549559348 1
   6   2.99999894615774      -0.00000000000002 8      0.00000000001352 9     -0.00002529216101 3
   7   2.99999841112323       0.00000000000000 0      0.00000000003058 2     -0.00003813292130 4
```

Triple root at x = 3

- **Original Newton-Raphson method required 135 iterations**
- **Modified Newton's method converged in only 7 iterations**

Modified Newton-Raphson Methods for Multiple Roots

- What's the disadvantage of the modified Newton-Raphson Methods for multiple roots over the original Newton-Raphson method?

- Note that the Secant method can also be modified in a similar fashion for multiple roots.

Summary of Open Methods

- Unlike bracketing methods, open methods do not always converge.
- Open methods, if converge, usually converge more quickly than bracketing methods.
- Open methods can locate even multiple roots whereas bracketing methods cannot. (why?)

Study Objectives

- Understand the graphical interpretation of a root
- Understand the differences between bracketing methods and open methods for root location
- Understand the concept of convergence and divergence
- Know why bracketing methods always converge, whereas open methods may sometimes diverge
- Realize that convergence of open methods is more likely if the initial guess is close to the true root.

Study Objectives

- Understand what conditions make a method converges quickly or diverges
- Understand the concepts of linear and quadratic convergence and their implications for the efficiencies of the fixed-point-iteration and Newton-Raphson methods
- Know the fundamental difference between the false-position and secant methods and how it relates to convergence
- Understand the problems posed by multiple roots and the modifications available to mitigate them

Summary

Method	Pros	Cons
Bisection	- Easy, Reliable, Convergent - One function evaluation per iteration - No knowledge of derivative is needed	- Slow - Needs an interval [a,b] containing the root, i.e., $f(a)f(b)<0$
Newton	- Fast (if near the root) - Two function evaluations per iteration	- May diverge - Needs derivative and an initial guess x_0 such that $f'(x_0)$ is nonzero
Secant	- Fast (slower than Newton) - One function evaluation per iteration - No knowledge of derivative is needed	- May diverge - Needs two initial points guess x_0, x_1 such that $f(x_0)- f(x_1)$ is nonzero

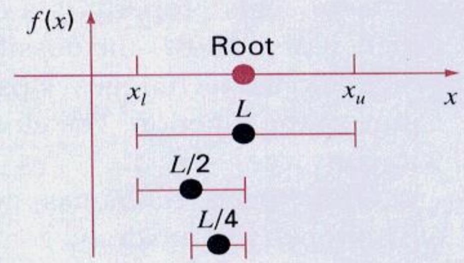
Summary: Root Finding

Bracketing methods:

Bisection

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

If $f(x_l)f(x_r) < 0,\; x_u = x_r$
$\quad f(x_l)f(x_r) > 0,\; x_l = x_r$



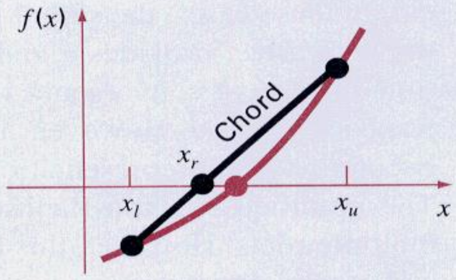
Stopping criterion:

$$\left|\frac{x_r^{\text{new}} - x_r^{\text{old}}}{x_r^{\text{new}}}\right| 100\% \le \epsilon_s$$

False position

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

If $f(x_l)f(x_r) < 0,\; x_u = x_r$
$\quad f(x_l)f(x_r) > 0,\; x_l = x_r$

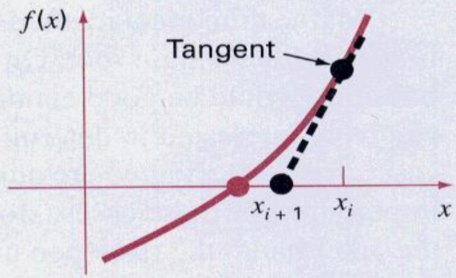


Stopping criterion:

$$\left|\frac{x_r^{\text{new}} - x_r^{\text{old}}}{x_r^{\text{new}}}\right| 100\% \le \epsilon_s$$

Newton-Raphson

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$



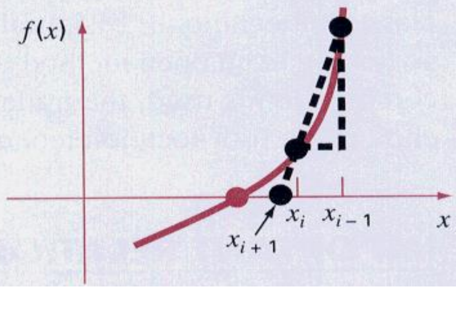
Stopping criterion:

$$\left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| 100\% \le \epsilon_s$$

Error: $E_{i+1} = O(E_i^2)$

Secant

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f'(x_{i-1}) - f(x_i)}$$



Stopping criterion:

$$\left|\frac{x_{i+1} - x_i}{x_{i+1}}\right| 100\% \le \epsilon_s$$

Systems of Nonlinear Equations

- Locate the roots of a set of simultaneous nonlinear equations:

$$f_1(x_1, x_2, x_3, \ldots, x_n) = 0$$

$$f_2(x_1, x_2, x_3, \ldots, x_n) = 0$$

$$\vdots$$

$$f_n(x_1, x_2, x_3, \ldots, x_n) = 0$$

Example:

$$x_1^{\,2} + x_1 x_2 = 10 \quad \Rightarrow \quad \mathrm{f}_1(x_1, x_2) = x_1^{\,2} + x_1 x_2 - 10 = 0$$

$$x_2 + 3x_1 x_2^{\,2} = 57 \quad \Rightarrow \quad \mathrm{f}_2(x_1, x_2) = x_2 + 3x_1 x_2^{\,2} - 57 = 0$$

- **First-order** Taylor series expansion of a function with more than one variable:

$$f_{1(i+1)} = f_{1(i)} + \frac{\partial f_{1(i)}}{\partial x_1}(x_{1(i+1)} - x_{1(i)}) + \frac{\partial f_{1(i)}}{\partial x_2}(x_{2(i+1)} - x_{2(i)}) = 0$$

$$f_{2(i+1)} = f_{2(i)} + \frac{\partial f_{2(i)}}{\partial x_1}(x_{1(i+1)} - x_{1(i)}) + \frac{\partial f_{2(i)}}{\partial x_2}(x_{2(i+1)} - x_{2(i)}) = 0$$

- The root of the equation occurs at the value of x_1 and x_2 where $f_{1(i+1)}=0$ and $f_{2(i+1)}=0$ Rearrange to solve for $x_{1(i+1)}$ and $x_{2(i+1)}$

$$\frac{\partial f_{1(i)}}{\partial x_1}x_{1(i+1)} + \frac{\partial f_{1(i)}}{\partial x_2}x_{2(i+1)} = -f_{1(i)} + \frac{\partial f_{1(i)}}{\partial x_1}x_{1(i)} + \frac{\partial f_{1(i)}}{\partial x_2}x_{2(i)}$$

$$\frac{\partial f_{2(i)}}{\partial x_1}x_{1(i+1)} + \frac{\partial f_{2(i)}}{\partial x_2}x_{2(i+1)} = -f_{2(i)} + \frac{\partial f_{2(i)}}{\partial x_1}x_{1(i)} + \frac{\partial f_{2(i)}}{\partial x_2}x_{2(i)}$$

$$\begin{bmatrix} \dfrac{\partial f_{1(i)}}{\partial x_1} & \dfrac{\partial f_{1(i)}}{\partial x_2} \\ \dfrac{\partial f_{2(i)}}{\partial x_1} & \dfrac{\partial f_{2(i)}}{\partial x_2} \end{bmatrix} \begin{bmatrix} x_{1(i+1)} \\ x_{2(i+1)} \end{bmatrix} = -\begin{bmatrix} f_{1(i)} \\ f_{2(i)} \end{bmatrix} + \begin{bmatrix} \dfrac{\partial f_{1(i)}}{\partial x_1} & \dfrac{\partial f_{1(i)}}{\partial x_2} \\ \dfrac{\partial f_{2(i)}}{\partial x_1} & \dfrac{\partial f_{2(i)}}{\partial x_2} \end{bmatrix} \begin{bmatrix} x_{1(i)} \\ x_{2(i)} \end{bmatrix}$$

- Since $x_{1(i)}$, $x_{2(i)}$, $f_{1(i)}$, and $f_{2(i)}$ are all known at the i^{th} iteration, this represents a set of two linear equations with two unknowns, $x_{1(i+1)}$ and $x_{2(i+1)}$
- You may use several techniques to solve these equations

Newton's Method for Systems of Non Linear Equations

$$\textit{Given}: X_0 \text{ an initial guess of the root of } F(x) = 0$$

$$\textit{Newton's Iteration}$$

$$X_{i+1} = X_i - \left[F'(X_i)\right]^{-1} F(X_i)$$

$$F(X) = \begin{bmatrix} f_1(x_1, x_2, \ldots) \\ f_2(x_1, x_2, \ldots) \\ \vdots \end{bmatrix}, \quad J(X) = F'(X) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \vdots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \\ \vdots & & \end{bmatrix}$$

The Jacobian Matrix

- Hence, we write

$$
\mathbf{J}(\mathbf{x}^{(i)}) = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1}\left(\mathbf{x}^{(i)}\right) & \frac{\partial f_1}{\partial x_2}\left(\mathbf{x}^{(i)}\right) & \cdots & \frac{\partial f_1}{\partial x_n}\left(\mathbf{x}^{(i)}\right) \\
\frac{\partial f_2}{\partial x_1}\left(\mathbf{x}^{(i)}\right) & \frac{\partial f_2}{\partial x_2}\left(\mathbf{x}^{(i)}\right) & \cdots & \frac{\partial f_2}{\partial x_n}\left(\mathbf{x}^{(i)}\right) \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial f_n}{\partial x_1}\left(\mathbf{x}^{(i)}\right) & \frac{\partial f_n}{\partial x_2}\left(\mathbf{x}^{(i)}\right) & \cdots & \frac{\partial f_n}{\partial x_n}\left(\mathbf{x}^{(i)}\right)
\end{bmatrix}
$$

Example

- Solve the following system of equations:

$$y + x^2 - 0.5 - x = 0$$

$$x^2 - 5xy - y = 0$$

$$\text{Initial guess} \quad x = 1, \quad y = 0$$

$$F = \begin{bmatrix} y + x^2 - 0.5 - x \\ x^2 - 5xy - y \end{bmatrix}, \quad J = F' = \begin{bmatrix} 2x - 1 & 1 \\ 2x - 5y & -5x - 1 \end{bmatrix}, X_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Solution Using Newton's Method

Iteration 1:

$$F = \begin{bmatrix} y + x^2 - 0.5 - x \\ x^2 - 5xy - y \end{bmatrix} = \begin{bmatrix} -0.5 \\ 1 \end{bmatrix}, \quad J = F' = \begin{bmatrix} 2x-1 & 1 \\ 2x-5y & -5x-1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & -6 \end{bmatrix}$$

$$X_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 2 & -6 \end{bmatrix}^{-1} \begin{bmatrix} -0.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.25 \\ 0.25 \end{bmatrix}$$

Iteration 2:

$$F = \begin{bmatrix} 0.0625 \\ -0.25 \end{bmatrix}, \quad J = F' = \begin{bmatrix} 1.5 & 1 \\ 1.25 & -7.25 \end{bmatrix}$$

$$X_2 = \begin{bmatrix} 1.25 \\ 0.25 \end{bmatrix} - \begin{bmatrix} 1.5 & 1 \\ 1.25 & -7.25 \end{bmatrix}^{-1} \begin{bmatrix} 0.0625 \\ -0.25 \end{bmatrix} = \begin{bmatrix} 1.2332 \\ 0.2126 \end{bmatrix}$$

Example

Try this

- Solve the following system of equations:

$$y + x^2 - 1 - x = 0$$

$$x^2 - 2y^2 - y = 0$$

$$\text{Initial guess} \quad x = 0, \quad y = 0$$

$$F = \begin{bmatrix} y + x^2 - 1 - x \\ x^2 - 2y^2 - y \end{bmatrix}, \quad J = F' = \begin{bmatrix} 2x - 1 & 1 \\ 2x & -4y - 1 \end{bmatrix}, \quad X_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Example

Solution

Iteration	0	1	2	3	4	5
X_k	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -0.6 \\ 0.2 \end{bmatrix}$	$\begin{bmatrix} -0.5287 \\ 0.1969 \end{bmatrix}$	$\begin{bmatrix} -0.5257 \\ 0.1980 \end{bmatrix}$	$\begin{bmatrix} -0.5257 \\ 0.1980 \end{bmatrix}$