다항식의 근

김기택
국민대학교 소프트웨어학과

개요

- 차수 n 의 다항식은 다음과 같다. 여기서 계수 a_i 는 실수 또는 복소수이다.

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

  - 우선 실수의 근을 가진 다항식을 다루지만, 이 절에 제시된 알고리즘은 복소수 근에도 적용된다.
  - 다항식 $P_n(x) = 0$ 의 근은 정확히 n 개를 가지며 실수 또는 복소수 이다. 계수가 실수인 경우 복소수 근은 항상 공액(conjugate) 쌍 $(x_r + ix_i, x_r - ix_i)$ 으로 발생한다. 여기서 x_r 과 x_i 는 각각 실수부와 허수부 이다.
- 실수 계수의 경우 실근의 수는 데카르트의 규칙에서 추정할 수 있다.
  - 양의 실수 근은 $P_n(x)$ 식의 부호 변화 수와 같거나 짝수 만큼 적다.
  - 음의 실수 근은 $P_n(-x)$ 의 부호 변화 수와 같거나 짝수 만큼 적다.
  - 예를 들어 $P_3(x) = x^3 - 2x^2 - 8x + 27$ 을 살펴 보자. 부호가 두 번 바뀌기 때문에 $P_3(x) = 0$ 은 2 개 또는 0 개의 양의 실근을 가진다.
  - $P_3(-x) = -x^3 - 2x^2 + 8x + 27$ 은 단일 부호 변화를 포함한다. 따라서 $P_3(x) = 0$ 은 하나의 음의 실근을 가진다.
- 복소수 근을 계산하려면 다항식을 전문적으로 다루는 방법을 사용한다. – Laguerre 방법
  - 필요한 수치 도구: 다항식과 그 유도식을 평가하는 효율적인 알고리즘, 다항식 축소 방법

다항식의 평가 (1)

- 다음 알고리즘에 의해 식 (4.9)의 다항식을 왼쪽에서 오른쪽으로 평가하려 한다. 계수가 배열 **a** 에 저장된다.

```
p = 0.0
for i in range(n+1):
   p = p + a[i]*x**i
```

$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$

- x^k 는 $x \times x \times \cdots \times x$ $(k - 1)$ 곱셈으로 평가되므로 이 알고리즘의 곱셈 수는 다음과 같다.

$$1 + 2 + 3 + \cdots + (n-1) = \frac{1}{2}n(n-1)$$

- n 이 크면 다항식을 오른쪽에서 왼쪽으로 평가하면 곱셈 수가 상당히 줄어들 수 있다. 예를 들어,

$$P_4(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

- 다항식을 다음과 같이 정리한 후

$$P_4(x) = a_0 + x\{a_1 + x[a_2 + x(a_3 + a_4 x)]\}$$

- 선호하는 계산 순서가 명확해진다.

$$P_0(x) = a_4, P_1(x) = a_3 + xP_0(x), P_2(x) = a_2 + xP_1(x), P_3(x) = a_1 + xP_2(x), P_4(x) = a_0 + xP_3(x)$$

다항식의 평가 (2)

- 차수 n 다항식의 경우 절차는 다음과 같고 다음 알고리즘으로 이어진다.

$$P_0(x) = a_n$$
$$P_i(x) = a_{n-1} + xP_{i-1}(x), \qquad i = 1,2,\dots,n \qquad (4.10)$$

```
p = a[n]
for i in range(1, n+1):
    p = a[n - i] + p*x
```

  - 마지막 알고리즘은 n 곱셈만 포함하므로 $n > 3$ 에 더 효율적이다. 그러나 이 알고리즘이 좋은 이유는 단순히 경제성 때문은 아니다. 곱셈 수가 적은 과정이 반올림 오차 누적을 적게 발생한다.

- Laguerre 방법을 포함한 일부 해 찾기 알고리즘도 $P_n(x)$ 의 1차 및 2차 미분 값을 평가해야 한다.

$$P_0'(x) = 0 \quad P_i'(x) = P_{i-1}(x) + xP_{i-1}'(x), \qquad i = 1,2,\dots,n \quad (4.11a)$$
$$P_0''(x) = 0 \quad P_i''(x) = 2P_{i-1}'(x) + xP_{i-1}''(x), \qquad i = 1,2,\dots,n \quad (4.11b)$$

evalPoly

다항식과 그 도함수들을 평가하는 함수는 다음과 같다.

```
## module evalPoly
''' p,dp,ddp = evalPoly(a,x).
    Evaluates the polynomial
    p = a[0] + a[1]*x + a[2]*x^2 +...+ a[n]*x^n
    with its derivatives dp = p' and ddp = p'' at x.
'''
def evalPoly(a,x):
    n = len(a) - 1
    p = a[n]                 a_n
    dp = 0.0 + 0.0j
    ddp = 0.0 + 0.0j
    for i in range(1,n+1):   a_n 는 건너 뛴다 (1부터 시작, n 까지)
        ddp = ddp*x + 2.0*dp   2차 도함수 평가
        dp = dp*x + p          1차 도함수 평가
        p = p*x + a[n-i]       다항식 평가
    return p,dp,ddp
```

다항식 축소 (1)

- $P_n(x)=0$ 의 해 r 이 계산된 후 다항식을 다음과 같이 인수분해하는 것이 바람직하다.

$$P_n(x)=(x-r)P_{n-1}(x) \qquad (4.12)$$

  - 축소(deflation) 또는 합성 분할(synthetic division)으로 알려진 이 절차는 $P_{n-1}(x)$ 의 계수를 계산하는 것에 지나지 않는다.

- $P_n(x)$ 의 나머지 해도 $P_{n-1}(x)$ 의 해이기 때문에 이제 해 찾기 절차를 $P_n(x)$ 가 아닌 $P_{n-1}(x)$ 에 적용할 수 있다.
  - 따라서 축소는 해를 찾을 때마다 다항식의 차수가 감소하기 때문에 연속으로 해 찾기를 점점 더 쉽게 해준다. 또한 이미 찾아진 해를 제거함으로써 동일한 해를 두 번 이상 계산할 가능성이 제거된다.

$$P_{n-1}(x)=b_0+b_1x+b_2x^2+\cdots+b_{n-1}x^{n-1}$$

  - 위 식을 고려하면 식 (4.12)는 다음과 같이 된다.

$$a_0+a_1x+a_2x^2+\cdots+a_{n-1}x^{n-1}+a_nx^n=(x-r)(b_0+b_1x+b_2x^2+\cdots+b_{n-1}x^{n-1})$$

다항식 축소 (2)

- x 의 거듭 제곱과 계수를 같게 하면 다음과 같이 된다.

$$b_{n-1} = a_n \quad b_{n-2} = a_{n-1} + rb_{n-1} \quad \cdots \quad b_0 = a_1 + rb_1$$

- 이는 Horner 의 축소 알고리즘과 연결된다.

```
b[n – 1] = a[n]
for i in range(n-2, -1, -1):
   b[i] = a[i+1] + r*b[i+1]
```

Laguerre 방법 (1)

- Laguerre 공식은 일반적인 다항식 $P_n(x)$ 에 대해 쉽게 도출되지 않는다. 그러나 다항식이 $x = r$ 에서 0 이고 $x = q$ 에서 $(n - 1)$ 개의 0 인 특수한 경우를 고려한다면 도출이 크게 단순화 된다. 따라서 다항식은 다음과 같이 쓸 수 있다.

$$P_n(x) = (x - r)(x - q)^{n-1} \qquad \text{(a)}$$

- 우리의 문제는 다음과 같다. 식 (a)의 다항식이 아래 형식으로 주어질 때 r 을 결정하는 것이다. (q 도 알려지지 않음)

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

  - 여기서 고려된 특별한 경우에 대한 정확한 결과는 다항식의 반복 공식을 적용하면 잘 작동한다. x 에 대해 식 (a) 를 미분하면

$$P_n'(x) = (x - q)^{n-1} + (n - 1)(x - r)(x - q)^{n-2} = P_n(x)\left(\frac{1}{x - r} + \frac{n - 1}{x - q}\right)$$

  - 따라서

$$\frac{P_n'(x)}{P_n(x)} = \frac{1}{x - r} + \frac{n - 1}{x - q} \qquad \text{(b)}$$

  - 이 미분에 따라 다음 결과를 가진다.

Laguerre 방법 (2)

$$\frac{P_n''(x)}{P_n(x)} - \left[\frac{P_n'(x)}{P_n(x)}\right]^2 = -\frac{1}{(x-r)^2} - \frac{n-1}{(x-q)^2} \qquad \text{(c)}$$

- 다음 표기법을 소개하는 것이 편리하다.

$$G(x) = \frac{P_n'(x)}{P_n(x)} \qquad H(x) = G^2(x) - \frac{P_n''(x)}{P_n(x)} \qquad (4.14)$$

- 따라서 식 (b)와 (c) 는 다음과 같다.

$$G(x) = \frac{1}{x-r} + \frac{n-1}{x-q} \qquad (4.15a)$$

$$H(x) = \frac{1}{(x-r)^2} + \frac{n-1}{(x-q)^2} \qquad (4.15b)$$

- 식 (4.15a) 를 $x-q$ 에 대해 풀고 결과를 식 (4.15b) 로 대입하면 $x-r$ 에 대한 2차 방정식을 얻는다. 이 방정식의 해는 Laguerre 의 공식이다. (분모가 더 큰 값이 되도록 부호를 선택한다.)

$$x - r = \frac{n}{G(x) \pm \sqrt{(n-1)[nH(x) - G^2(x)]}} \qquad (4.16)$$

Laguerre 방법으로 다항식 근 구하는 절차

- x 를 $P_n(x) = 0$ 의 근에 대한 추정값으로 설정 (아무 값이나 좋다.)
- $|P_n(x)| < \varepsilon$ 혹은 $|x - r| < \varepsilon$ (ε 는 허용 오차) 를 만족할 때까지 반복
  - $P_n(x), P_n'(x), P_n''(x)$ 를 evalPoly 함수를 이용하여 평가
  - 식 (4.14) 를 이용하여 $G(x)$, $H(x)$ 계산
    - 식 (4.16) 에서 분모의 더 큰 크기의 결과가 나타나도록 부호를 선택하여 근 r 을 개선한다.
  - $x \leftarrow r$ 변경

- Laguerre 방법의 장점 중 하나는 초기 추정값에 영향을 받지 않는 것이다.
  - 어떤 값을 선택하여도 수렴한다.

polyRoots 프로그램 (1)

이 모듈의 polyRoots 함수는 $P_n(x) = 0$ 의 모든 해를 계산한다. 여기서 다항식 $P_n(x)$ 의 계수 배열은 $\mathbf{a} = [a_0, a_1, \dots, a_n]$ 이다. 중첩 함수 Laguerre 에 의해 첫번째 해가 계산된 후 deflPoly 를 사용하여 다항식이 축소되고, 축소된 다항식에 Laguerre 를 적용하여 다음 해가 계산된다. 이 과정은 모든 n 개의 해가 발견될 때까지 반복된다. 계산된 근이 매우 작은 허수 부분일 경우 반올림 오차를 나타낼 가능성이 높다. 따라서 polyRoots 는 작은 허수 부분을 0 으로 대체한다.

```
## module polyRoots
''' roots = polyRoots(a).
    Uses Laguerre's method to compute all the roots of
    a[0] + a[1]*x + a[2]*x^2 +...+ a[n]*x^n = 0.
    The roots are returned in the array 'roots',
'''
from evalPoly import *
import numpy as np
import cmath    복소수 계산을 위해 cmath 모듈 사용
from random import random
```

polyRoots 프로그램 (2)

```
def polyRoots(a,tol=1.0e-12):

    def laguerre(a,tol):   Laguerre 함수        임의의 초기 값 선택
        x = random()     # Starting value (random number)
        n = len(a) – 1
        for i in range(30):
            p,dp,ddp = evalPoly(a,x)
            if abs(p) < tol: return x    다항식 평가 결과가 허용
            g = dp/p                     오차 한계보다 작으면 종료
            h = g*g - ddp/p
            f = cmath.sqrt((n - 1)*(n*h - g*g))
            if abs(g + f) > abs(g - f): dx = n/(g + f)
            else: dx = n/(g - f)   분모 값 중 큰 것을 선택
            x = x - dx
            if abs(dx) < tol: return x    (x – r) 이 허용 오차한계
        print('Too many iterations')      보다 작으면 종료
```

다항식 축소

```
    def deflPoly(a,root):      # Deflates a polynomial
        n = len(a)-1
        b = [(0.0 + 0.0j)]*n
        b[n-1] = a[n]
        for i in range(n-2,-1,-1):
            b[i] = a[i+1] + root*b[i+1]
        return b

# solve roots of polynomial    다항식 근 구하기
    n = len(a) - 1
    roots = np.zeros((n),dtype=complex)   복소수 형태의 배열
    for i in range(n):
        x = laguerre(a,tol)
        if abs(x.imag) < tol: x = x.real   근의 허수값이 허용한계보
        roots[i] = x                       다 작으면 실수값 만 반환
        a = deflPoly(a,x)   근을 구함에 따라 다항식 차수 축소
    return roots
```

예제 4.10

다항식 $P_4(x) = 3x^4 - 10x^3 - 48x^2 - 2x + 12$ 의 근은 $x = 6$ 이다. Horner의 알고리즘으로 다항식을 정의하라. 즉, $(x-6)P_3(x) = P_4(x)$ 가 되도록 $P_3(x)$ 를 찾아라.

[풀이]

$r = 6$ 이고 $n = 4$ 일 때, 식 (4.13) 은

$$b_3 = a_4 = 3$$

$$b_2 = a_3 + 6b_3 = -10 + 6(3) = 8$$

$$b_1 = a_2 + 6b_2 = -48 + 6(8) = 0$$

$$b_0 = a_1 + 6b_1 = -2 + 6(0) = -2$$

그러므로

$$P_3(x) = 3x^3 + 8x^2 - 2$$

예제 4.11

방정식 $P_3(x) = x^3 - 4.0x^2 - 4.48x + 26$ 의 해는 대략 $x = 3 - i$ 이다. Laguerre 반복 공식을 한 번 적용하여 이 해의 정확한 값을 찾아라.

[풀이] 주어진 근의 추정값을 시작값으로 사용하면,

$$x = 3 - i \quad x^2 = 8 - 6i \quad x^3 = 18 - 26i$$

이 값을 $P_3(x)$ 와 그 미분식에 대체하면

$$P_3(x) = x^3 - 4.0x^2 - 4.48x + 26.1 = (18 - 26i) - 4.0(8 - 6i) - 4.48(3 - i) + 26.1 = -1.34 + 2.48i$$

$$P_3'(x) = 3.0x^2 - 8.0x - 4.48 = 3.0(8 - 6i) - 8.0(3 - i) - 4.48 = -4.48 - 10.0i$$

$$P_3''(x) = 6.0x - 8.0 = 6.0(3 - i) - 8.0 = 10.0 - 6.0i$$

식 (4.14) 는

$$G(x) = \frac{P_3'(x)}{P_3(x)} = \frac{-4.48 - 10.0i}{-1.34 + 2.48i} = -2.36557 + 3.08462i$$

$$H(x) = G^2(x) - \frac{P_3''(x)}{P_3(x)} = (-2.36557 + 3.08462i)^2 - \frac{10.0 - 6.0i}{-1.34 + 2.48i} = 0.35995 - 12.48452i$$

예제 4.11 (continued)

식 (4.16) 에서 분모의 제곱근 기호 아래의 항목은

$$F(x) = \sqrt{(n-1)[nH(x) - G^2(x)]} = \sqrt{2[3(0.35995 - 12.48452i) - (-2.36557 + 3.08462i)^2]}$$
$$= 5.08670 - 4.49402i$$

이제 식 (4.16) 의 어느 부호가 더 큰 분모를 생성하는지 찾는다.

$$|G(x) + F(x)| = |(2.36557 + 3.08462i) + (5.08670 - 4.49402i)| = 3.06448$$
$$|G(x) - F(x)| = |(2.36557 + 3.08462i) - (5.08670 - 4.49402i)| = 10.62884$$

빼기 부호를 사용하면 식 (4.16) 은 해에 대해 다음과 같은 개선된 근사값을 얻는다.

$$r = x - \frac{n}{G(x) - F(x)} = (3 - i) - \frac{3}{-7.45227 + 7.57864i} = 3.19790 - 0.79875i$$

좋은 시작값 덕분에 이 근사값은 이미 정확한 값 $r = 3.20 - 0.80i$ 에 매우 가깝다.

예제 4.12

polyRoots 프로그램을 사용하여 $x^4 - 5x^3 - 9x^2 + 155x - 250 = 0$ 의 모든 근을 구하라.

[풀이] 다음 프로그램을 사용하면

```
## example4_12
from polyRoots import *
import numpy as np

c = np.array([-250.0,155.0,-9.0,-5.0,1.0])
print('Roots are:\n',polyRoots(c))
```

다음의 출력을 만든다.

```
Roots are:
[ 2.+0.j    4.-3.j    4.+3.j    -5.+0.j]
```