# 개방법 근 구하기

이병옥
아주대 기계공학과

# 개요

- 개방법으로 근을 구하는 방법은 근이 존재하는 구간을 정할 필요가 없다. 근이 있을 것으로 추정하는 어떤 위치라도 초기 추정값으로 설정하여 근을 찾아 간다.
  - 단점은 구간법과 달리 근으로 항상 수렴하는 것은 아닌 점이다.
  - 장점으로서는 구간법에 비해 비교적 근을 찾는 속도가 빠르다.

- 주요 개방법
  - Newton-Raphson 법 – 가장 널리 사용되며 수렴 속도가 매우 빠르다.
  - secant 법 (앞에서 간단하게 소개하였으나 여기서는 다루지 않는다.)
  - Brent 법 (생략함)

# Newton-Raphson 법

- 간단하고 가장 빠른 방법이다.
  - 유일한 단점은 **함수의 1차 도함수를 사용**하므로 미분을 쉽게 할 수 있는 문제에서만 사용 가능
- Newton-Raphson 공식은 $x$ 에 대한 $f(x)$ 의 Taylor 시리즈 확장에서 유도될 수 있다.

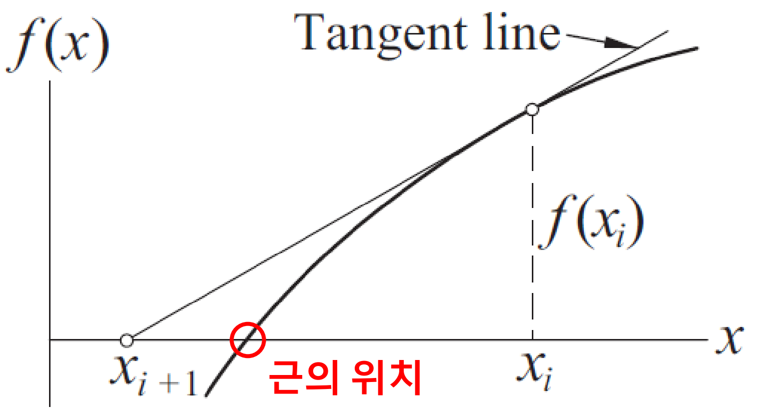
$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + O(x_{i+1} - x_i)^2$$

  - 여기서 $O(z)$ 는 "z 의 차수" 이다. $x_{i+1}$ 이 $f(x) = 0$ 의 근인 경우 위식은

$$0 = f(x_i) + f'(x_i)(x_{i+1} - x_i) + O(x_{i+1} - x_i)^2$$

  - $x_i$ 가 $x_{i+1}$ 에 가깝다고 가정하면 위식의 마지막 항은 무시할 만큼 작아서 없앨 수 있고, $x_{i+1}$ 에 대해 풀면 다음과 같은 Newton-Raphson 공식이 된다.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad (4.3)$$




- 이 관계를 그래프로 나타내면 그림과 같다.
  - $x_{i+1}$ 은 $x$ 축과 접선의 교차점에 있다.
  - N-R 법에서는 다음의 수렴기준이 만족할 때가지 반복한다.

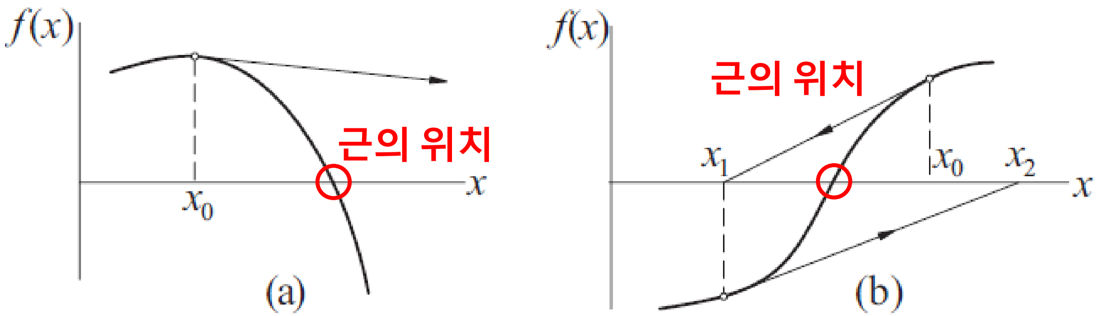
$$|x_{i+1} - x_i| < \varepsilon$$ **허용 오차**

# Newton-Raphson 법의 오차

- N-R 법의 알고리즘은 다음과 같다.

$f(x) = 0$ 에 대한 초기 추정값을 선택한다.
$|\Delta x| < \varepsilon$ 이 될 때까지 반복한다:

$$\Delta x = -f(x)/f'(x)$$
$$x \leftarrow x + \Delta x$$




- N-R 공식의 절단오차 $E$ 는 다음과 같다.

$$E_{i+1} = -\frac{f''(x)}{2f'(x)} E_i^2$$

  - 여기서 $x$ 는 근이다. 이 수식은 2차적으로 수렴됨을 나타낸다. (오차는 이전 단계 오차의 제곱)
  - 의미 있는 유효숫자의 수가 모든 반복에서 대략 두배가 된다.
  - N-R 법은 근의 근처에서는 수렴속도가 빠르지만 넓은 영역에서는 그리 빠르지 않다. 이유는 접선이 항상 함수의 근사값인 것은 아니다. **이분법과 결합하면 거의 안전한 방법**이 된다. (위의 그림 참조)

# newtonRaphson 프로그램

다음 안전한 버전의 프로그램은 계산할 근이 (a, b) 구간에 있다고 가정한다. 구간의 중간값을 초기 추정값으로 사용되며, 각 반복 후에 구간이 갱신된다. N-R 반복이 구간 내에 있지 않으면 무시되고 이분법으로 대체된다.

```
## module newtonRaphson
''' root = newtonRaphson(f,df,a,b,tol=1.0e-9).
    Finds a root of f(x) = 0 by combining the Newton-Raphson
    method with bisection. The root must be bracketed in (a,b).
    Calls user-supplied functions f(x) and its derivative df(x).
'''
def newtonRaphson(f,df,a,b,tol=1.0e-9):
    import error
    from numpy import sign
    fa = f(a)              구간의 양끝점에 해가 있으면 종료
    if fa == 0.0: return a
    fb = f(b)
    if fb == 0.0: return b           구간 내에 해가 있는지 검사
    if sign(fa) == sign(fb): error.err('Root is not bracketed')
    x = 0.5*(a + b)  중간점을 초기값으로 설정
    for i in range(30):   최대 반복 횟수를 30으로 설정
        fx = f(x)
        if fx == 0.0: return x
        # Tighten the brackets on the root  구간 축소
        if sign(fa) != sign(fx): b = x
        else: a = x
        # Try a Newton-Raphson step   N-R 법 적용
        dfx = df(x)
        # If division by zero, push x out of bounds
        try: dx = -fx/dfx
        except ZeroDivisionError: dx = b - a
        x = x + dx
        # If the result is outside the brackets, use bisection
        if (b - x)*(x - a) < 0.0:        근이 구간 밖에 있으면 이분법 적용
            dx = 0.5*(b - a)
            x = a + dx
        # Check for convergence
        if abs(dx) < tol*max(abs(b),1.0): return x
    print('Too many iterations in Newton-Raphson')
```

# 예제 4.6

Newton-Raphson 법으로 두 정수의 비율로 $\sqrt{2}$ 의 근사값을 연속으로 구하라.

[풀이] 이 문제는 $f(x) = x^2 - 2 = 0$ 의 근을 찾는 것과 같다. N-R 공식은 다음과 같다.

$$x \leftarrow x - \frac{f(x)}{f'(x)} = x - \frac{x^2 - 2}{2x} = \frac{x^2 + 2}{2x}$$

$x = 1$ 로 시작하여 연속 반복을 하면 다음과 같다.

$$x \leftarrow \frac{(1)^2 + 2}{2(1)} = \frac{3}{2}$$

$$x \leftarrow \frac{(3/2)^2 + 2}{2(3/2)} = \frac{17}{12}$$

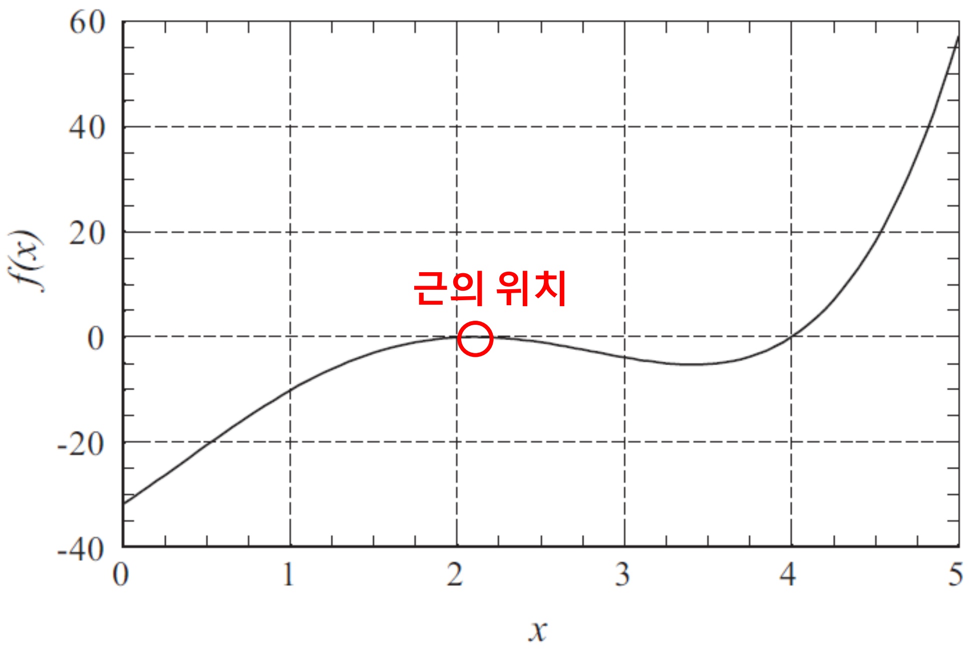
$$x \leftarrow \frac{(17/12)^2 + 2}{2(17/12)} = \frac{577}{408} = 1.1414216$$

마지막 값은 실제 $\sqrt{2} = 1.1414214$ 에 매우 가깝다. 시작값에 따라 다른 비율을 만들어 낸다.

# 예제 4.7

아래 함수에서 가장 작은 양의 근을 찾아라.

$$f(x) = x^4 - 6.4x^3 + 6.45x^2 + 20.538x - 31.752$$




[풀이] 함수의 그래프를 보면 가장 작은 양의 근이 $x = 2$ 근처에서 나타나고 이중근으로 보인다. 이분법과 Ridder 법에서는 근의 양쪽에서 부호가 바뀌는 것을 조사하여 근의 존재를 알아보기 때문에 이 경우에는 작동하지 않는다. N-R 법에서도 동일한 상황이 적용되지만 앞서 소개한 프로그램보다 이분법을 적용시키지 않은 순수한 N-R 법이 성공하지 못할 이유가 없으므로 활용하여 보기로 한다.

다음 프로그램을 실행한다. 실제 근의 값은 $x = 2.1$ 이다.

# 예제 4.7 (continued)

```
## example4_8
def f(x): return x**4 - 6.4*x**3 + 6.45*x**2 + 20.538*x - 31.752
def df(x): return 4.0*x**3 - 19.2*x**2 + 12.9*x + 20.538

def newtonRaphson(x,tol=1.0e-9):    변형이 없는 Newton-Raphson 법
    for i in range(30):
        dx = -f(x)/df(x)
        x = x + dx
        if abs(dx) < tol: return x,i
    print (f'Too many iterations\n')

root,numIter = newtonRaphson(2.0)   tol 입력이 없을 때는 기본값으로 설정
print(f'Root = {root}')
print(f'Number of iterations = {numIter}')
```

```
Root = 2.0999999786199406
Number of iterations = 22
```

하나의 다중 근 근처에서 N-R 법의 수렴이 반복 횟수가 많은 2차보다는 선형으로 나타난다. 식 (4.3)의 공식을 다음 식으로 대체하면 다중 근에 대한 수렴 속도를 높일 수 있다.

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

$m$ 은 근의 다중성이다. (이 문제에서는 $m = 2$) 프로그램을 변경하여 실행하여 보면 5번 만에 결과를 얻을 수 있다.

# 연립방정식

- 지금까지는 단일 방정식 $f(x) = 0$ 의 근을 구하는 것이었다면, 이번에는 같은 문제의 $n$ 차원에 대해 풀어보자.

$$\mathbf{f}\,(\mathbf{x}) = \mathbf{0}$$

  - 스칼라 표기법을 사용하면 다음과 같다.

$$\begin{gathered} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{gathered}$$

- $n$ 개의 **비선형 방정식을 동시에 만족하는 근을 찾는 것**으로 단일 방정식의 근을 찾는 것보다 훨씬 강력한 방법이다. 그러나, 문제는 신뢰할 만한 방법으로 근이 존재하는 구간을 찾을 수 없다는 것이다. 이런 경우 Newton-Raphson 법이 효과적이다. 적절한 시작점이 주어진다면 비선형 방정식에 잘 작동한다. 수렴 특성이 좋은 다른 많은 방법이 있지만 모두 Newton-Raphson 법의 변형이다.

# 연립방정식에 대한 Newton-Raphson 법 (1)

- 점 **x** 에 대한 $f_i$ (**x**) 의 Taylor 시리즈 확장을 살펴 보자.

$$f_i(\mathbf{x}+\Delta\mathbf{x}) = f_i(\mathbf{x}) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j}\Delta x_j + O(\Delta x^2) \qquad (4.5a)$$

  - $\Delta x^2$ 의 조건을 삭제하면 다음과 같이 쓸 수 있다.

$$\mathbf{f}\,(\mathbf{x}+\Delta\mathbf{x}) = \mathbf{f}\,(\mathbf{x}) + \mathbf{J}(\mathbf{x})\,\Delta\mathbf{x} \qquad (4.5b)$$

  - 여기서 **J**(**x**) 는 다음과 같은 편미분으로 구성된 자코비안(Jacobian) 행렬(크기 $n \times n$) 이다.

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

  - 식 (4.5b) 는 점 **x** 근처에서 벡터 값 함수 **f** 의 선형 근사값이다.

- **x** 가 **f** (**x**) = **0** 의 해의 현재 근사값이라고 가정하면 **x** + Δ**x** 를 개선된 해로 하자. 보정값 Δ**x** 를 찾기 위해 식 (4.5b) 에서 **f** (**x** + Δ**x**) = **0** 으로 설정한다. 결과는 Δ**x** 에 대한 연립방정식이다.

$$\mathbf{J}(\mathbf{x})\,\Delta\mathbf{x} = -\mathbf{f}\,(\mathbf{x}) \qquad (4.7)$$

# 연립방정식에 대한 Newton-Raphson 법 (2)

- 각각의 $\partial f_i/\partial x_j$ 의 분석적 도출은 어렵거나 비현실적일 수 있으므로 다음과 같이 유한한 차이를 가지는 근사값으로 컴퓨터가 계산하도록 하는 것이 바람직하다.

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i\left(\mathbf{x} + \mathbf{e}_j h\right) - f_i(\mathbf{x})}{h} \qquad (4.8)$$

  - 여기서 $h$ 는 작은 증분이고, $\mathbf{e}_j$ 는 $x_j$ 방향의 단위 벡터를 나타낸다. 이 공식은 차수 $\Delta x^2$ 를 제거하고 $\Delta\mathbf{x} = \mathbf{e}_j h$ 를 설정한 후 식 (4.5a)에서 얻을 수 있다. Newton-Raphson 법이 자코비안 행렬의 오류에 대해 민감하지 않기 때문에 식 (4.8) 의 근사식을 사용하는데 문제가 없다. 이 근사식을 사용하면 컴퓨터 코드에 $\partial f_i/\partial x_j$ 표현식을 입력하는 것을 피할 수 있다.

- 비선형 연립방정식에 대한 Newton-Raphson 법의 알고리즘은 다음과 같다. 성공적인 해로의 수렴은 초기 추정값의 선택에 의존한다.

  - 벡터 $\mathbf{x}$ 의 근에 대한 추정값을 선택한다.
  - $|\Delta x| < \varepsilon$ 을 만족할 때까지 반복한다:
    - 식 (4.8) 에서 자코비안 행렬, $\mathbf{J}(\mathbf{x})$ 을 계산한다.
    - $\Delta\mathbf{x}$ 에 대해 $\mathbf{J}(\mathbf{x})\,\Delta\mathbf{x} = -\mathbf{f}\,(\mathbf{x})$ 를 푼다.
    - $\mathbf{x}$ 값을 $\mathbf{x} + \Delta\mathbf{x}$ 로 변경한다.

# newtonRaphson2 프로그램

비선형 연립방정식을 Newton-Raphson 법으로 푸는 프로그램이다. 자코비언은 근사식을 통해 계산한다. 식 (4.7) 의 동시 방정식은 피벗팅을 통한 가우스 소거법으로 해결한다. (자코비언 행렬 계산은 매번 루프 내에서 실행하므로 이를 줄이기 위해 $x$ 값이 충분히 근에 가까운 경우 자코비언을 한번만 실행하여 계산 시간을 절약할 수 있도록 프로그램을 변경하는 것도 가능하다.)

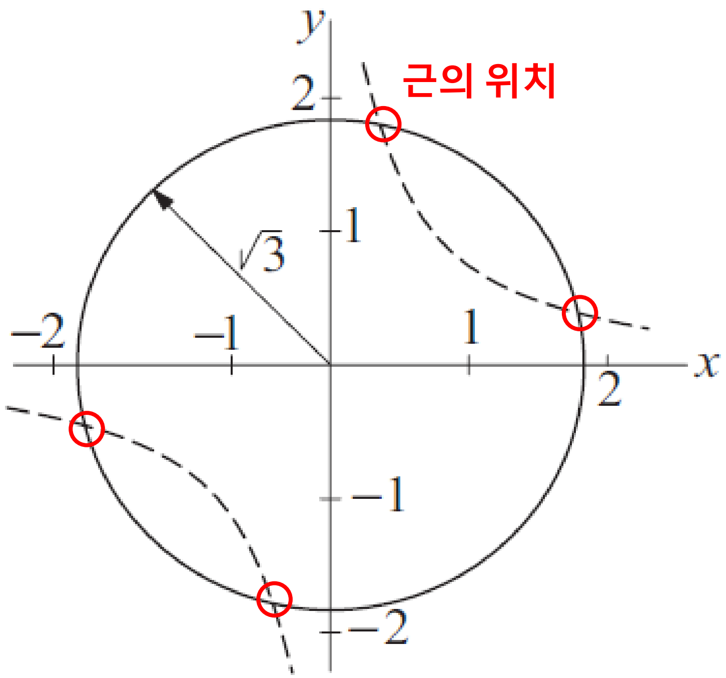
```
## module newtonRaphson2
''' soln = newtonRaphson2(f,x,tol=1.0e-9).
    Solves the simultaneous equations f(x) = 0 by
    the Newton-Raphson method using {x} as the initial
    guess. Note that {f} and {x} are vectors.
'''
import numpy as np
from gaussPivot import *
import math

def newtonRaphson2(f,x,tol=1.0e-9):
    def jacobian(f,x):   근사식을 이용한 자코비언 행렬 계산
        h = 1.0e-4
        n = len(x)
        jac = np.zeros((n,n))
        f0 = f(x)          주어진 모든 함수에 대해 계산 (벡터화)
        for i in range(n):
            temp = x[i]
            x[i] = temp + h
            f1 = f(x)          주어진 모든 함수에 대해 계산 (벡터화)
            x[i] = temp
            jac[:,i] = (f1 - f0)/h   하나의 열에서 행을 따라 계산
        return jac,f0

    for i in range(30):  반복횟수를 30번으로 설정
        jac,f0 = jacobian(f,x)                      근사값의 크기로 판정
        if math.sqrt(np.dot(f0,f0)/len(x)) < tol: return x
        dx = gaussPivot(jac,-f0)  식 (4.7) 계산
        x = x + dx
        if math.sqrt(np.dot(dx,dx)) < tol*max(max(abs(x)),1.0):
            return x               허용한계 내에 들어오면 종료
    print(Too many iterations')
```

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i\left(\mathbf{x} + \mathbf{e}_j h\right) - f_i(\mathbf{x})}{h}$$

# 예제 4.8




원 $x^2 + y^2 = 3$ 과 쌍곡선 $xy = 1$ 의 교점을 구하라.

[풀이] 이를 방정식으로 나타내면,

$$f_1(x, y) = x^2 + y^2 - 3 = 0$$

$$f_2(x, y) = xy - 1 = 0$$

정의에 따른 Jacobian 행렬을 구하면,

$$\mathbf{J}(x, y) = \begin{bmatrix} \dfrac{\partial f_1}{\partial x} & \dfrac{\partial f_1}{\partial y} \\ \dfrac{\partial f_2}{\partial x} & \dfrac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix}$$

따라서 Newton-Raphson 법과 관련된 선형방정식 $\mathbf{J}(\mathbf{x})\, \Delta\mathbf{x} = -\mathbf{f}\,(\mathbf{x})$ 은 다음과 같다.

$$\begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} -x^2 - y^2 + 3 \\ -xy + 1 \end{bmatrix} \qquad \text{(c)}$$

그래프를 살펴 보면 4개의 교점이 있다. 그러나 서로 대칭이므로 이 중 하나만 찾으면 다른 점은 쉽게 추론이 가능하다. 시작점으로서 $x = 0.5, y = 1.5$ 를 선택한다.

**첫번째 반복:** 식 (c)에 $x = 0.5, y = 1.5$ 를 대입하면,

# 예제 4.8 (continued)

$$\begin{bmatrix} 1.0 & 3.0 \\ 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} 0.50 \\ 0.25 \end{bmatrix}$$

해는 $\Delta x = \Delta y = 0.125$ 이다. 따라서 개선된 교차점의 좌표는

$$x = 0.5 + 0.125 = 0.625 \qquad y = 1.5 + 0.125 = 1.625$$

**두번째 반복**: $x$ 와 $y$ 의 최신 값을 사용하여 절차를 반복하면

$$\begin{bmatrix} 1.250 & 3.250 \\ 1.625 & 0.625 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} -0.031250 \\ -0.015625 \end{bmatrix}$$

결과는 $\Delta x = \Delta y = -0.00694$ 따라서

$$x = 0.625 - 0.00694 = 0.61806 \qquad y = 1.625 - 0.00694 = 1.61806$$

**세번째 반복**: 다시 최신 값을 사용하여 반복하면

$$\begin{bmatrix} 1.23612 & 3.23612 \\ 1.61806 & 0.61806 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} -0.000116 \\ -0.000058 \end{bmatrix}$$

결과는 $\Delta x = \Delta y = -0.00003$ 따라서

$$x = 0.61806 - 0.00003 = 0.61803 \qquad y = 1.6180625 - 0.00003 = 1.61803$$

5개의 유효숫자 내에서 변화가 없으므로, 교점의 좌표는 $\pm(0.61803, 1.61803), \pm(1.61803, 0.61803)$

# 예제 4.8 (continued)

**대체 해**

방정식이 몇 개만 있으면 미지수 중 하나를 제외하고 모두 제거가 가능하다. 그러면 단일 방정식을 얻어 더 쉽게 풀 수 있다. 이 문제는 방정식 중 두번째 방정식에서 다음 관계를 얻는다.

$$y = \frac{1}{x}$$

이 식을 첫번째 방정식에 대입하여 식을 변경하면,

$$x^2 + \frac{1}{x^2} - 3 = 0 \quad \rightarrow \quad x^4 - 3x^2 + 1 = 0$$

위의 2차 방정식의 해는 $x = \pm 0.61803, \pm 1.61803$ 이며 이는 앞서 구한 결과와 일치한다.

# 예제 4.9

newtonRaphson 프로그램을 이용하여 아래 방정식의 해를 찾아라. 시작점은 (1, 1, 1) 로 하라.

$$\sin x + y^2 + \ln z - 7 = 0$$
$$3x + 2^y - z^3 + 1 = 0$$
$$x + y + z - 5 = 0$$

[풀이] 다음 프로그램을 실행한다.

```
## example4_10
import numpy as np
import math
from newtonRaphson2 import *
def f(x):
    f = np.zeros(len(x))
    f[0] = math.sin(x[0]) + x[1]**2 + math.log(x[2]) - 7.0
    f[1] = 3.0*x[0] + 2.0**x[1] - x[2]**3 + 1.0
    f[2] = x[0] + x[1] + x[2] - 5.0
    return f      함수 3개의 계산결과를 리스트로 만들어 반환
x = np.array([1.0, 1.0, 1.0])
print(newtonRaphson2(f,x))
```

```
[ 0.59905376 2.3959314 2.00501484]
```

(x, y, z) 결과