**NA-HW #03**

2015004693\_양상헌

**Goal:**

* **Bisection Method**
* **Linear Interpolation Method(False Position)**
* **Secant Method**
* **Newton Raphson Method**
* **Newton Raphson with Bracketing.**
* **Muller Methods**

위 6가지 Root finding Methods를 이용하여 Bessel Function의 해를 구하고, 해를 구하는 과정에서 Iteration이 각각 얼마나 많이 일어나는지(Convergence Speed를) 확인해본다. 또한 각각의 방법의 장단점을 알아본다.

**Background Knowledge:**

Root Finding Methods는 크게 Bracketing Methods와 Open Methods 스타일로 구분되고 Bisection Method와 Linear Interpolation Method 는 Bracketing Methods 에 해당하고, Secant Methods, Newton Raphson Methods, Muller Methods는 Open Methods에 해당한다.

Bisection Method는 단순히 근이 존재하는 구간을 계속해서 절반으로 나눠가면서 근을 찾는 방법이고, Linear Interpolation Method는 구간의 시작과 끝점의 함수 값을 지나는 직선이 0이 되는 곳을 새로운 구간의 시작점 혹은 끝점으로 잡고 구간을 점점 줄여 나가며 해를 구하는 방법이다.

Newton Raphson Method는 임의의 지점에서의 도함수(미분값)를 이용하여 다음 지점을 지정하여 이 과정을 반복하며 해를 구하는 방식이고, Secant Method는 Newton Raphson Method와 비슷한 원리이지만 도함수를 이용하지 않고, 임의의 두 지점의 평균 기울기를 이용하여 다음 지점을 지정하고 이를 반복해 해를 구하는 방식이다. Muller Method는 Secant Method에서 1차원 직선함수를 2차원 곡선 함수로 바꿔 다음 지점을 지정하는 방식이다. 임의의 세 지점을 지나는 곡선의 방정식을 구하여 다음 지점을 지정하고 이를 반복해 해를 구하는 방식이다.

실제 Iteration 횟수를 비교하여 Convergence Speed를 비교해보면 Bracketing Method 보다 Open Method가 더 빠르게 해를 찾는 편이고, 정확한 해를 구하는(Complete)면 에서는 Bracketing Method가 Open Method보다 더 좋은 효과를 갖는다. 요약하면 Bracketing Method는 low speed & high completeness 이고, Open Method는 high speed & low completeness 인 셈이다.

**Process:**

* **Example code 사용**

Muller Method를 제외하고는 모든 Method는 이미 NR에 구현되어 있는 함수 소스코드와 Example 소스코드를 약간만 편집하여 사용했다. 함수 소스코드 내부에서 해를 찾아 반환하는 부분에서 Iteration의 횟수를 출력해주는 부분만 추가 하여 사용하였다. 또한 rtsafe.c를 이용하여 비선형 방정식 4문제를 푸는 문제의 경우 xtrsafe.c 파일에 각각의 방정식 식과, 그 것들의 미분식을 함수로 먼저 선언해 주고 main함수안에서 실행하는 함수 인자의 이름을 바꿔가며 결과를 출력해보는 방식으로 실행해보았다**.**

*편집된<xrtsafe.c>*

|  |
| --- |
| */\* Driver for routine rtsafe \*/*  #include <math.h>  #include <stdio.h>  #define NRANSI  #include "nr.h"  #include "nrutil.h"  #define N 100  #define NBMAX 20  #define X1 1.0  #define X2 10.0  **float** func01(**float** x){  **float** ans = (10.0\*expf(-1\*x)\*sin(2.0\*M\_PI\*x) - 2.0);  **return** ans;  }  **float** dfunc01(**float** x){  **float** ans = (10\*-expf(-1\*x)\*(sin(2\*M\_PI\*x)-2\*M\_PI\*cos(2\*M\_PI\*x)));  **return** ans;  }  **static** **float** fx01(**float** x){  **return** func01(x);  }  **static** **void** function01(**float** x, **float**\* fn, **float**\* df){  \*fn = func01(x);  \*df = dfunc01(x);  }  **float** func02(**float** x){  **float** ans = ((x - expf(-1\*x))\*(x - expf(-1\*x)));  *//float ans = x\*x - 2\*x\*exp(-1\*x) + exp(-2\*x);*  **return** ans;  }  **float** dfunc02(**float** x){  **float** ans = (2\*(x - expf(-1\*x))\*(expf(-1\*x) + 1));  **return** ans;  }  **static** **float** fx02(**float** x){  **return** func02(x);  }  **static** **void** function02(**float** x, **float**\* fn, **float**\* df){  \*fn = func02(x);  \*df = dfunc02(x);  }  **float** func03(**float** x){  **float** ans = (cos(x+sqrt(2))+x\*(x/2 + sqrt(2)));  **return** ans;  }  **float** dfunc03(**float** x){  **float** ans = (-sin(x+sqrt(2))+x+sqrt(2));  **return** ans;  }  **static** **float** fx03(**float** x){  **return** func03(x);  }  **static** **void** function03(**float** x, **float**\* fn, **float**\* df){  \*fn = func03(x);  \*df = dfunc03(x);  }  **float** funcMyOwn(**float** x){  **float** ans = (expf(2\*x)+x-5);  **return** ans;  }  **float** dfuncMyOwn(**float** x){  **float** ans = (2\*expf(2\*x)+1);  **return** ans;  }  **static** **float** fxMyOwn(**float** x){  **return** funcMyOwn(x);  }  **static** **void** functionMyOwn(**float** x, **float**\* fn, **float**\* df){  \*fn = funcMyOwn(x);  \*df = dfuncMyOwn(x);  }  **static** **float** fx(**float** x)  {  **return** bessj0(x);  }  **static** **void** funcd(**float** x,**float** \*fn, **float** \*df)  {  \*fn=bessj0(x);  \*df = -bessj1(x);  }  **int** main(**void**)  {  **int** i,nb=NBMAX;  **float** xacc,root,\*xb1,\*xb2;  **float** b0,b1;  b0 = 0.0;  b1 = 2.0;      xb1=vector(1,NBMAX);  xb2=vector(1,NBMAX);  zbrak(fx,X1,X2,N,xb1,xb2,&nb);  *//zbrak(fx,b0,b1,N,xb1,xb2,&nb);*    printf("by Newton w/ Bracketing Method\n");  printf("\nRoots of bessj0 on [1, 10]:\n\n");  *//printf("\nRoots of 10e^(-x)sin(2PIx)-2 on [0.1, 1]:\n\n");*  *//printf("\nRoots of x^2-2xe^(-x)+e^(-2x) on [0, 1]:\n\n");*  *//printf("\nRoots of cos(x+sqrt(2))+x(x/2+sqrt(2)) on [-2, -1]:\n\n");*  *//printf("\nRoots of e^(2x)+x-5 on [0, 2]:\n");*  printf("%21s %15s\n","x","f(x)\n");  **for** (i=1;i<=nb;i++) {  xacc=(1.0e-6)\*(xb1[i]+xb2[i])/2.0;  root=rtsafe(funcd,xb1[i],xb2[i],xacc);  printf("root %3d %14.6f %14.6f\n\n",i,root,fx(root));  }  free\_vector(xb2,1,NBMAX);  free\_vector(xb1,1,NBMAX);  **return** 0;  }  #undef NRANSI |

**-hw03Muller.c 구현**

Muller Method의 구현은 다음의 소스코드와 같다. 처음 인자로 들어오는 세개의 점 p0, p1, p2를 받아 이 세점을 지나는 2차원 곡선의 방정식의 coefficient인 a, b, c를 구하고 이 곡선의 방정식을 이용하여 f(x)값이 0이되는 점 p3를 구하고 , p3가 해이거나 p3-p2가 오차범위 내인 경우 이를 반환하고, p3가 해가 아니고, p3-p2가 오차범위 를 넘는 경우 p1,p2,p3가 p0,p1,p2가 되고 위 과정을 반복하게 된다. Iteration 과정에서 coefficient a,b,c를 계산하는 과정은 수업자료 pdf에 나온 식을 그대로 사용 하였다.

*구현된<hw03Muller.c>*

|  |
| --- |
| #include "nr.h"  #include "nrutil.h"  #include <math.h>  #include <stdio.h>  #include <stdlib.h>  #define MAXIT 100  #define N 100  #define NBMAX 20  #define X1 1.0  #define X2 10.0  **static** **float** fx(**float** x)  {  **return** bessj0(x);  }  **static** **void** funcd(**float** x,**float** \*fn, **float** \*df)  {  \*fn=bessj0(x);  \*df = -bessj1(x);  }  **float** muller(**float** (\*func)(**float**), **float** x0, **float** x1, **float** x2, **float** bound0, **float** bound1, **float** xacc){  **void** nrerror(**char** error\_text[]);  **int** j;  **int** sgn;  **float** p0, p1, p2, a, b, c, rts, f;    p0 = x0;  p1 = x1;  p2 = x2;    **for**(j=1; j<=MAXIT; j++){  c = (\*func)(p2);  b = ((p0-p2)\*(p0-p2)\*((\*func)(p1)-(\*func)(p2))-(p1-p2)\*(p1-p2)\*((\*func)(p0)-(\*func)(p2)))/((p0-p2)\*(p1-p2)\*(p0-p1));  a = ((p1-p2)\*((\*func)(p0)-(\*func)(p2))-(p0-p2)\*((\*func)(p1)-(\*func)(p2)))/((p0-p2)\*(p1-p2)\*(p0-p1));    **if** (b < 0.0)  sgn = -1;  **else**  sgn = 1;  rts = p2 - ((2\*c)/(b+sgn\*sqrtf(b\*b-4\*a\*c)));  f = (\*func)(rts);  **if** (rts < bound0 || rts > bound1)  nrerror("Jumped out of brackets in muller");  **if**(fabs(rts-p2) < xacc || f == 0.0){  printf("Total Iteration : %d\n",j);  **return** rts;  }  **else**{  p0 = p1;  p1 = p2;  p2 = rts;  }  }  nrerror("Maximum number of iterations exceeded in muller");  **return** 0.0;  }  **int** main(**void**)  {  **int** i,nb=NBMAX;  **float** xacc,root,\*xb1,\*xb2;  **float** tmp;    xb1=vector(1,NBMAX);  xb2=vector(1,NBMAX);  zbrak(fx,X1,X2,N,xb1,xb2,&nb);  printf("by Muller Method\n");  printf("\nRoots of bessj0: on [1, 10]\n");  printf("%21s %15s\n","x","f(x)\n");  **for** (i=1;i<=nb;i++) {  xacc=(1.0e-6)\*(xb1[i]+xb2[i])/2.0;  tmp = xb2[i]-xb1[i];  root=muller(fx,xb1[i],xb1[i]+tmp/10.0, xb1[i]+tmp/2.0, xb1[i], xb2[i], xacc);  printf("root %3d %14.6f %14.6f\n\n",i,root,fx(root));  }  free\_vector(xb2,1,NBMAX);  free\_vector(xb1,1,NBMAX);  **return** 0;  } |

**Results & Discussion:**

|  |
| --- |
| **-01. finding Roots of Bessel Function by Bisection Methods** |
|  |
| Bisection method를 사용했을 때 iteration횟수는 각각 16회 14회 14회 로 평균 15회정도 iteration이 일어난 후 근을 찾을 수 있다는 것을 알 수 있다. |

|  |
| --- |
| **-02. finding Roots of Bessel Function by Linear interpolation Methods** |
|  |
| Linear Interpolation Method를 사용 했을 때의 Iteration횟수는 각각 4회 3회 3회 이다. 위의 Bisection Method를 사용했을 때보다 월등히 더 적은 Iteration 횟수를 보인다. |

|  |
| --- |
| **-03. finding Roots of Bessel Function by Newton Raphson Methods** |
|  |
| Newton Raphson Method를 사용했을땐 평균적으로 3회 정도의 Iteration 횟수를 보인다. |

|  |
| --- |
| **-04. finding Roots of Bessel Function by Secant Methods** |
|  |
| Secant Method를 사용했을땐 Newton Raphson Method와 거의 근사한 Iteration 횟수가 나오게된다. |

|  |
| --- |
| **-05. finding Roots of Bessel Function by Newton Raphson w/ Bracketing Methods** |
|  |
| Bessel 함수로 실행 했을 땐 일반 Newton과 Newton w/ Bracketing 의 Iteration횟수에는 차이가 없다는 결과가 나온다. |

|  |  |
| --- | --- |
| **-05-1. finding Roots of** 10 e^(-x) sin(2p x)-2= 0, on [0.1,1] **by Newton Raphson w/ Bracketing Methods** | |
|  | |
| 실제 그래프에서 보듯이 위 구간에는 실근이 하나 존재하고 이를 빠르게 잘 찾았다고 볼 수 있다. |  |

|  |  |
| --- | --- |
| **-05-2. finding Roots of** x2 - 2xe-x + e-2x = 0, on [0,1] **by Newton Raphson w/ Bracketing Methods** | |
|  | |
| 그래프 상으로는 실근이 하나 존재하지만, 이 근이 중근이기 때문에 Newton Raphson Method에서는 찾을 수 없는 것 같다. Newton Raphson Method는 도함수를 이용하기 때문에 중근은 찾을 수 없다. |  |

|  |  |
| --- | --- |
| **-05-3. finding Roots** cos(x+sqrt(2))+x(x/2+sqrt(2)) = 0, on [-2,-1] **by Newton Raphson w/ Bracketing Methods** | |
|  | |
| 이 함수 역시 위의 함수와 같이 그래프상으론 구간 안에서 실근이 하나 존재하지만 이 역시 중근이기 때문에 Newton Raphson으로는 해가 찾아지지 않는다. |  |

|  |  |
| --- | --- |
| **-05-4. finding Roots of e^(2x)+x-5 = 0, on [0,2] by Newton Raphson w/ Bracketing Methods** | |
|  | |
| 그래프상으로 구간 안에서 실근이 하나 존재하고 이것이 중근이 아니기 때문에 근을 잘 찾을 수 있다. |  |

|  |
| --- |
| **-06-0. finding Roots of Bessel Function by Muller Methods** |
|  |

|  |
| --- |
| **-06-1. finding Roots of Bessel Function by Muller Methods** |
|  |

|  |
| --- |
| **-06-2. finding Roots of Bessel Function by Muller Methods** |
|  |
| 위의 Muller Method를 사용한 방법들 에서는 같은 Bessel 함수의 근을 구할 때 초기지점으로 주어지는 3개의 점을 각각 다 다르게 입력하여 Iteration 횟수의 차이를 보려고 하였다. 실제로 3개의 지점의 초기값에 따라 Iteration 횟수의 차이가 크진 않았지만 차이가 있긴 있었고, 이로 인해 초기값 설정이 Iteration횟수의 매우 중요한 요인이라고 결론 내렸다. |

|  |
| --- |
| **Overall Discussion about Convergence speed** |
| * 위 수업자료에 나온 사진처럼 Convergence Speed는 Newton-Raphson > Secant > Linear Interpolation > Bisection 순서와 비슷한 결과를 확인할 수 있었다. * Bessel function을 기준으로 Bisection 과 Linear Interpolation 만 비교해도 매우 큰 convergence speed 차이를 보였고, Secant 와 Newton-Raphson 은 Linear Interpolation과 큰 차이 없이 아주 조금 더 빠른 정도의 속도를 확인할 수 있었다. * Muller method의 경우 처음 세 점의 초기값을 어떻게 설정해주는지에 따라서 Convergence Speed가 많이 달라질 수 있다. 예시로 했던 3경우 모두 Bessel function의 해를 구하는 과정에서 secant나 Newton-Raphson 만큼 빠른 convergence Speed를 보였고 평균적으로 조금 더 빠르다는 결론을 확인 하였다. * 또한 root finding Method를 선택함에 있어서 Speed와 Completeness 사이에 TradeOff 관계가 성립한다는 것을 확인할 수 있었다. Bisection 과 Linear Interpolation 은 느리지만, 중근도 찾아낼수 있지만, 이와 반대로 Newton-Raphson 같은 경우 중근은 찾아내지 못하기 때문에 completeness와 speed 사이에는 trade off 관계가 있다고 결론 지을 수 있다. |