# 방정식 근 구하기

김기택
국민대학교 소프트웨어학과

# 개요

- 공학적인 문제에서 발생하는 일반적인 문제는 함수 $f(x)$ 가 주어질 때 $f(x) = 0$ 인 $x$ 값을 구하는 것이다.
  - 해($x$ 의 값)는 방정식 $f(x) = 0$ 의 근(roots) 또는 함수 $f(x)$ 의 제로(zeros) 라고 한다.
- 함수의 개념
  - 함수 $f(x)$ 는 입력 값, $x$ 출력 값, $y$ 그리고 출력의 계산을 위한 규칙, $f$ 3가지 요소를 포함한다.
  - 방정식의 근은 실수 또는 복소수 일 수 있다. 물리적으로는 실수 근이 중요성을 가진다. 단, 다항식에서의 복소수 근은 의미를 가질 수 있어 예외이다.
- 본 장에서는 실수를 찾는데 집중할 것이다. 이후 복소수 근에 대해서도 다룬다.
  - 일반적으로 방정식에는 임의의 수의 실수 근이 있거나, 근이 전혀 없을 수 있다.
- 근을 찾는 모든 방법은 **출발점에서 절차를 반복**하는 방법이다. 따라서 **초기 추정값은 매우 중요**하다.
  - 잘못된 출발점으로 인해 근을 찾지 못하거나 원하는 근을 찾지 못하는 수가 있다.
  - 시각적으로 그래프를 그려 찾는 방법도 있으나 항상 가능한 방법은 아니다.
  - **구간법(bracketing method)와 개방법(open method)** 으로 구분할 수 있다.

# 구간법 (Bracketing method)

# 구간법 개요

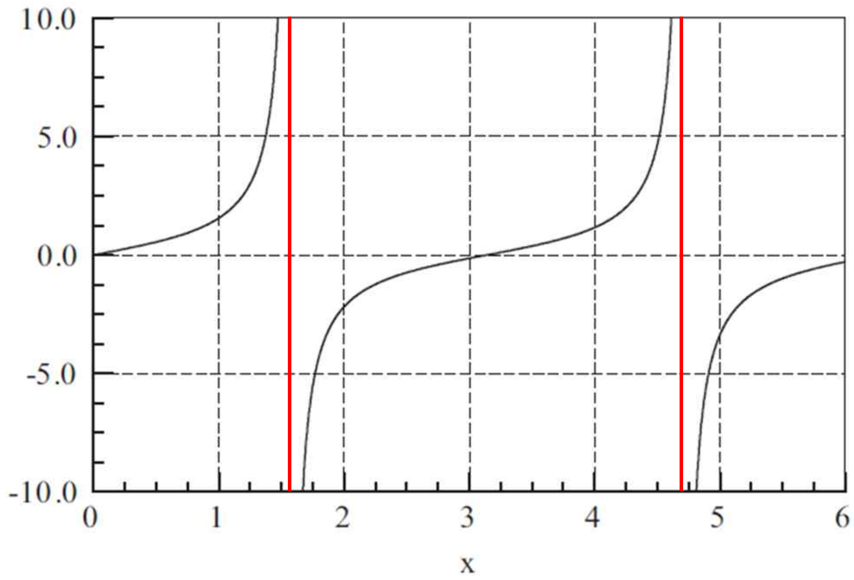
- 구간법은 근이 있을 것으로 추정되는 구간을 미리 정한 후 구간 내에서 근을 탐색하는 방법
    - 함수의 그래프를 그리는 방법이 대략적인 근의 위치를 파악하는 가장 수월한 방법이다.

- 주요 방법
    - 증분 탐색법
    - 이분법
    - Ridder 법

# 증분 탐색법 (incremental search method)

- 대략적인 그래프를 그려 근의 위치를 대략적으로 파악한 후 근 주변을 구간을 정하고 작은 세부 구간을 정해서 차례로 탐색하는 방법이 증분 탐색법이다.
  - 탐색 기준: 구간 **양끝점에서의 함수 값이 서로 반대 부호**를 가지고 있으면 해당 구간에 **적어도 하나의 근이 존재**한다. (구간이 충분히 작으면 단일 근을 포함할 수 있다.)

- 증분 탐색법의 문제점
  - 검색 증분 $\Delta x$ 가 근의 간격보다 크면 두개의 근접한 근을 놓칠 수 있다.
  - 이중근(동일한 2개의 근)은 감지되지 않는다.
  - $f(x)$ 의 **특정 특이점(극점)이 근으로 오인**될 수 있다. 예를 들어 그림에 나타난 것과 같이 $f(x) = \tan x$ 는 $x = \pm 0.5n\pi, n = 1,3,5, \ldots$ 에서 부호를 변경한다. 그러나 이 위치는 함수가 $x$ 축을 가로 지르지 않기 때문에 근이 아니다.



# rootsearch 프로그램

이 함수는 간격 (a, b) 에서 증분 dx 단위로 사용자 제공 함수 $f(x)$ 의 근을 검색한다. 검색에 성공한 경우 근의 경계 (x1, x2) 를 반환한다. x1 = x2 = None 은 근이 감지되지 않았음을 나타낸다. 첫번째 근(a에 가장 가까운 근)이 감지되면 다음 루트를 찾기 위해 x2 로 대체된 rootsearch 를 다시 호출할 수 있다. 근을 감지하는 한 이를 반복할 수 있다.

```
## module rootsearch
''' x1,x2 = rootsearch(f,a,b,dx).
   Searches the interval (a,b) in increments dx for
   the bounds (x1,x2) of the smallest root of f(x).
   Returns x1 = x2 = None if no roots were detected.
'''
from numpy import sign

def rootsearch(f,a,b,dx):
   x1 = a; f1 = f(a)      구간을 설정하고 함수 값을 계산
   x2 = a + dx; f2 = f(x2)
   while sign(f1) == sign(f2):   구간 내에 근이 없으면 구간 조정
      if x1 >= b: return None,None

      x1 = x2; f1 = f2                 증분을 더해 다음 구간 탐색
      x2 = x1 + dx; f2 = f(x2)
   else:
      return x1,x2   구간 내에 근이 있으면 구간 값 반환
```

구간 오른쪽 끝에 다다랐는데도 근
이 없으면 근 없음을 반환하고 종료

# 예제 4.1

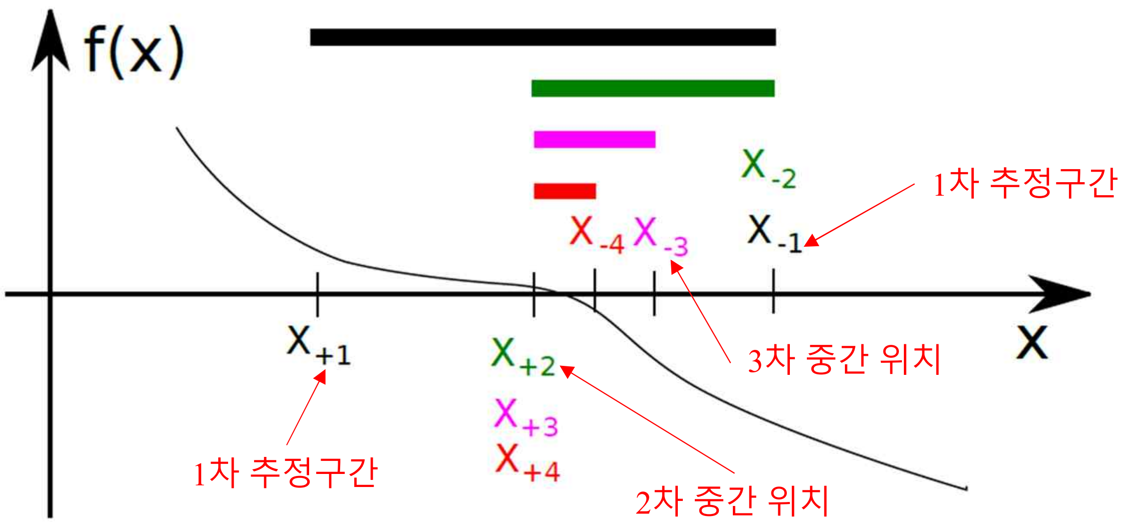
함수 $f(x) = x^3 - 10x^2 + 5 = 0$ 의 근은 구간 (0, 1) 에 있다. rootsearch 를 사용하여 소수점 아래 4자리 정도의 정확도로 근을 계산 하라.

[풀이] 네 자리 정도의 정확도를 얻으려면 $\Delta x = 0.0001$ 보다 크지 않은 검색 증분이 필요하다. 간격 (0, 1) 에서 증분 $\Delta x = 0.0001$ 을 이용하여 검색하면 10,000 번의 함수가 반복 평가된다. 이를 줄이기 위해 다음 프로그램은 4단계로 해를 검색하여 함수 평가 수를 40번으로 줄인다. 각 단계는 10개의 검색 간격을 가진다.

```
## example4_1
from rootsearch import *
def f(x): return x**3 - 10.0*x**2 + 5.0

x1 = 0.0; x2 = 1.0
for i in range(4):
    dx = (x2 - x1)/10.0    단계별로 구간을 10개로 나누어 증분을 구간의 1/10 로 한다. 이렇게 하여 소수점 아래 자리를 하나씩 증가한다.
    x1,x2 = rootsearch(f,x1,x2,dx)    해당 구간에 근이 있는지 찾는다. 찾아지면 해당 구간을 다시 10등분하여 찾는다.
x = (x1 + x2)/2.0    찾아진 구간 내에서 근 위치를 중간 값으로 결정한다.
print('x =', '{:6.4f}'.format(x))
```

# 이분법 (bisection search method, interval halving method)

- 이분법은 탐색 구간을 **연속적으로 반으로 줄여 간격이 충분히 작아질 때까지 진행**한다.
  - 이 방법은 간격 반감 방법(interval halving method)이라고도 한다.
  - 이분법은 근을 계산하는 가장 빠른 방법은 아니지만 **가장 신뢰할 수 있는 방법**이다. 근이 구간 내에 있으면 이분법은 항상 근으로 접근한다.



# 이분법 (2)

- 증분 검색과 동일한 원리를 사용한다.
  - 구간 $(x_1, x_2)$ 에 **근이 있으면 $f(x_1)$ 와 $f(x_2)$ 는 반대 부호**가 된다.
  - 구간을 반으로 줄이기 위해 중간점 $x_3 = (x_1 + x_2)/2$ 을 기준으로 $(x_1, x_3)$ 와 $(x_3, x_2)$ 를 탐색하여 어느 구간에 근이 있는지 찾아 내고 **새로운 탐색 구간으로 변경**한다.
  - 구간이 작은 값 $\varepsilon$ 으로 줄어들 까지 반복한다. $|x_2 - x_1| \le \varepsilon$

- 규정된 $\varepsilon$ 에 도달하는데 필요한 이분법의 수 계산
  - 원래 구간 $\Delta x$ 는 이분한 후 $\Delta x/2$ 로 줄어들고 두 개의 분할 후 $\Delta x/2^2$ 로 줄어들며, $n$ 개의 이분 후에 $\Delta x/2^n$ 으로 줄어든다.
  - $\Delta x/2^n = \varepsilon$ 로 설정하고 $n$ 을 구하면 다음과 같다. ($n$ 은 정수이므로 올림값을 사용한다.)

$$n = \frac{\ln(\frac{\Delta x}{\varepsilon})}{\ln 2} \qquad (4.1)$$

# bisection 프로그램

이 함수는 구간 $(x_1, x_2)$에 이분법을 사용하여 $f(x) = 0$ 의 근을 구한다. 구간을 허용 오차 내로 줄이는데 필요한 이분법의 수, $n$ 은 식 (4.1)을 이용한다. switch = 1 로 설정하면 함수의 크기가 반씩 감소하는지 확인할 수 있다. 만약 무언가 잘못 되었을 수도 있는데, 그런 경우 root = None 이 반환된다. 그러나, 이 기능은 반드시 필요한 것은 아니므로 기본값을 0 으로 설정한다.

```
## module bisection
''' root = bisection(f,x1,x2,switch=0,tol=1.0e-9).
   Finds a root of f(x) = 0 by bisection.
   The root must be bracketed in (x1,x2).
   Setting switch = 1 returns root = None if
   f(x) increases upon bisection.
'''
import math
import error
from numpy import sign

def bisection(f,x1,x2,switch=0,tol=1.0e-9):
   f1 = f(x1)
   if f1 == 0.0: return x1    구간의 양끝점에 근이 있으면 반환
   f2 = f(x2)
   if f2 == 0.0: return x2
   if sign(f1) == sign(f2):    구간 내에 근이 없으면 반환
      error.err('Root is not bracketed')
   n = int(math.ceil(math.log(abs(x2–x1)/tol)/math.log(2.0)))
                                        식 (4.1)에 의해 이분 횟수 결정
   for i in range(n):
      x3 = 0.5*(x1 + x2); f3 = f(x3)    중간점 계산
      if (switch == 1) and (abs(f3) > abs(f1))        \
                       and (abs(f3) > abs(f2)):
         return None
      if f3 == 0.0: return x3    중간점에 근이 있으면 반환
      if sign(f2)!= sign(f3): x1 = x3; f1 = f3
      else: x2 = x3; f2 = f3    근이 있는 구간으로 탐색 구간 변경
   return (x1 + x2)/2.0
```

switch가 1 인경우, 중간점 함수값의 절대값이 양끝에 비해 모두 크면 None 반환 (특이점일 가능성)

# 예제 4.2

이분법을 사용하여 간격 $(0, 1)$ 에서 4자리 정확도까지 $x^3 - 10x^2 + 5 = 0$ 의 근을 찾아라. 몇 번의 함수 계산이 필요한가?

증분 탐색법 예제와 동일한 함수

[풀이] 다음 프로그램을 사용하자.

```
## example4_2
from bisection import *

def f(x): return x**3 - 10.0*x**2 + 5.0
x = bisection(f, 0.0, 1.0, tol = 1.0e-4)
print('x =', '{:6.4f}'.format(x))
```

출력은

```
x = 0.7346
```

식 (4.1)을 사용하면,

$$n = \frac{\ln(\frac{|\Delta x|}{\varepsilon})}{\ln 2} = \frac{\ln(\frac{1.0}{0.0001})}{\ln 2} = 13.29 \approx 14$$

증분 탐색법에 비해 월등히 빠르다.

# 예제 4.3

이분법으로 구간 (0, 20) 에서 $f(x) = x - \tan x$ 의 모든 근을 찾아라. rootsearch 와 bisection 함수를 사용하라.

[풀이] $\tan x$ 는 $x = \frac{\pi}{2}, \frac{3\pi}{2}, \ldots$ 에서 특이점을 가지며 부호가 바뀐다. 이분법에 의해 이런 점을 해라고 오판하지 않도록 switch = 1 로 설정한다. 해가 특이점에 가깝다는 것은 rootsearch 에 작은 증분($\Delta x$)을 사용하여 완화할 수 있는 또 다른 잠재적인 문제이다. $\Delta x = 0.01$ 을 선택하여 다음 프로그램에 이를 수 있다.

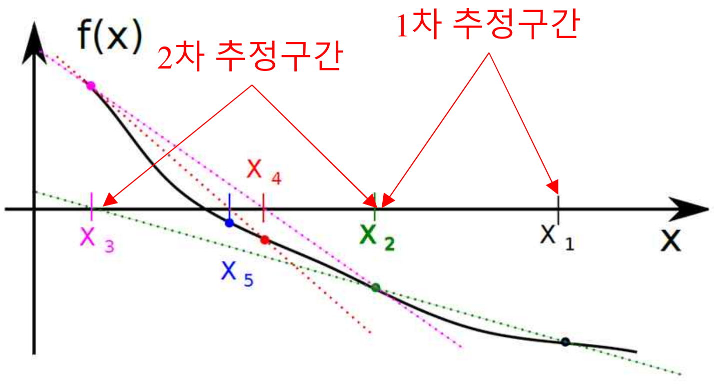
```
## example4_3
import math
from rootsearch import *
from bisection import *

def f(x): return x - math.tan(x)

a,b,dx = (0.0, 20.0, 0.01)
print("The roots are:")

while True:
    x1,x2 = rootsearch(f,a,b,dx)      탐색 구간을 크게 구분하여 우선 탐색
    if x1 != None:    근이 해당 구간에 있는 경우
        a = x2     다음 탐색 구간 조정
        root = bisection(f,x1,x2,1)
        if root != None: print(root)   특이점이 아닌 경우
    else:
        print("\nDone")  근이 없는 경우
        break
```

출력

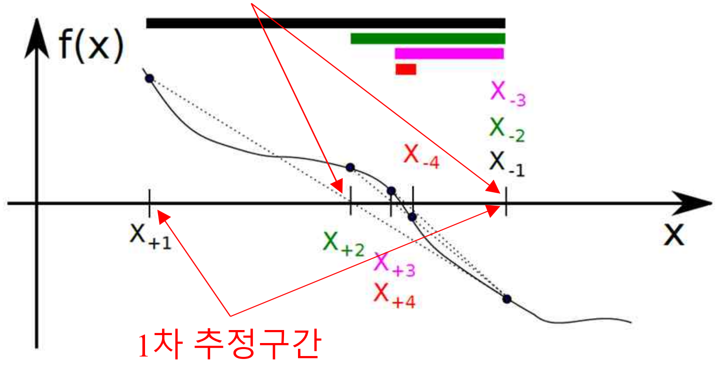
```
The roots are:
0.0
4.493409458100745
7.725251837074637
10.904121659695917
14.06619391292308
17.220755272209537
Done
```

# 선형 보간에 기반한 방법

- **시컨트법(secant method)**과 **가위치법(false position method)**은 새로운 근의 추정값을 두점 간의 선형 보간에 의해 추정한다.
  - 가위치법은 근이 있는 구간을 지정하기 위해 두점$(x_1, x_2)$이 필요하지만 구간 내에 근이 없어도 된다. 식 (4.2) 에 의해 구간이 개선되며 근이 있는 구간을 새롭게 지정한다.
  - 시컨트법은 구간 내에 근이 있는 두점 사이에서 선형보간 직선을 이용하여 근을 찾아 나간다.
- **두 방법은 모두 Ridder's method 에 비해 우수하지 않아 더 이상 사용하지 않는다.**
  - 그러나, 명칭과 기본적인 이해는 할 필요가 있다.



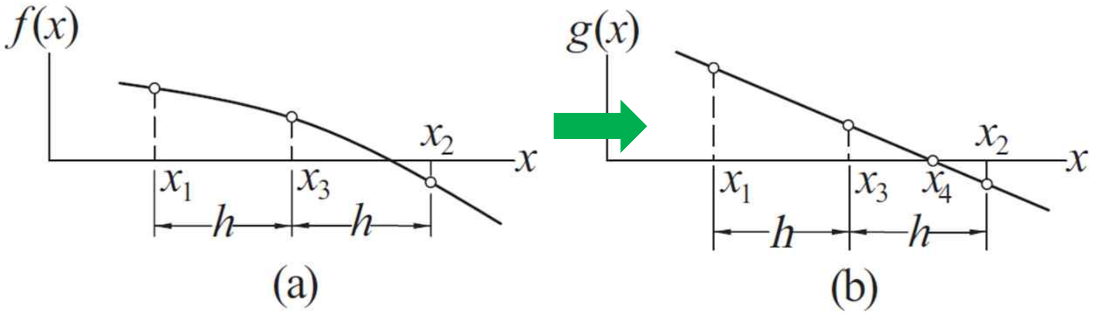
가위치법 (false position method)

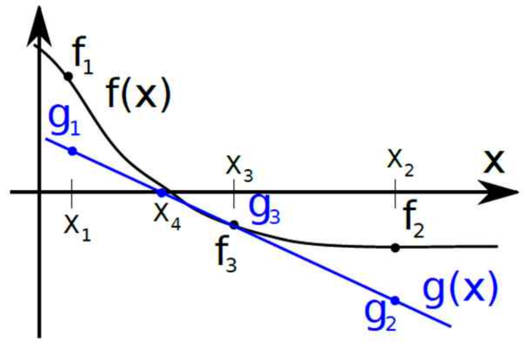


Secant 법

# Ridder 방법 (1)

- Ridder 방법은 가위치법에 비해 우수하다. 근이 $(x_1, x_2)$ 구간 내에 있다고 하면 다음 그림과 같이 $f_3 = f(x_3)$ 를 계산한다.




- 다음 함수를 예로 들어 살펴 봄으로써 상세하게 설명한다.

$$g(x) = f(x)e^{(x-x_1)Q} \qquad \text{(a)}$$

  - 여기서 상수 $Q$ 는 위 그림과 같이 점 $(x_1, g_1), (x_2, g_2), (x_3, g_3)$ 이 직선 위에 놓이도록 결정한다. 그리고 $g_i = g(x_i)$ 이다.
  - 근의 개선된 값은 $f(x)$ 가 아닌 $g(x)$ 의 선형보간에 의해 얻어진다. 세부적으로 살펴 보면, 식 (a) 에서

$$g_1 = f_1 \qquad g_2 = f_2 e^{2hQ} \qquad g_3 = f_3 e^{hQ} \qquad \text{(b)}$$

# Ridder 방법 (2)

- 여기서 $h = (x_2 - x_1)/2$ 이다. 그림 처럼 3 점이 직선 상에 있어야 하는 조건은 $g_3 = (g_1 + g_2)/2$ 또는

$$f_3 e^{hQ} = \frac{1}{2}\left(f_1 + f_2 e^{2hQ}\right)$$

- 이 것은 $e^{hQ}$ 의 2차 방정식이며 근은 다음과 같다.

$$e^{hQ} = \frac{f_3 \pm \sqrt{f_3^2 - f_1 f_2}}{f_2} \qquad \text{(c)}$$

- 점 $(x_1, g_1)$ 과 $(x_3, g_3)$ 를 기반으로 한 선형보간으로 개선된 근 $x_4$ 를 구할 수 있다.

$$x_4 = x_3 - g_3 \frac{x_3 - x_1}{g_3 - g_1} = x_3 - f_3 e^{hQ} \frac{x_3 - x_1}{f_3 e^{hQ} - f_1}$$

- 식 (c) 의 표현을 대입하여 정리하면 다음과 같은 식을 얻는다.

$$x_4 = x_3 \pm (x_3 - x_1) \frac{f_3}{\sqrt{f_3^2 - f_1 f_2}} \qquad (4.3)$$

$x_4$ 는 정확한 근은 아닌 근사값이다.

- $f_1 - f_2 > 0$ 인 경우 + 부호를, $f_1 - f_2 < 0$ 인 경우 – 부호를 선택한다. 이렇게 얻어진 $x_4$ 를 기준으로 근이 있는 구간으로 구간을 재설정하고 $x_i$ 의 연속 값 사이의 차이가 한계 내로 들어올 때까지 반복한다.

# ridder 프로그램

```
## module ridder
''' root = ridder(f,a,b,tol=1.0e-9).
   Finds a root of f(x) = 0 with Ridder's method.
   The root must be bracketed in (a,b).
'''
import error
import math
from numpy import sign

def ridder(f,a,b,tol=1.0e-9):
   fa = f(a)
   if fa == 0.0: return a 근이 구간 양끝점에 있으면 종료
   fb = f(b)
   if fb == 0.0: return b
   !! 교과서에 오류가 있는 코드 (삭제됨)
   for i in range(30):  최대 반복 횟수 = 30 번 고정
    # Compute the improved root x from Ridder's
formula
      c = 0.5*(a + b); fc = f(c)  구간의 중간점 계산
      s = math.sqrt(fc**2 - fa*fb)
      if s == 0.0: return None
      dx = (c - a)*fc/s                        식 (4.3) 계산
      if (fa - fb) < 0.0: dx = -dx
      x = c + dx; fx = f(x)
    # Test for convergence
      if i > 0:
         if abs(x - xOld) < tol*max(abs(x),1.0): return x
      xOld = x
    # Re-bracket the root as tightly as possible
      if sign(fc) == sign(fx):
         if sign(fa)!= sign(fx): b = x; fb = fx
         else: a = x; fa = fx
      else:
         a = c; b = x; fa = fc; fb = fx
   return None
   print('Too many iterations')
```

# 예제 4.4

Ridder 방법으로 (0.6, 0.8) 에 있는 $f(x) = x^3 - 10x^2 + 5 = 0$ 의 근을 구하시오.

[풀이] 시작점은 다음과 같다.

$$x_1 = 0.6 \quad f_1 = 0.6^3 - 10(0.6)^2 + 5 = 1.6160$$

$$x_2 = 0.8 \quad f_2 = 0.8^3 - 10(0.8)^2 + 5 = -0.8880$$

첫번째 반복: 이분법으로 다음 점을 산출한다.

$$x_3 = 0.7 \qquad f_3 = 0.7^3 - 10(0.7)^2 + 5 = 0.4430$$

Ridder 공식을 사용하여 근의 개선된 추정값을 계산한다.

$$s = \sqrt{f_3^2 - f_1 f_2} = \sqrt{0.4330^2 - 1.6160(-0.8880)} = 1.2738$$

$$x_4 = x_3 \pm (x_3 - x_1)\frac{f_3}{s}$$

$f_1 > f_2$ 이므로 + 부호를 선택한다. 그러면 추정값과 그에 해당되는 함수 값은

$$x_4 = 0.7 + (0.7 - 0.6)\frac{0.4430}{1.2738} = 0.7348 \qquad f_4 = 0.7348^3 - 10(0.7348)^2 + 5 = -0.0026$$

# 예제 4.4 (continued)

근이 간격 $(x_3, x_4)$ 에 있으므로 간격을 수정한다.

$$x_1 \leftarrow x_3 = 0.7 \qquad f_1 \leftarrow f_3 = 0.4430$$

$$x_2 \leftarrow x_4 = 0.7348 \qquad f_2 \leftarrow f_4 = -0.0026$$

두번째 반복

$$x_3 = 0.5(x_1 + x_2) = 0.5(0.7 + 0.7348) = 0.7174$$

$$f_3 = 0.7174^3 - 10(0.7174)^2 + 5 = 0.2226$$

$$s = \sqrt{f_3^2 - f_1 f_2} = \sqrt{0.2226^2 - 0.4430(-0.0026)} = 0.2252$$

$$x_4 = x_3 + (x_3 - x_1)\frac{f_3}{s} \leftarrow f_1 > f_2$$

$$x_4 = 0.7174 + (0.7174 - 0.7)\frac{0.2226}{0.2252} = 0.7346 \qquad f_4 = 0.7346^3 - 10(0.7346)^2 + 5 = 0.0000$$

그러므로 근은 $x = 0.7346$ 이며 소수점 이하 4자리까지 정확하다.

# 예제 4.5

함수의 근을 Ridder 프로그램을 이용하여 구하라.

$$f(x) = \frac{1}{(x-0.3)^2 + 0.01} - \frac{1}{(x-0.8)^2 + 0.04}$$

[풀이] 그래프를 이용하여 함수의 형태를 살핀다. 근은 $(0,1)$ 에 있음을 확인한다. 다음 프로그램을 이용하여 계산한다.

```
## example4_5
from ridder import *
def f(x):
    a = (x - 0.3)**2 + 0.01
    b = (x - 0.8)**2 + 0.04
    return 1.0/a - 1.0/b

print("root =",ridder(f,0.0,1.0))
```

```
root = 0.5800000000000001
```