**고급소프트웨어실습1 [CSE4152-04]**

5주차 숙제   
: Random Sequence Generation on a Specific Distribution

**학번 : 20171640**

**이름 : 박수진**

1. Secant
   1. 초기값 결정 방법
      1. Code

|  |
| --- |
| double L = 0.0, R = 1.0;  double x0 , x1 = (L + R) / 2;  x0 = x1;  if (interpolationF(x1, U) \* interpolationF(L, U) < 0) {  R = x0;  }  else {  L = x0;  }  x1 = (L + R) / 2;  x0 = floor(x1 \* 10) / 10;  x1 = ceil(x1 \* 10) / 10; |

* + 1. Description

초기값의 범위를 줄이기 위해 Bisection method를 한번 실행한다. 그 결과 값은 한 개이므로 서로 같지 않은 두 개의 초기값을 생성하기 위해 소수점 둘째 자리에서 버림한 값을 x0, 올림한 값을 x1으로 잡았다.

* 1. 결과  
       
       
     PDF 그래프와 Secant method의 Histogram 그래프를 비교하면 개형이 거의 일치하는 것을 볼 수 있다. 또한 발산하는 값이 없는 것으로 보아 초기값을 적절히 잘 잡았다고 할 수 있다.

1. Improved
   1. 방법
      1. Code

|  |
| --- |
| double genRandN\_Improved(double U, int si) {  double L = 0.0, R = 1.0;  double x1 = (L + R) / 2, x0 = 0; // these variables are for initial build without coding  // you may change the names or delete them if needed  // \*\*\* 이 함수를 작성하세요  // 숙제입니다.  // Bisection과 Newton-Raphson을 적절히 조합하여 어떤 경우에도 발산하지  // 않는 함수를 만들어 봅시다.  // 필요하다면, 이 함수에서 호출하는 다른 함수를 작성하여 호출해도 좋습니다.  int iter = 0;  double nrvar;  if (showSteps == OUTPUT\_STEPS) {  printf(" n(%6d) xn1 |f(xn1)| U = %.4lf\n", si, U);  }  for (iter = 0; iter <= maxIter; iter++) {  if (showSteps == OUTPUT\_STEPS) {  printf("%2d %.12e %.16e\n", iter, x1, fabs(interpolationF(x1, U)));  }  if (fabs(interpolationF(x1, U)) < delta)  break;  if (iter != 0 && fabs(x1 - x0) < epsilon)  break;  x0 = x1;  nrvar = x0 - (interpolationF(x0, U) / interpolationFD(x0));  if ((L < nrvar) && (nrvar < R)) {  x1 = nrvar;  if (interpolationF(x1, U) \* interpolationF(L, U) < 0)  R = x1;  else  L = x1;  }  else {  if (interpolationF(x1, U) \* interpolationF(L, U) < 0)  R = x1;  else  L = x1;  x1 = (L + R) / 2;  }  }  return x1;  } |

* + 1. Description  
       Newton-Raphson method의 문제점은 미분 값이 매우 작을 경우 x 값이 범위를 벗어나 발산할 수도 있다는 것이다. 따라서 이를 보정하기 위해서 이러한 경우에 Newton-Raphson method 대신 Bisection method를 사용하였다.(범위를 벗어나지 않으면 계속 Newton-Raphson method 사용한다) 초기 Left 값과 Right 값은 각각 0과 1로 잡고 Newton-Raphson method 혹은 Bisection method를 사용하면서 계속 Left값과 Right값을 줄인다. – L, R값 보정은 Bisection에서 사용하는 방법을 두 방법 모두에 적용하였다. x값이 L값과 R값 사이의 범위를 벗어나는 경우에는 Bisection method를 사용하고 그게 아니면 Newton-Raphson method를 사용한다. 종료조건은 기존 Newton-Raphson method와 동일하다.
  1. 증명  
     Newton-Raphson은 세가지 방법 중 가장 빠른 방법이기 때문에 최대한 Newton-Raphson 방법을 사용하고 발산하는 상황에서만 Bisection을 사용한다. Newton-Raphson은 초기값에서 시작하여 근을 향해서 점점 가까워진다. Newton-Raphson을 사용하면서 Bisection 방법으로 L과 R변수를 조정해주면 근에 대한 범위를 점점 좁힐 수 있다. 만약 Newton-Raphson을 사용했을 때 L과 R사이의 범위를 벗어나면 Bisection으로((L+R)/2) 다시 값을 설정해준다. 그러면 발산하지 않으면서 근과 가까운 값에서 다시 Newton-Raphson 방법을 적용할 수 있다.  
     즉, Improved를 사용하면 1) 범위를 벗어나는 경우 다시 Bisection으로 값을 잡아주므로 발산하지 않고 2) 가장 빠른 Newton-Raphson을 사용하면서 보조로 Bisection을 사용하므로 Bisection보다 확실히 빠르다.  
       
       
       
     PDF 그래프와 Improved method(Newton-Raphson + Bisection)의 Histogram 그래프를 비교하면 개형이 거의 일치하는 것을 볼 수 있다. 또한 발산하는 값이 없으며 속도가 Newton-Raphson과 거의 비슷하게 빠르다.   
       
     \* Newton-Raphson(2526.00ms) < Improved(2724.00ms) < Bisection(6607.00ms)  
     \* 위 실행 시간은 lab05\_01\_IN.txt 파일을 사용하였을 때 기준이다.

1. Bisection / Newton-Raphson / Secant / Improved 비교
   1. PDF  
      - 입력파일 : lab05\_04\_IN.txt  
        
      - 입력파일 : lab05\_03\_IN.txt
   2. Bisection  
      - 입력파일 : lab05\_04\_IN.txt  
        
        
        
      - 입력파일 : lab05\_03\_IN.txt  
        
      
   3. Newton-Raphson  
      - 입력파일 : lab05\_04\_IN.txt  
        
        
        
      - 입력파일 : lab05\_03\_IN.txt  
        
      
   4. Secant  
      - 입력파일 : lab05\_04\_IN.txt  
        
        
        
      - 입력파일 : lab05\_03\_IN.txt  
        
      
   5. Improved  
      - 입력파일 : lab05\_04\_IN.txt  
        
        
        
      - 입력파일 : lab05\_03\_IN.txt  
        
      
   6. Conclusion  
      - lab05\_04\_IN.txt 파일의 경우, pdf 그래프가 값이 0에 가깝게 작아지는 구간이 많다. (cdf 그래프에서 기울기가 0에 가까워 지는 지점이 많다) 그래서 Newton-Raphson의 경우 계속 발산하여 시간도 오래 걸리고 제대로 된 값도 구하지 못하였다. (PDF그래프 개형과 완전 상이) 그러나 다른 방법들은 PDF 그래프와 비슷한 개형을 보이며 거의 수렴하여 알맞은 값을 나타낸다. 속도는 Newton-Raphson 방법이 주가 되면서 발산하지 않는 Improved가 제일 빠르고 값이 발산하는 Newton-Raphson 방법이 제일 느리다.  
        
      Improved(3289ms) < Secant(4135ms) < Bisection(8324ms) < Newton-Raphson(13436ms)  
        
      - lab05\_03\_IN.txt 파일의 경우, pdf 그래프가 0에 가깝게 아주 작아지는 점이 양 끝점 밖에 없다. 그래서 모든 방법이 거의 다 수렴한다. PDF 그래프의 개형과 다른 모든 방법의 개형이 거의 일치한다. 이렇게 거의 다 수렴할 경우 Bisection이 가장 느리고 나머지 세 방법은 비슷한 속도를 나타낸다. 실제 실험결과도 이를 나타낸다. Bisection 방법은 절반씩 줄여가며 값을 찾는 방법이기 때문에 항상 값을 찾을 수 있고 모든 함수에서 비슷한 실행시간을 보이지만 Newton-Raphson과 Secant는 초기값과 함수 개형에 따라 실행시간이 달라진다. 보통 Newton-Raphson이 Secant보다 빠르다고 알려져 있지만 두 method code를 작성할 때 초기값을 다르게 설정했기 때문에 이번 실험에서는 Secant가 더 빠른 결과가 나타났다.  
        
      Improved(3145ms) < Secant(3251ms) < Newton-Raphson(3396ms) < Bisection(7498ms)