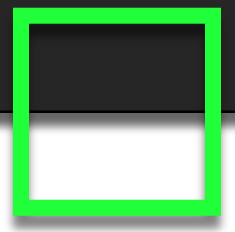
#수치 해석
#과제 2

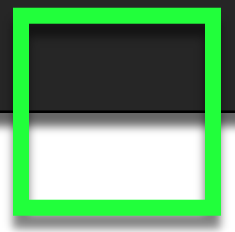
Finding Minimum

컴퓨터소프트웨어학부
2018008395 박정호

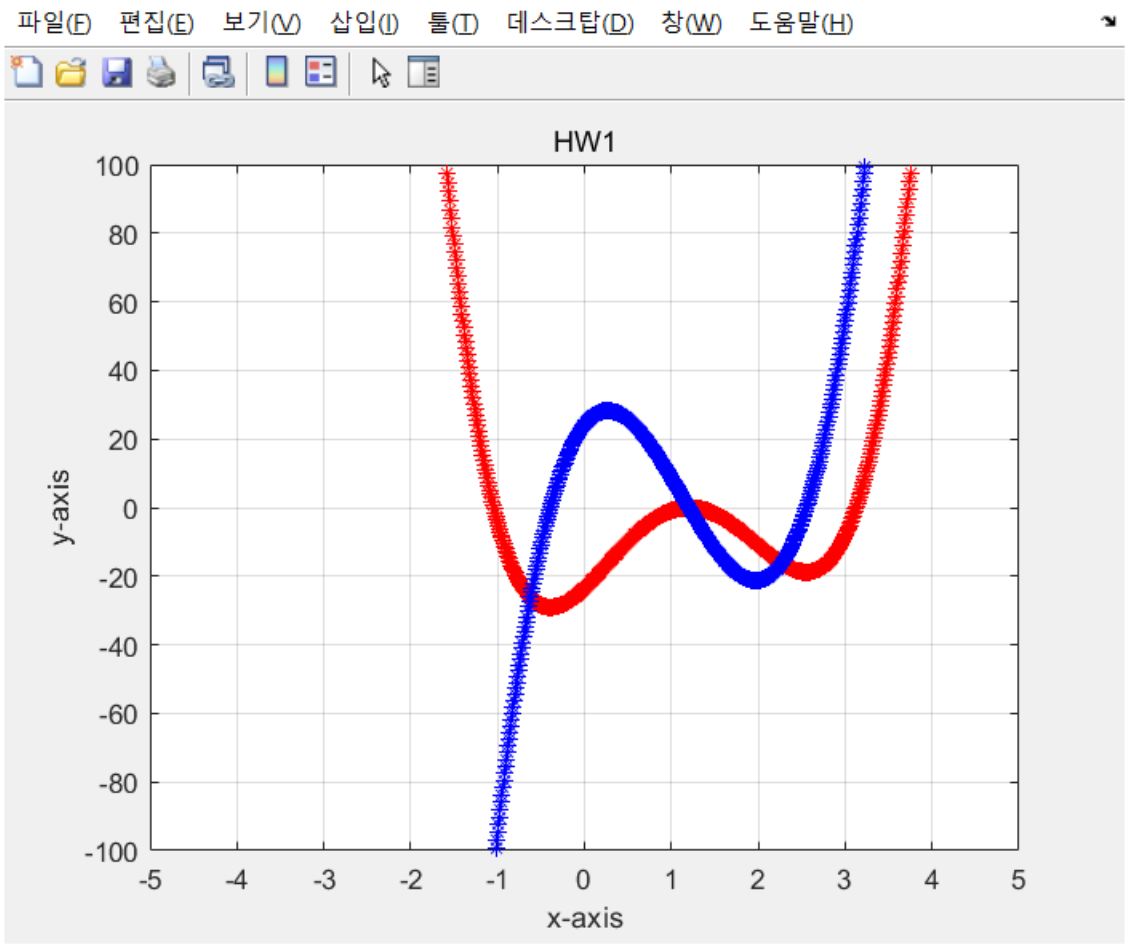


#1

Guessing



MATLAB을 이용한 함수 개형 파악



지난 과제와 동일하게 MATLAB을 사용해서 함수 개형을 파악했다. 단, 이번에는 추가로 도함수의 그래프도 그려보았다.

이번에는 최솟값의 위치를 파악하는 것이기 때문에,
convex한 모습을 보이는 x좌표를 initial value로 잡았다.

Guessing한 값은 -0.5와 2.5이다.
하지만 이번에 필자는 여러 가지 서로 다른 guessing을 통해서 값을 비교해보고자 한다. 어떤 값을 골랐는지는 다음 슬라이드에서 설명하겠다.

Guessing의 유형

1. 최솟값의 위치와 근접한 좌표
   가장 기본적인 guessing이다.
   -0.5와 2.5를 사용한다.

2. 기울기의 변화가 가파른 곳
   상대적으로 x_i와 x_{i+1}간의 차이가 작아서, 각 iteration마다 이동하는 정도가 작고,
   따라서 정해에 converge할 가능성이 높은 곳이다.
   -1과, 3을 사용한다.

3. 기울기의 변화가 가파른 곳
   상대적으로 x_i와 x_{i+1}간의 차이가 커서, 각 iteration마다 이동하는 정도가 크고,
   따라서 정해에 converge하기 어려울 수 있는 곳이다.
   0과 2를 사용한다.

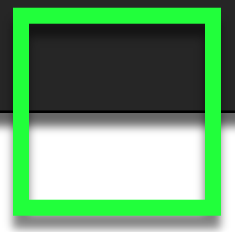
Guessing의 이유

1번 유형은 단순히 가장 정해에 근접할 것 같은 값을 육안으로 판별한 것이다. 이는 그저 필자가 구현한 newton method의 동작이 잘 되는지 확인하는 이유로 골랐다.

2, 3번 유형은 a값에 의한 교정이나 moment 값 없이 newton method를 사용했을 때, 그 정확성이 어느 정도인지 확인하기 위해 골랐다. 물론 도함수값을 구하는 방식에 따른 iteration의 횟수를 비교하기 위함도 있었다. Iteration을 비교하기 위해선 일단 각 방식의 iteration 횟수가 비교가 가능할 정도로 커야 할 텐데, 이는 정해와 guess의 차가 충분히 커야 가능하기 때문이다.

#2

구현



Newton Method

```
double newton_raphson(double x) {
    double prev;

    do {
        prev = x;
        x = x - f_p(x) / f_p_p(x);
    } while (!equal(x, prev));
    return x;
}
```

지난 과제에서 Newton-Raphson method를 구현한 코드와 같다.

단지, 기존에 함수 f와 f_p를 사용했다면, 이번에는 각각의 도함수인 f_p와 f_p_p를 사용했다. 이는 minimum을 찾는 문제가 곧 도함수의 근을 찾는 것과 같기 때문에 생긴 변경이다.

논리를 다시 설명하자면, 매 iteration마다 다음 x를 구하는데, 여기서 다음 x란, x에서 그은 접선의 방정식이 x축과 만나는 점을 말한다. 이렇게 다음 x를 반복해서 구하면서, 이번에 구한 x가 이전의 x와 같다면,(오차 범위 이내라면) 반복을 종료하고 결과로써 x를 return한다.

f_p와 f_p_p는 지난 과제와 달리, 과제에 따라서 구현을 다르게 했다. 이에 대한 설명은 다음 슬라이드에서 진행하겠다.

```
double equal(double a, double b) {
    return abs(a - b) <= 0.000001;
}
```

두 값이 (거의) 같은지를 판단하는 함수 equal이다.
단순히 두 값의 차를 구해서 오차 범위 이내인지를 판단한다.

Using Exact Derivative

```
double f_p(double x) {
    return 20 * x * x * x - 67.2 * x * x
        + 31.70544 * x + 24.161472;
}

double f_p_p(double x) {
    return 60 * x * x
        - 134.4 * x + 31.70544;
}
```

수학적으로 구해진 도함수와 이계도함수를 사용하는 방식이다.
정말 주어진 식을 직접 미분해서 도함수와 이계도함수를 하드 코딩했다.

앞의 슬라이드에서 나온 newton method 함수에서 이 함수를 사용해서 minimum의 좌표를 구해보기로 한다.

옆의 사진이 이 방식을 사용한 결과이다.
Initial value는 위에서 언급한 1번 유형을 사용했다. (-0.5와 2.5)
대략적으로 guessing과 비슷한 값이 나옴을 확인할 수 있다.

Using Approximation

```
const double h = 0.001;

double f_p(double x) {
    return (f(x + h) - f(x))
        / h;
}

double f_p_p(double x) {
    return (f(x + h) - f(x) * 2 + f(x - h))
        / (h * h);
}
```

```
-0.3946532359
2.5536532265

C:₩Users₩pch68₩source₩repos₩Numerica
 0개).
이 창을 닫으려면 아무 키나 누르세요.
```

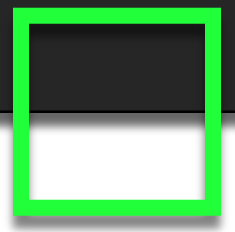
임의로 정한 interval을 사용해서 미분값을 추정하는 방식이다.
Interval은 0.001로 설정했고, 도함수와 이계도함수의 approximate는 강의 슬라이드의 정의를 따랐다. 이 함수 역시 앞에서 나온 newton method의 함수 연산 시 사용된다.

이번에는 이 방식으로 최솟값의 위치를 구해보자.

옆의 사진이 이 방식을 사용한 결과이다.
역시 Initial value는 위에서 언급한 1번 유형을 사용했다. (-0.5와 2.5)
앞에서 구한 정해와 거의 같은 값을 보여주는 것을 알 수 있다.

#3

비교



미분 가능성

모든 함수가 미분 가능하지는 않다는 건 너무나도 자명한 사실이다.
그리고 이 사실은 곧, 도함수와 이계도함수를 직접적으로 이용하는 방법은 그 적용 범위에 한계가 있다는 것을 의미한다.

이런 면에 있어서는 approximation을 이용하는 것이 좋아 보인다. 함수의 미분가능성을 굳이 판단할 필요 없이, 적당히 작은 interval 값만 있다면 newton method를 충분히 적용할 수 있기 때문이다.

물론 여기서 interval과 moment 등등을 어떻게 고르느냐에 따라 결과에 큰 차이가 생길 수 있겠지만, 적어도 적용 가능하다는 점에 있어서는 이 방법이 더 좋다고 생각한다.

Iteration 횟수 – 유형별 비교

answer	iteration
-0.3941533163	5
2.5541533163	5

Using Exact Derivative
Guessing 유형 2번

answer	iteration
-0.3946532358	6
2.5536532267	5

Using Approximate
Guessing 유형 2번

우선 기울기의 변화가 가파른, 2번 유형을 가지고 비교를 진행했다.
대체로 적은 횟수의 iteration으로 location of minimum을 찾아내는 것을 알 수 있었다.

Approximate를 이용한 방법이 iteration을 한번 정도 더 하는 것을 볼 수 있는데, 이는 유의미한 차이가 아닌 것으로 보인다. 다음 유형을 보자.

Iteration 횟수 – 유형별 비교

answer	iteration
-0.3941533163	6
2.5541533163	10

Using Exact Derivative
Guessing 유형 3번

answer	iteration
-0.3946532358	6
2.5536532268	10

Using Exact Derivative
Guessing 유형 3번

기울기의 변화가 적은, 3번 유형을 가지고 비교를 진행했다.
상대적으로 많은 횟수의 iteration으로 location of minimum을 찾아내는 것을 알 수 있었다.

여기서는 완전히 같은 횟수의 iteration으로 값을 찾아내는 것을 확인할 수 있었다. 따라서 단순히 유형별로 고르는 것만으로는 유의미한 차이를 찾아낼 수 없음을 알 수 있다.

Iteration 횟수 – 랜덤 데이터

```
exact derivative        approximate
1595026                 1584999
```

100,000회 랜덤 데이터

```
exact derivative        approximate
15948728                15849875
```

1,000,000회 랜덤 데이터

```
exact derivative        approximate
159433178               158450000
```

10,000,000회 랜덤 데이터

단순히 고른 유형별 데이터로는 iteration횟수의 비교를 할 수 없었다.

따라서 여러 번 랜덤 데이터를 가지고 실험을 반복해서 축적된 데이터를 통해서 비교해보기로 했다. 랜덤으로 구해진 데이터를 가지고 두가지 method를 적용시켜서 각 method 별 iteration의 총합을 구했다. approximate를 이용한 method가 더 많은 iteration 차이를 보였고, 평균적으로는 0.1회 정도의 차이를 보였다.

대체로 두 방법이 비슷한 효율을 가지고 있음을 보여주는 결과였다.