고급소프트웨어 과제 4주차

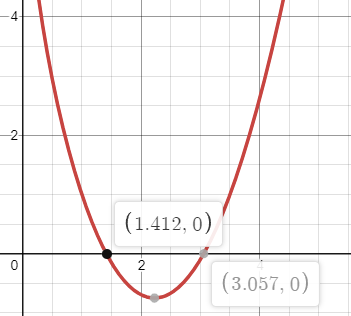
학번/ 이름 : 20161565 권기윤

* 실습문제 1, 2, 4

1-1) 방정식 f1(x) = x^2−4x+4−ln x = 0의 근을 Newton-Raphson 방법 (프로그램 1-1)과 Secant 방법 (프로그램 1-2)을 사용하여 구하는 프로그램을 작성하라.

|  |
| --- |
| program1\_1.cpp |
| #include "my\_solver.h"  extern double (\*\_f)(double);  extern double (\*\_fp)(double);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Newton-Rapson Method  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void program1\_1(FILE\* fp) {  double x0, x1;  int n = 0;  //int count = 0;  //int N\_max = Nmax;  if (fp == NULL)  return;  printf("input x\_0 : ");  scanf("%lf", &x0);  fprintf(fp,"n x |f(x)|\n");  while (n < Nmax) {  double \_fx = \_f(x0);  fprintf(fp,"%-2d %25.15e %25.15e\n",n, x0, fabs(\_fx));  if (fabs(\_fx) < DELTA) break;  x1 = x0 - (\_fx / \_fp(x0));  if (fabs(x1 - x0) < EPSILON) break;  x0 = x1;  n++;  }  printf("%25.15e\n", x0);  } |
| program1-2.cpp |
| #include "my\_solver.h"  extern double (\*\_f)(double);  extern double (\*\_fp)(double);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Newton-Rapson Method  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void program1\_1(FILE\* fp) {  double x0, x1;  int n = 0;  //int count = 0;  //int N\_max = Nmax;  if (fp == NULL)  return;  printf("input x\_0 : ");  scanf("%lf", &x0);  fprintf(fp,"n x |f(x)|\n");  while (n < Nmax) {  double \_fx = \_f(x0);  fprintf(fp,"%-2d %25.15e %25.15e\n",n, x0, fabs(\_fx));  if (fabs(\_fx) < DELTA) break;  x1 = x0 - (\_fx / \_fp(x0));  if (fabs(x1 - x0) < EPSILON) break;  x0 = x1;  n++;  }  printf("%25.15e\n", x0);  //scanf 입력값 받고 뉴튼 뭐시기 호출 \_f \_f  } |

1-2) 방법에 대한 초기값을 각각 x0 = 3.0과 x0 = 2.0, x1 = 4.0으로 설정하여 자신이 작성한 프로그램을 수행시켜 자신의 프로그램이 원하는 근을 정확히 찾고 있는지 분석하라. 과연 근이 맞는지 어떻게 확인할 수 있을까? 논리적으로 타당한 방법으로 분석한 내용을 보고서에 기술하라.



<의 그래프>

텍스트이(가) 표시된 사진

자동 생성된 설명

<프로그램에서 산출한 결과>

각 프로그램은 위와 같은 방법으로 계산과정에서 결과값으로 3.057라는 해를 도출한다. 이러한 함수의 출력을 관찰하면 값이 0으로 수렴하고 있다는 것을 볼 수 있다. 이는 위의 의 그래프의 f(x) = 0 의 값과 같다.

근 중 하나인 1.412는 실험과정에서 할당한 x의 값이 3.057에 가까웠기 때문에 해당 근으로 수렴하여 성공적으로 근을 구했음을 알 수 있다.

1-3) 위에서 산출한 결과를 볼 때, 각 방법의 근의 수렴 속도가 과연 앞에서 설명한 속도로 수렴하는지 비교 분석한 후 그 결과를 보고서에 기술하라.

일반적으로 Secant 방법은 Newton-Rapson 방법보다 수렴 속도가 느리다. 이러한 특징은 위의 결과를 통해 확인할 수 있다.

반복문 수행을 통해 수열 이 근 에 수렴한다고 할 때, 각 에는 만큼의 절대오차가 존재한다. Newton-Rapson 방법의 경우, 아래와 같은 2차 수렴을 한다. (근이 simple root인 경우)

이때 과 같은 모습으로 오차가 감소한다.

반면 Secant 방법은 다음과 같이 차수가 1.62로 수렴하는데, 아래와 같다.

따라서 와 같은 모습으로 감소한다.

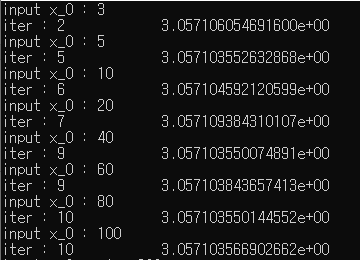
즉, 동일한 조건 상에서 Secant 방법은 Newton-Rapson 방법보다 수렴 속도가 느림을 확인할 수 있다.

텍스트, 배터리, 점수판이(가) 표시된 사진

자동 생성된 설명

이러한 의 수렴과정은 위의 출력을 통해 확인 가능하다.

1-4) 위에서 제시한 초기값 외에 임의의 초기값들을 사용하여 자신이 작성한 프로그램을 수행한 후, 이 두 방법이 항상 임의의 초기값에 대해 빠르게 수렴하는지 보고서에 기술하라.

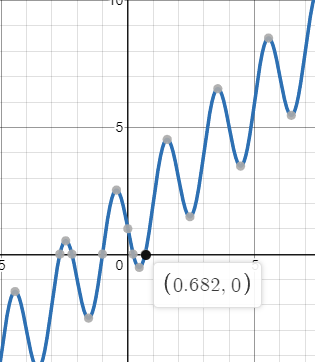


텍스트이(가) 표시된 사진

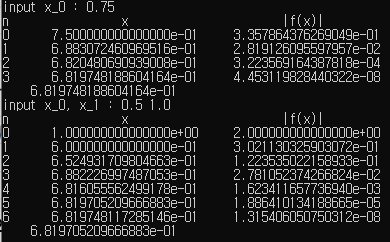
자동 생성된 설명

Newton-Rapson 방법과 Secant 방법 모두 근과 차이가 큰 초기값일수록 더 많은 횟수 함수를 반복하였다. 두 방법 모두 부적절한 초기값이 입력으로 주어졌다면, Nmax를 초과하는 횟수를 반복하여 정확한 근을 찾지 못하고 종료할 가능성이 있다. 따라서 함수의 성능에 있어 적절한 초기값의 입력이 중요하다.

1-5) 위에서 작성한 프로그램을 사용하여 함수 에 대해 초기값을 와 으로 설정하여 위 문제를 반복하라.



위 그림은 의 그래프이다.



각 프로그램은 위와 같은 계산과정을 통해 결과 값으로 0.68197…을 구한다. 이러한 함수의 출력을 관찰하면, 함수가 수행될수록 의 값이 0으로 수렴함을 볼 수 있다. 이는 우리가 구하고자 하는 의 근을 정확히 찾았다고 볼 수 있다.

텍스트, 배터리, 점수판, 하얀색이(가) 표시된 사진

자동 생성된 설명

각 프로그램은 계산결과 값으로 0.6819… 을 도출한다. 함수의 출력을 관찰하면, 함수가 반복문을 수행할수록 x0의 값이 f(x0)의 값을 0으로 수렴함을 볼 수 있다. 즉, 우리가 구하고자 하는 근을 정확히 찾았다고 볼 수 있다. Newton-Rapson 방법은 c가 1.311035326…로 수렴하고, secant방법은 c가 2.0297087…로 수렴하는 것을 볼 수 있다.

위에서 제시한 초기값 외에 임의의 초기값들을 사용하여 자신이 작성한 프로그램을 수행한 후, 이 두 방법이 항상 임의의 초기값에 대해 빠르게 수렴하는지 보고서에 기술하라

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

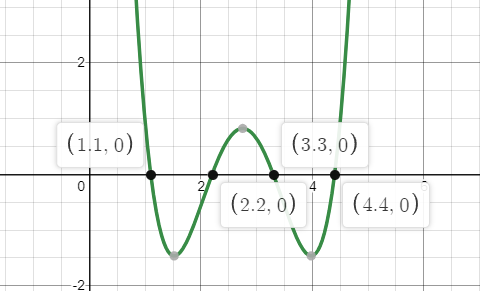
2-1) 다음과 같은 다항식에 대한 방정식을 고려하자.

지금 어떤 근 분리 방법을 사용하여, 이 방정식은 네 개의 서로 다른 실근을 가지며, 각 실근은 [1.02, 1.48], [1.95, 2.37], [2. ,3.], [3.83, 4.61] 구간에 존재한다는 사실을 밝혀냈다. 이 사실을 근거로 Newton-Raphson 방법을 사용하여 모든 실근을 구하라.

텍스트이(가) 표시된 사진

자동 생성된 설명

,초기값의 좋은 후보를 고려하여 각 구간의 중점을 잡아 실험하였다.



위의 출력 결과와 실제 그래프의 f(x) =0의 값을 비교해보면 값이 거의 정확히 일치함을 알 수 있다.

4-1) 실습문제 1-1에서 작성한 자신의 코드의 single-precision 버전, 즉 모든 계산을 float 타입의 계산을 수행하는 프로그램을 작성하라. 여러분이 작성한 코드는 실습문제 1-1과 동일한 방식으로 파일에 저장을 하는데, single-precision 버전임을 강조하기위해 해당하는 파일 이름 앞에 sp\_를 붙일 것.

|  |
| --- |
| #include "my\_solver.h"  extern float (\*\_sp\_f)(float);  extern float (\*\_sp\_fp)(float);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Newton-Rapson Method  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void sp\_program1\_1(FILE\* fp) {  float x0, x1;  int n = 0;  if (fp == NULL)  return;  printf("input x\_0 : ");  scanf("%f", &x0);  fprintf(fp, "n x |f(x)|\n");  while (n < Nmax) {  float \_sp\_fx = \_sp\_f(x0);  fprintf(fp, "%-2d %25.15e %25.15e\n", n, x0, fabs(\_sp\_fx));  if (fabs(\_sp\_fx) < DELTA) break;  x1 = x0 - (\_sp\_fx / \_sp\_fp(x0));  if (fabs(x1 - x0) < EPSILON) break;  x0 = x1;  n++;  }  printf("%25.15e\n", x0);  } |

4-2) 다음 f1(x) = ln x−1 = 0과 같이 근을 알고 있는 비선형 방정식을 고려하자(잘 알다시피 근은 e=2.718281828459045235360287471352· · ·임). 이 방정식의 근을 Newton-Raphson 방법을 사용하여 구하려 하는데, 적절한 초기값 x0에 대해 자신이 작성한 double-precision 버전과 single-precision 버전 각각을 사용하여 근을 구하여 보자. 이때 부동 소수점 연산의 정밀도가 다

른 두 방법이 구한 근의 값이 정확한 근 e와 비교하여 어떤 차이가 있는지, 자신이 알아낸 사실을 보고서에 상세히 기술하라.

텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |  |
| --- | --- | --- |
|  | 2.71828182846 | 오차 |
| double-precision | 2.71828182821 | 0.00000000025 |
| single-precision | 2.71828174591 | 0.00000008255 |

위의 도표를 통해 double-precision을 사용한 경우가 single-precision보다 더 정확한 값을 구할 수 있음을 확인 가능하다. 이러한 차이점은 각 precision 마다 표현 가능한 자리수가 다르기 때문인데, IEEE 754 format에 따른 각 precision은 아래와 같이 데이터를 저장한다.

* Float type은 지수(E)가 8bits, 가수가 23bits
* double type은 지수(E)가 11bits, 가수가 52bits

위의 특징으로 single-precision을 사용한 float를 사용한 경우, 2의 23승 까지의 가수의 표현이 가능하고, double-precision을 사용한 double은 2의 52승까지 가수의 표현이 가능하다.

수학적 풀이로 근을 구하는 프로그램을 작성할 때는, 부동소수점 오차가 더 적은 double type을 쓰는 것을 고려해야 한다.

* 과제 1,2

1-1) 자료 조사를 통하여 Bisection 방법을 정확히 이해한후, 이 방법을 사용하여 방정식의 근을 구해주는 프로그램을 작성하라 (프로그램 1-3).

|  |
| --- |
| #include "my\_solver.h"  extern double (\*\_f)(double);  extern double (\*\_fp)(double);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Bisection Method -- HOMEWORK  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void program1\_3(FILE \*fp)  {  double a0, b0, x0, x1 , temp;  double \_fa, \_fb, \_fc;  int n = 0;  printf("input [a, b] : ");  scanf("%lf %lf", &a0, &b0);  //fprintf(fp, "n x |f(x)|\n");  printf( "n x |f(x)|\n");  while (n < Nmax) {  \_fa = \_f(a0);  \_fb = \_f(b0);  temp = (a0 + b0) / 2;  x0 = a0;  \_fc = \_f(temp);  //fprintf(fp, "%-2d %25.15e %25.15e\n", n, x0, fabs(\_fa));  printf( "%-2d %25.15e %25.15e\n", n, x0, fabs(\_fa));  if (\_fc == 0) break;  if (fabs(a0 - b0) < EPSILON) break;  if (fabs(fabs(a0 - b0) / fabs(a0)) < EPSILON || fabs(\_f(a0)) < EPSILON) break;  if (\_fa \* \_fc < 0.0 && \_fc \* \_fb > 0) b0 = temp;  if (\_fa \* \_fc > 0.0 && \_fc \* \_fb < 0) a0 = temp;      n++;  }  printf("%25.15e\n", x0);  } |

1-2) 프로그램이 완성되면, 실습 시간에 사용한 세 개의 함수 에 대한 적절한 초기 구간을 사용하여 올바르게 근에 수렴하는지에 대한 분석 내용을 보고서에 기술하라.

|  |  |  |
| --- | --- | --- |
| f(x)1 | f(x)2 | |
|  | 텍스트이(가) 표시된 사진  자동 생성된 설명 | |
| f(x)3 | | |
| 텍스트이(가) 표시된 사진  자동 생성된 설명 | | 텍스트이(가) 표시된 사진  자동 생성된 설명 |
| 텍스트이(가) 표시된 사진  자동 생성된 설명 | | 텍스트이(가) 표시된 사진  자동 생성된 설명 |

Bisection 방법의 경우 전반적인 수렴과정이 다음과 같음을 확인 가능하다.

마지막으로 갈수록 오차가 생기는 것은 부동 소수점의 오차로 인한 것으로 판단된다.

1-3) Bisection 방법의 수렴 속도는 선형적인, 즉 형태를 보인다. 위

의 세 함수에 대한 방정식을 대상으로 Newtown-Raphson 방법, Secant 방법,

그리고 Bisection 방법을 적용해보고, 과연 각 방법이 이론적인 수렴속도를 보이는지를 분석하고 그 내용을 보고서에 기술하라.

반복문 수행을 통해 수열 이 근 에 수렴한다고 할 때, 각 에는 만큼의 절대오차가 존재한다. Newton-Rapson 방법의 경우, 아래와 같은 2차 수렴을 한다. (근이 simple root인 경우)

이때 과 같은 모습으로 오차가 감소한다.

반면 Secant 방법은 다음과 같이 차수가 1.62로 수렴하는데, 아래와 같다.

따라서 와 같은 모습으로 감소한다.

마지막으로 Bisection 방법의 경우 선형적인 수렴 속도를 보이는데, 때문에 실험적으로 아래와 같이 수렴해야 한다.

이제 각 함수에 대하여 이론적인 수렴 속도를 보이는지 확인하도록 하겠다.

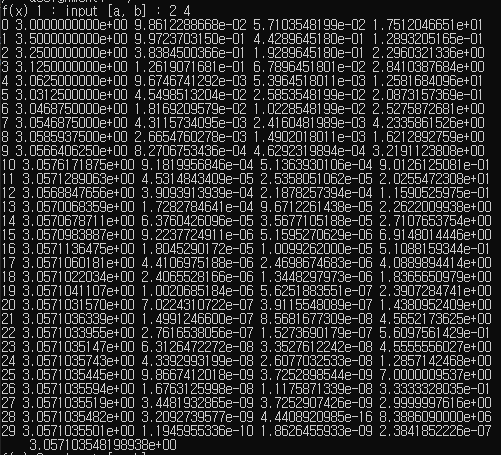
정확한 확인을 위해 각 함수의 근으로 를 설정하고, EPSILON값을 0.00000001으로 설정하여 부동소수점 정밀도를 높여 실험하였다.

텍스트, 배터리, 점수판이(가) 표시된 사진

자동 생성된 설명

Newton-Rapson방법은 c가 0.0000707…로 수렴, Secant 방법은 c가 0.7460966231..로 수렴한다.

실습 1 수식



실습 2 수식

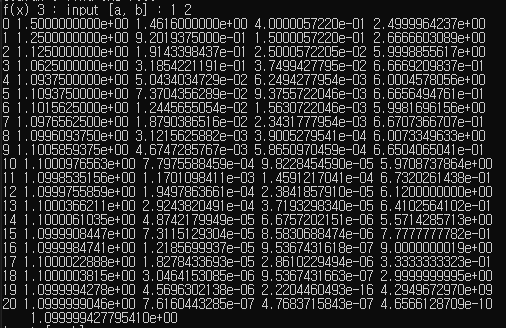
-

텍스트이(가) 표시된 사진

자동 생성된 설명

실습 3 수식

-



텍스트이(가) 표시된 사진

자동 생성된 설명

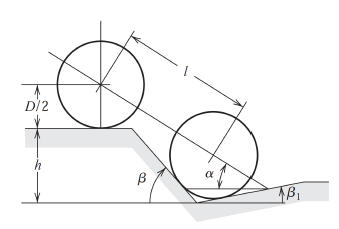


텍스트이(가) 표시된 사진

자동 생성된 설명

함수를 가지고 3가지 근에 접근하는 방식을 분석하면, Newton-Rapson 과 Secant 방식은 특정 상수 값 c에 수렴, Bisection 방식은 이론적으로 도출된 값 1/2을 얻을 것으로 기대했으나 그렇지 않음을 확인 할 수 있다. Newton-Rapson 방식이 3가지 방법론 중 가장 빠름을 보였고, Secant 방법은 1.02~1.7사이에 수렴하려고 하는 것을 확인할 수 있었다. 그렇지만 Bisection 방식은 근은 찾았으나 입실론 값의 비율이 0.5로 수렴하다 다시 변동하는 것을 볼 수 있는데 이는 부동 소수점의 오차로 인한 것으로 판단할 수 있다.

2)



이 그림에 표시된 여러 인자에 대하여 다음과 같은 방정식이 성립한다고 하는데,

f (α) = Asinα cosα +Bsin2α −Ccosα −E sinα = 0,

여기서 각 상수 값은 다음과 같이 정의된다.

A = l sinβ1,

B = l cosβ1,

C = (h+0.5D) sinβ1−0.5Dtanβ1,

E = (h+0.5D)cosβ1−0.5D.

만약 l = 89인치, h = 49인치, D = 55인치, 그리고 β1 = 11.5도이라면, 각도 α는 대략

33도 정도가 됨이 알려져 있다. 여러분이 작성한 Newton-Raphson 방법을 사용하여 정확한 각도 α 값을 구하라.

|  |
| --- |
| double \_fa(double x) {  double result;  double l = 89;  double h = 49;  double D = 55;  double beta1 = 11.5 \* (M\_PI/180);  double A = l \* sin(beta1);  double B = l \* cos(beta1);  double C = (h + 0.5 \* D) \* sin(beta1) - 0.5 \* D \* tan(beta1);  double E = (h + 0.5 \* D) \* cos(beta1) - 0.5 \* D;  x \*= (M\_PI / 180);  result = A \* sin(x) \* cos(x) + B \* sin(x) \* sin(x) - C \* cos(x) - E \* sin(x);  return result;  } |
| double \_fpa(double x) {  double result;  double l = 89;  double h = 49;  double D = 55;  double beta1 = 11.5 \* (M\_PI / 180);  double A = l \* sin(beta1); //17.743746  double B = l \* cos(beta1); //87.213298  double C = (h + 0.5 \* D) \* sin(beta1) - 0.5 \* D \* tan(beta1);  double E = (h + 0.5 \* D) \* cos(beta1) - 0.5 \* D;  x \*= (M\_PI / 180);  result = -A \* sin(x) \* sin(x) + 2 \* B \* cos(x) \* sin(x) + C \* sin(x) - E \* cos(x);  result \*= (M\_PI / 180);  return result;  } |

삼각함수는 라디안으로 변수를 받기 때문에 입력을 라디안으로 변환하여 사용했다. 해당 함수를 작성한 후 이전에 사용한 program1\_1() 함수를 사용하여 해를 구한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

해를 구할 때 실습에서 작성하였던 program1\_1 함수를 사용하였다.

해를 구하는 과정은 아래와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명



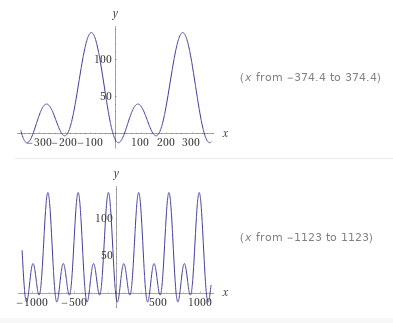
위 식의 실근은 다음과 같다.

x≈114.592 (3.14159 n - 0.100356), n element Z

x≈114.592 (3.14159 n + 0.287737), n element Z

x≈114.592 (3.14159 n + 1.28306), n element Z

x≈114.592 (3.14159 n + 1.47044), n element Z



해당 근을 구하는 식을 이용하여 해를 구하면 다음과 같다.



프로그램 구동 방법 및 소개

이 프로그램은 Newton-Rapson 방법, Secant 방법, Bisection 방법을 사용하여 함수의 근을 구하는 프로그램이다. 각 방법은 program1\_1, program1\_2, program1\_3 함수를 통해 사용할 수 있다.

추가적으로 double-precision과 single-precision을 비교하기 위해 single-precision을 사용하는 sp\_program1\_1, sp\_program1\_2 함수가 있다. 이 함수들은 Newton-Rapson 방법, Secant 방법을 single-precision을 사용하여 구현한 것이다.

각 문제를 해결하는 방법은 main함수 상에 코드로 구현되어 있다. 필요한 실험을 위하 각 부분의 주석을 해제하여 코드를 실행하면 된다. 추가적으로 각 실험마다 필요로 하는 출력이 다른데, 이는 각 함수마다 주석으로 처리된 출력문을 주석 해제하여 출력형식을 변경하면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명