# Homework#3

2015004302 곽상원

# 목차



---

## 1. Purpose of program

-이 프로그램은 비선형방정식의 해를 구하는 프로그램으로, 널리 알려진 methods를 이용해 해를 구하고 각 method의 iteration횟수를 측정한다.
-**Bessel function(bessj0.c)**을 대상으로 [1.0,10.0]에서 해를 구한다.
-Method는 크게 **Bracketing method**와 **Open method**로 나뉜다. Bracketing은 구간 자체를 좁혀나가며 해를 찾는 방식이라면, Open method는 특정 방법에 의해 해에 근사적인 점을 얻어가면서 해를 찾는다.

## 2 .Experimental process

Root finding process

❶ **Incremental search(zbrak.c)**로 해가 존재하는 구간들을 찾는다.
❷ **Method**를 이용해 구간내의 근사 해를 찾는다. 이때 **(Error) < 10^-6**인 경우에 근사해로 인정한다.

### Step1) Incremental search

<code>

```
n = 20;
b = 10;
zbrak(bessj0 ,1.0, 10.0, n, xb1, xb2, &b);
if (b == 0) {//n개로 구간을 나눠 살펴봤을 때, 해를 찾을 수 없는 경우
                printf("구간 개수 다시 정하시오.\n");
                return 0;
}
```

**Bessel function J0에 대해, 10개(=b)의 해를 찾는다고 가정하고, 구간 [1.0,10.0]을 20개(=n)로 나눈다. 해가 존재하는 구간의 시작점, 끝점을 xb1,xb2에 저장한다.**

Step2) Method를 이용해 Root finding

<code>

```
for (int i = 1; i <= b; i++) {
        a = 0;
        proper_routine = (rtbis, rtflsp, rtmull, rtsec 중 하나 선택); or
        another =(rtnewt, rtsafe중 하나 선택);
        root = proper_routine(bessj0, xb1[i], xb2[i], xacc, &a); or
        root = another(user_func, xb1[1], xb2[1], xacc, &a);
        printf("%d번째root:%e    //  iter_num:%d\n", i, root, a);
}
```

**proper_routine을 b번 사용해서 b개의 roots와 iteration 횟수를 얻는다.**

*<Bracketing method>*

**❶ Bisection method(rtbis.c)** -해당 구간을 반으로 줄여가면서 해를 찾는 방법.

```
for (j=1;j<=JMAX;j++) {
                (*iter_num)++;
                fmid=(*func)(xmid=rtb+(dx *= 0.5));
                if (fmid <= 0.0) rtb=xmid;
                if (fabs(dx) < xacc || fmid == 0.0) return rtb;
}
```

**❷ False-position method(rtflsp.c)** -해당 구간의 양 끝점을 이어 f(x) = 0을 만족하는 x를 찾아낸 뒤, 해가 존재하는 구간을 좁혀가면서 해를 찾는 방법.

```
for (j=1;j<=MAXIT;j++) {
                (*iter_num)++;
                rtf=xl+dx*fl/(fl-fh);// one of root sequence
                f=(*func)(rtf);
                if (f < 0.0) {
                        del=xl-rtf;
                        xl=rtf;
                        fl=f;
                } else {
                        del=xh-rtf;
                        xh=rtf;
                        fh=f;
                }
                dx=xh-xl;
                if (fabs(del) < xacc || f == 0.0) return rtf;
}
```

*<Open method>*

❶ **Newton-Raphson method(rtnewt.c)** -특정 점의 순간기울기를 이용해 해를 근사해나가는 방법.

```
for (j=1;j<=JMAX;j++) {
        (*iter_num)++;
        (*funcd)(rtn,&f,&df);
        dx=f/df;
        rtn -= dx;
        if ((x1 - rtn)*(rtn - x2) < 0.0)
                nrerror("Jumped out of brackets in rtnewt");
        if (fabs(dx) < xacc) return rtn;
}
```

❷ **Fail-safe method(rtsafe.c)** -Newton-Raphson method를 이용해 해를 구하다가 범위가 벗어나거나 update 폭이 일정이상 작아지면 Bisection method를 통해 구간을 다시 좁히면서 해를 찾는 방법.

```
for (j=1;j<=MAXIT;j++) {
        (*iter_num)++;
        if ((((rts-xh)*df-f)*((rts-xl)*df-f) > 0.0)
                || (fabs(2.0*f) > fabs(dxold*df))) {
                dxold=dx;
                dx=0.5*(xh-xl);
                rts=xl+dx;
                if (xl == rts) return rts;
        } else {
                dxold=dx;
                dx=f/df;
                temp=rts;
                rts -= dx;
                if (temp == rts) return rts;
        }
        if (fabs(dx) < xacc) return rts;
        (*funcd)(rts,&f,&df);
        if (f < 0.0)
                xl=rts;
        else
                xh=rts;
}
```

❸ **Secant-method(rtsec.c)** -두 점을 이어 f(x) = 0을 만족하는 x를 찾는다. 계속 x를 갱신해나가며 해를 구하는 방법.

```
for (j=1;j<=MAXIT;j++) {
                (*iter_num)++;
                dx=(xl-rts)*f/(f-fl);
                xl=rts;
                fl=f;
                rts += dx;
                f=(*func)(rts);
                if (fabs(dx) < xacc || f == 0.0) return rts;
}
```

❹ **Muller method(rtmull.c)** -interpolatin방식으로, 세 점을 이용해 2차 다항함수를 하나 구하고 f(x)=0을 만족하는 x를 찾는다. 계속 x를 갱신해나가며 해를 구하는 방법. 복소수근을 찾는데 사용되지만 Incremental search를 통해 실근이 존재하는 범위 내에서 해를 찾기 때문에 실근을 구함. (**rtmull.c 참조**)

## 3.Result

*-각 method를 이용해 얻은 roots와 iteration횟수를 출력*

1)Bisection

```
1번째root:2.404826e+00   //  iter_num:20
2번째root:5.520078e+00   //  iter_num:20
3번째root:8.653728e+00   //  iter_num:20
```

2)False-position method

```
1번째root:2.404826e+00   //  iter_num:6
2번째root:5.520078e+00   //  iter_num:4
3번째root:8.653728e+00   //  iter_num:4
```

3)Newton-Raphson method - 2번째 해를 구할 때 error발생

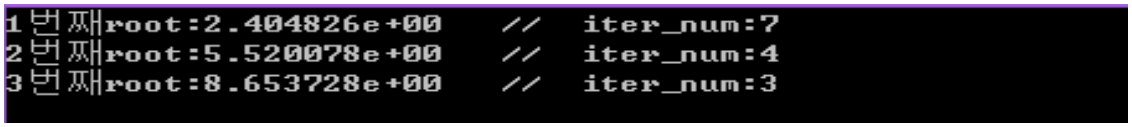
```
1번째 root:2.404825e+00   //   iter_num:3
Jumped out of brackets in rtnewt  /2번째 root:0.000000e+00   //   iter_num:1
3번째 root:8.653728e+00   //   iter_num:2
```

4)Fail-safe method

```
1번째 root:2.404825e+00   //   iter_num:3
2번째 root:5.520078e+00   //   iter_num:6
3번째 root:8.653728e+00   //   iter_num:2
```

5)Secant method


6)Muller method




## 4.Discussion

| method | Average of iterations |
|---|---|
| Bisection | 20 |
| False-position | 4.7 |
| Newton-Raphson | 2.5(Error 제외) |
| Fail-safe | 3.7 |
| Secant | 4.3 |
| Muller | 4.7 |

-Open method가 Bracketing method보다 전반적으로 converge속도가 빠름을 확인할 수 있다. 그 중에서 Bisection method는 구간을 안정적으로 줄여나가는 만큼 iteration횟수가 상대적으로 크다.

-Newton-Raphson은 converge가 굉장히 빠르지만 error가 발생할 수 있어 불안정하다**(이번 Process에서 2번째 해를 구할 때 error발생)**. 이 두 method를 섞어 쓰는 Fail-safe는 빠른 converge에 안정성을 보여준다.