[수치해석] NA-rootfinding2

Polynomial (다항식)

$$p = \sum_i c_i x^i$$

Horner’s Theorem

위 다항식을 코드로 구현할 때, **곱셈의 횟수가 가장 적도록 하는 방법은 계속 중첩**하는 방법이다.

```
p=c[0]+x*(c[1]+x*(c[2]+x*(c[3]+x*c[4])));
```

```
p=(((c[4]*x+c[3])*x+c[2])*x+c[1])*x+c[0];
```

C code

```
    p=c[n];
    for(j=n-1;j>=0;j--) p=p*x+c[j];

or

    p=c[j=n];
    while (j>0) p=p*x+c[--j];
```

- 곱셈 횟수가 n번이 된다.

p, dp를 값이 아니라 p는 다항식, dp는 p를 미분했을 때의 다항식이라고 생각하면, 아래의 코드는 **p, dp를 Horner’s method에 기반하여 생성하는 코드**이다.

```
p=c[n];
dp=0.0;
for(j=n-1;j>=0;j--) {dp=dp*x+p; p=p*x+c[j];}
```

or

```
p=c[j=n];
dp=0.0;
while (j>0) {dp=dp*x+p; p=p*x+c[--j];}
```

- dp = dp*x + p 인 이유는 n-1번 반복하면 자연스럽게 dp의 최고차항인 $(n-1)c_n x^{n-1}$이 되는 것을 생각해보면 알 수 있다.

Multiplication

$c(x)$가 다항식일 때, $d(x) = (x-a)c(x)$의 새로운 계수는 아래와 같이 구할 수 있다.

```
c[n]=c[n-1];
for (j=n-1;j>=1;j--) c[j]=c[j-1]-c[j]*a;
c[0] *= (-a);
```

Synthetic division (조립제법)

$c(x)$가 다항식일 때, $c(x) = (x-a)q(x) + r$에서의 $q(x)$의 새로운 계수는 아래와 같이 구할 수 있다.

```
rem=c[n];
c[n]=0.0;
for(i=n-1;i>=0;i--) {
    swap=c[i];
    c[i]=rem;
    rem=swap+rem*a;
}
```

- 직접 조립제법을 해보면 같은 결과를 얻을 수 있다.
- 조립제법의 결과로 **차원이 감소**한다.
    - **Deflation**은 Root finding으로 Root $x = a$를 찾았을 때, **a를 제외한 다른 Root를 찾기 위해 수식, 함수 f(x)를 $f(x) = q(x)(x-a) + r$에서의 q(x)로 변환**하는 과정이다.
    - a를 제외한 f(x)의 나머지 Root와 q(x)의 Root는 동일하다.
    - f(x)에 대해 Root를 찾기 보다, **q(x)를 이용하여 Root를 찾는 것이 쉽고 효율적**이다.

Deflation method

Root를 찾은 이후, f(x)를 더 쉬운 함수 q(x)로 변환하여 다른 Root를 찾도록 하는 방법

과정

1. f(x)에 대한 Root $x = a$를 찾았다고 가정하자.
2. $f(x) = q(x)(x-a)$를 만족할 것이기 때문에 **f(x) 대신 q(x)를 이용하여 다른 Root를 찾도록 한다.**

a. 이후 q(x)에 대해 **Root finding method**를 진행한다.

b. Root finding method에는 **Open method, Bracketing method가 포함**된다.

Bairstow's method

알고리즘 자체가 Deflation method처럼 동작하는 Root를 찾는 알고리즘 (방법)

- Root finding method로 따져보자면 Open method에 속한다.

Basic idea

1. $f(x) = q(x)(x-a) + r$은 항상 만족하는 수식이다.
2. 만약 a가 f(x)의 Root라면, r = 0이 될 것이다.
3. 그렇다면 **Synthetic Division**에 따라 f(x) = q(x)(x-a)에서 **q(x)의 식**을 얻을 수 있다.
   a. 이때, 위에서 본 **Synthetic Division의 코드를 이용해 효율적으로 q(x)의 식**을 얻을 수 있다.
4. **q(x)의 식으로 Root를 찾는 것은 f(x)에서 a를 제외한 나머지 root를 찾는 것과 동일하다.**

과정

1. $f(x)$가 다항식일 때, $f(x) = (x^2 - rx - s)(b_2 + b_3x + ... + b_nx^{n-2}) + b_1(x-r) + b_0$ 으로 나타낼 수 있다.
2. (1)의 식에서 r, s를 잘 설정하면 b_1, b_0가 0이 된다.
   a. 이는 곧 나머지 $b_1(x-r) + b_0$이 0이 되어 $x^2 - rx - s = 0$**의 두 켤레복소수 근이 f(x)의 근이 되는 것과 동일하다.**
3. 나머지가 0이 될 때까지 **r, s를 수정**한다.
4. 나머지가 0에 가장 근접할 때의 r, s값을 이용하여 $f(x)$를 $(x^2 - rx - s)$로 **Synthetic division하여 Deflation한다.**

5. (4)에서 얻은 몫, $q(x) = (b_2 + b_3x + ... + b_nx^{n-2})$에 대해 (2) ~ (4) 과정을 반복한다.
   a. **반복 종료 조건은 q(x)가 2차 또는 1차인 경우**이다.

q(x)의 계수 b에 대한 정의

$$f(x) = (x^2 - rx - s)(b_2 + b_3x + ... + b_nx^{n-2}) + b_1(x - r) + b_0$$

위 식에 대해 조립 제법을 수행하면, b_i는 아래와 같이 정의된다.

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + rb_n \\ b_i &= a_i + rb_{i+1} + sb_{i+2} \qquad \text{for } i = n-2 \text{ to } 0 \end{aligned}$$

- b_i는 **r, s에 대한 함수 형식**이 된다.

r, s를 업데이트하는 자세한 계산 과정

우리가 **알고 있는 (또는 초기값) r, s 값에 대해 업데이트 할 양**인 $\Delta r, \Delta s$를 알아내는 것이 목표이다.

- 이를 위해 $b_1(r + \Delta r, s + \Delta s), b_0(r + \Delta r, s + \Delta s)$를 Taylor Series로 예측할 수 있다.
- $b_1(r + \Delta r, s + \Delta s) = b_0(r + \Delta r, s + \Delta s) = 0$ 이 되도록 하는 $\Delta r, \Delta s$를 알아내는 것이 **목표**이다.

1차 Taylor 전개를 수행하면 다음과 같다.

$$b_1(r + \Delta r, s + \Delta s) = b_1 + \frac{\partial b_1}{\partial r}\Delta r + \frac{\partial b_1}{\partial s}\Delta s$$
$$b_0(r + \Delta r, s + \Delta s) = b_0 + \frac{\partial b_0}{\partial r}\Delta r + \frac{\partial b_0}{\partial s}\Delta s$$

$b_1(r+\Delta r, s+\Delta s) = b_0(r+\Delta r, s+\Delta s) = 0$ 이 되는 것을 원하기 때문에 $b_1(r+\Delta r, s+\Delta s) = b_0(r+\Delta r, s+\Delta s) = 0$ **임을 가정**해서 위 수식을 아래와 같이 정리할 수 있다.

- 아래 연립방정식을 계산하면 $\Delta r, \Delta s$를 얻을 수 있다.

$$\frac{\partial b_1}{\partial r}\Delta r + \frac{\partial b_1}{\partial s}\Delta s = -b_1$$
$$\frac{\partial b_0}{\partial r}\Delta r + \frac{\partial b_0}{\partial s}\Delta s = -b_0$$

$q(x) = (b_2 + b_3 x + ... + b_n x^{n-2})$**의 수식을 한 번 더** $x^2 - rx - s$**로 나눈 조립제법 꼴로 정리하고 그 계수를** c_i로 나타낸다면 b에 대한 **편미분을 직접 계산하는 대신 더 효율적으로 위 연립방정식을 해결할 수 있다.**

- q(x)에 b_0, b_1이 $b_0, b_1 c_0, c_1$이 b_0, b_1에 의해 정의되는데 이것은 조립제법 알고리즘 자체에서 점화식을 정의하려면 b_0, b_1이 존재해야 하기 때문이다.

$$c_n = b_n$$
$$c_{n-1} = b_{n-1} + r c_n$$
$$c_i = b_i + r c_{i+1} + s c_{i+2}$$
$$\text{for } i = n-2 \text{ to } 1$$

c_i를 위와 같이 정의하니 b_1, b_0를 r, s에 대해 각각 편미분한 결과도 우연히 c_i로 표현할 수 있었다.

$$\partial b_0/\partial r = c_1$$
$$\partial b_0/\partial s = \partial b_1/\partial r = c_2$$
$$\partial b_1/\partial s = c_3$$

그럼 이제 아까 식에서의 **편미분 값을 c_i로 치환한 더욱 간단한 형태로 표현**할 수 있다.

$$c_2\Delta r + c_3\Delta s = -b_1$$
$$c_1\Delta r + c_2\Delta s = -b_0$$

위 식을 연립방정식 형태로 계산하여 $\Delta r, \Delta s$를 구한다.

Stop-condition

- r, s**를 업데이트 하는 반복을 멈추는 조건**은 $|\frac{\Delta r}{r}|, |\frac{\Delta s}{s}| < StoppingCriterion$ 이다.
    - 이때 두 Root는 $x = \frac{r \pm \sqrt{r^2+4s}}{2}$ 이다.

- **Deflation을 멈추는 조건은 Synthetic Division 후 나오는 몫 q(x)가 2차거나 1차인 경우이다.**
    - 3차 이상이라면 반복한다.

Deflation method와 Bairstow's method는 f(x)가 Polynomial인 경우에만 사용이 가능한 방법이다!

Lagerre method

가정

- $P(x) = (x - x_1)(x - x_2)\ldots(x - x_n)$ 일 때, $x_1 \ldots x_n$은 모두 $P(x)$의 Root 이다.
- $ln|P(x)| = ln|x - x_1| + ln|x - x_2| + \ldots + ln|x - x_n|$
- 우리가 Root로 추정한 값이 x^*라고 할 때, x^***가 $P(x)$의 Roots 중 x_1과는 매우 가깝고 나머지 Roots ($x_2 \ldots x_n$은 모두 한 점 근처에 모여있다고 가정**하자.

알고리즘

1. 우선 x^*에 대해 다음과 같이 G, H를 정의한다.

$$G = \frac{dln|P(x^*)|}{dx^*} = \sum \frac{1}{x^* - x_i} = \frac{P'(x^*)}{P(x^*)}$$

$$H = \frac{d^2ln|P(x^*)|}{dx^{*2}} = \sum \frac{1}{(x^* - x_i)^2} = [\frac{P'(x^*)}{P(x^*)}]^2 - \frac{P'(x^*)}{P(x^*)}$$

2. $a = x^* - x_1$이라고 하면, a는 **현재 추정치 x^*에서 가장 가까운 근과의 거리**가 된다.
    a. **현재 추정치에서 가장 가까운 근으로 이동시켜 가장 빠르게 Root를 찾는 효율적인 방법**이다.
3. $b = x^* - x_i$ ($i = 2, 3, \ldots, n$)이라고 하면 b는 **현재 추정치 x^*에서 가장 가까운 근을 제외한 나머지 근과의 거리**가 된다.
4. (2)와 (3)과 같이 a, b를 잡으면, G, H를 a, b에 대해 재정의할 수 있다.

$$G = \frac{1}{a} + \frac{(n-1)}{b}$$

$$H = \frac{1}{a^2} + \frac{(n-1)}{b^2}$$

5. 위 G, H를 연립하여 a에 대해 정리하면 다음과 같다.

$$a = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}}$$

6. (5)의 a를 이용하여 $x^* = x^* - a$로 업데이트한다.
7. (1) ~ (6)의 과정을 a가 0에 충분히 가까워질 때까지 반복한다.

위 알고리즘을 **Deflation Method에서 Root finding algorithm으로 사용**할 수 있다.

- 특정 함수, 다항식의 모든 근을 찾는 경우 **Deflation Method와 결합하여 사용할 수 있다.**
- 알고리즘 자체가 Deflation을 포함하지 않지만, **Deflation Method와 결합하여 사용하면 강력하다.**