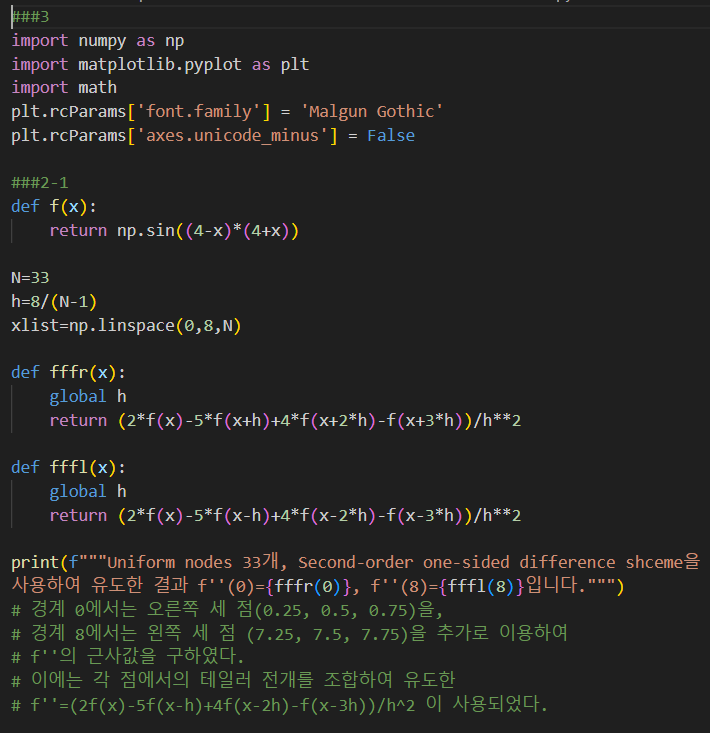
2주차 과제1 #3

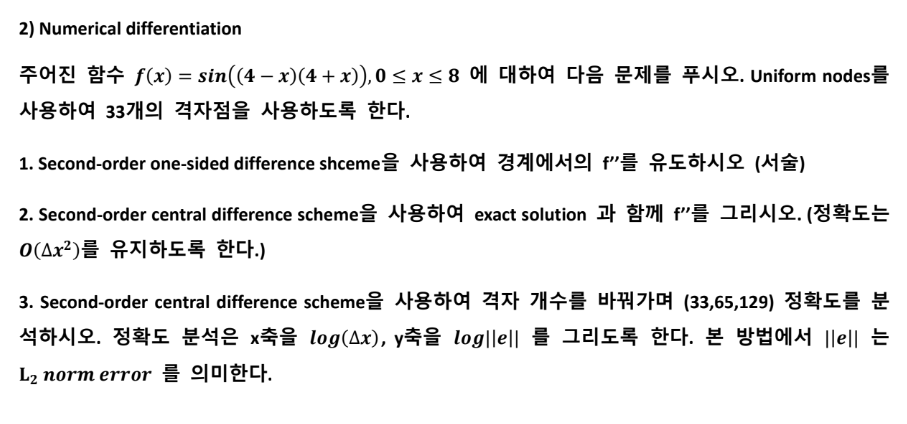
2022145079 임혜린

본 과제는 Python, VS Code를 사용하였음을 밝힙니다.

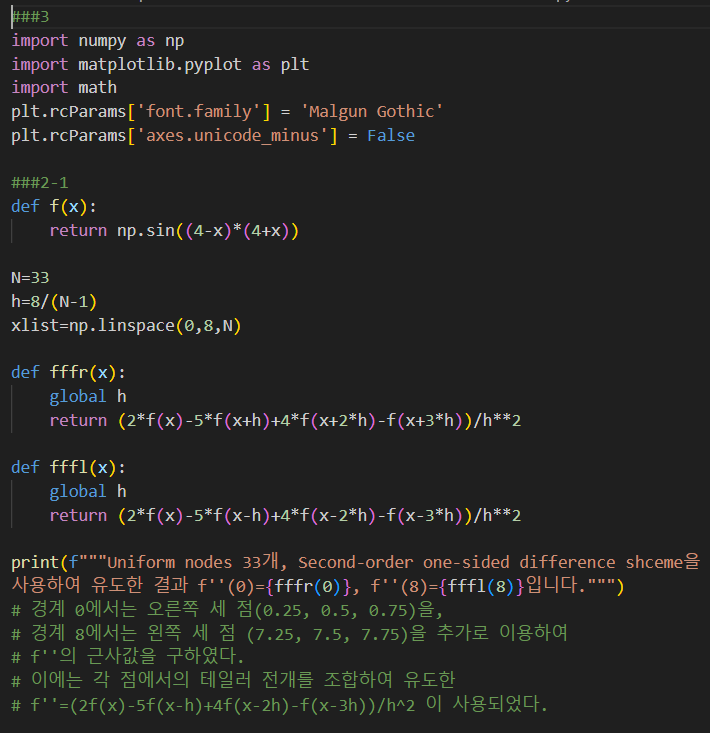
문제를 푸는데 있어 공통적으로 아래의 환경을 사용하였다.



2-1



코드



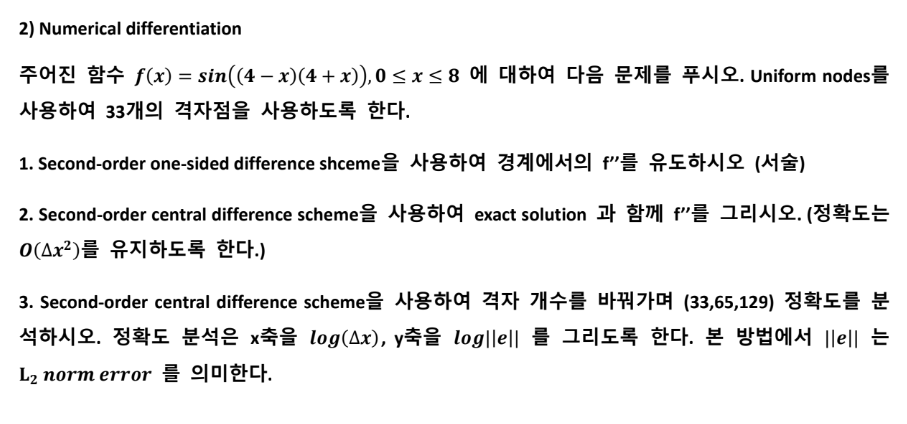
경계 0에서는 오른쪽의 세 점(0.25, 0.5, 0.75)을 이용한 3step 2차 후진 차분, 경계 8에서는 왼쪽 세 점 (7.25, 7.5, 7.75)을 이용한 3step 2차 전진 차분을 이용하여 f’’의 근사값을 구하였다.

이에는 각 점에서의 테일러 전개를 조합하여 유도한 f’’=(2f(x)-5f(x-h)+4f(x-2h)-f(x-3h))/h^2가 사용되었다.

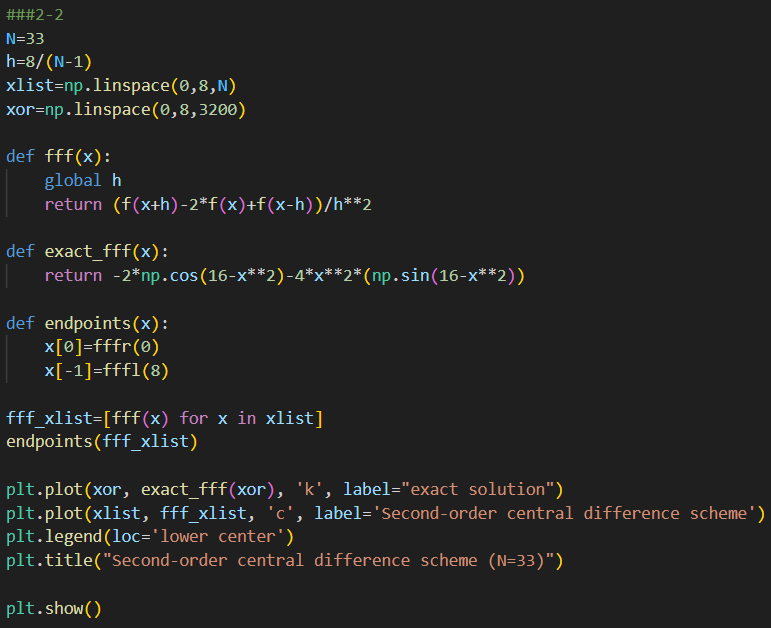
결과



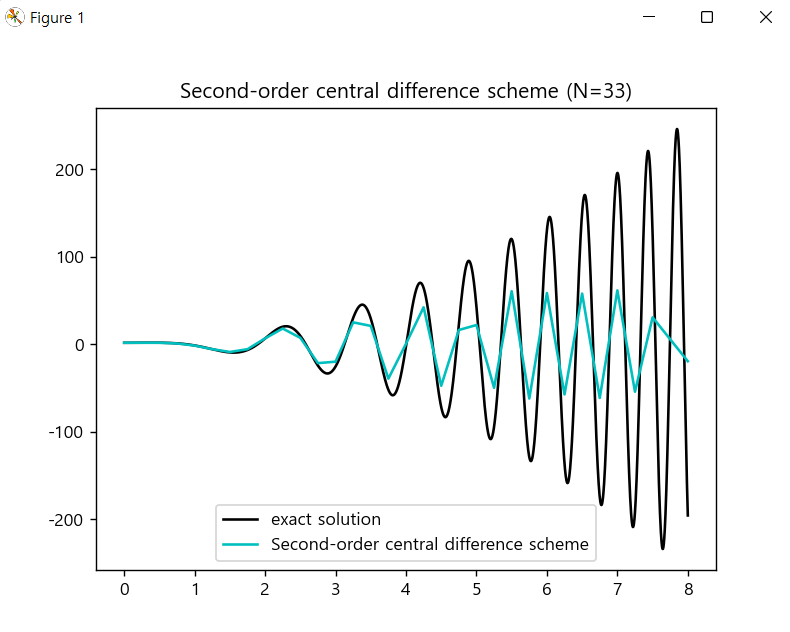
2-2



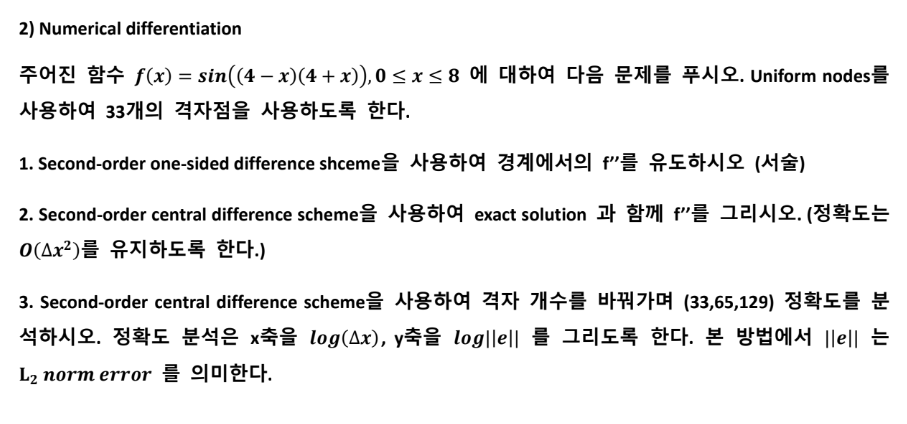
코드

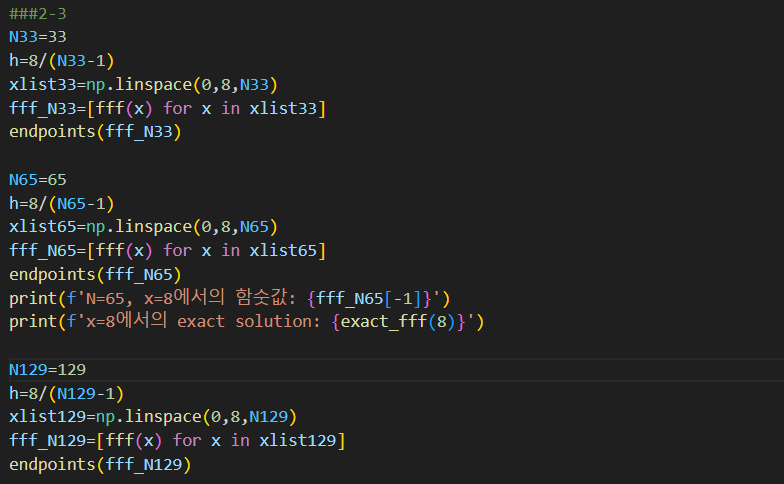


결과

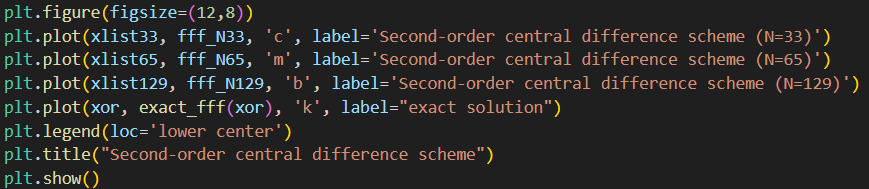


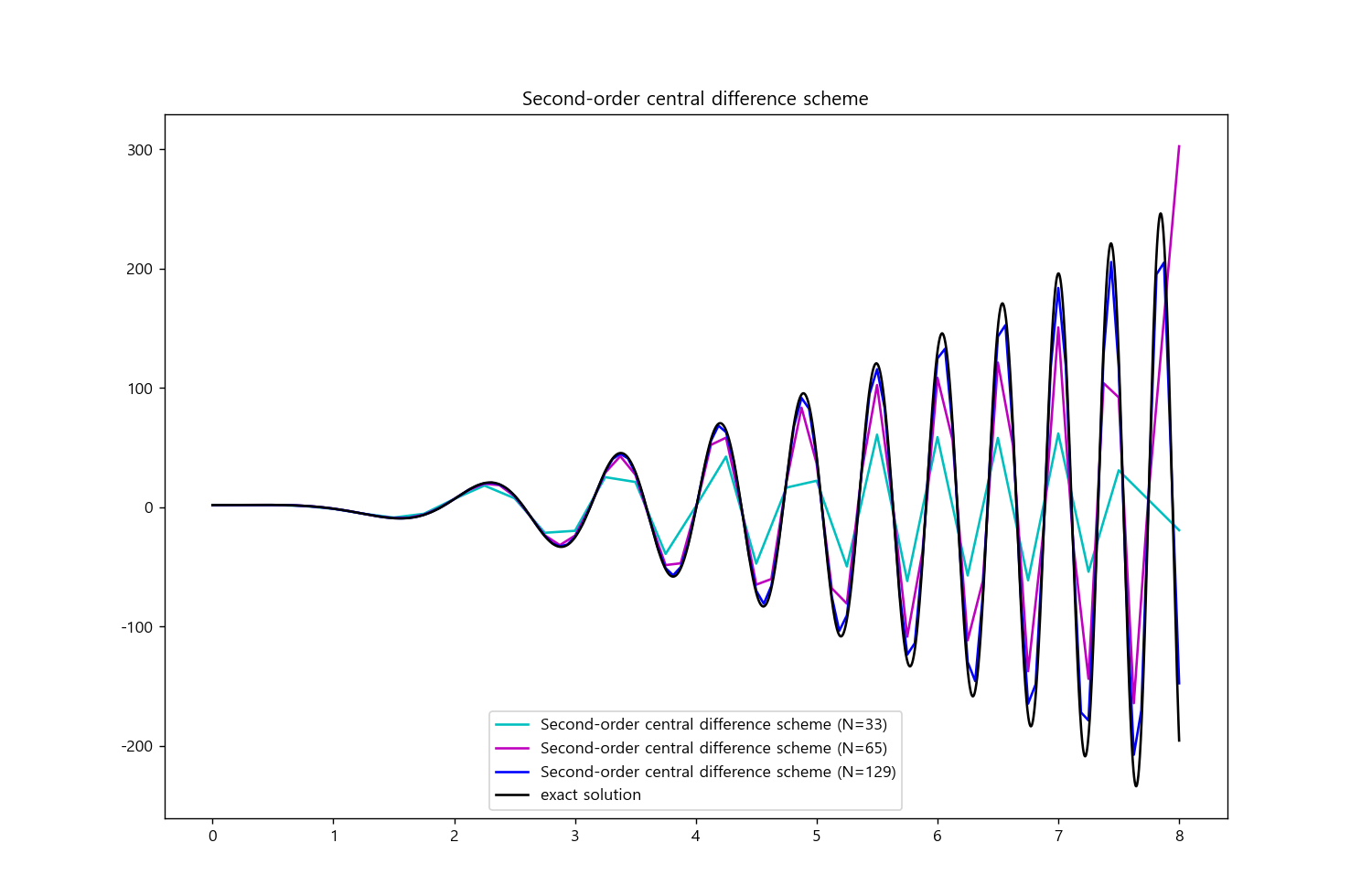
2-3

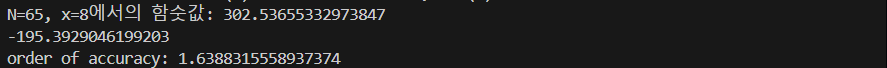


코드

정확도 분석에 앞서 N에 따른 세 f’’ 그래프를 정의하고 한 번에 그려보았다. 각 경계에서는 중앙 차분을 사용할 수 없기 때문에 2-1처럼 후진 차분과 전진 차분을 사용하였다.

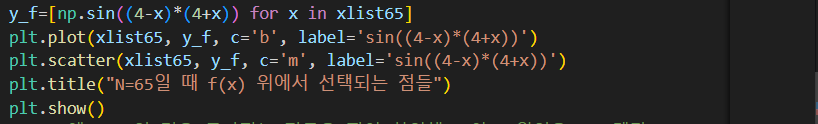


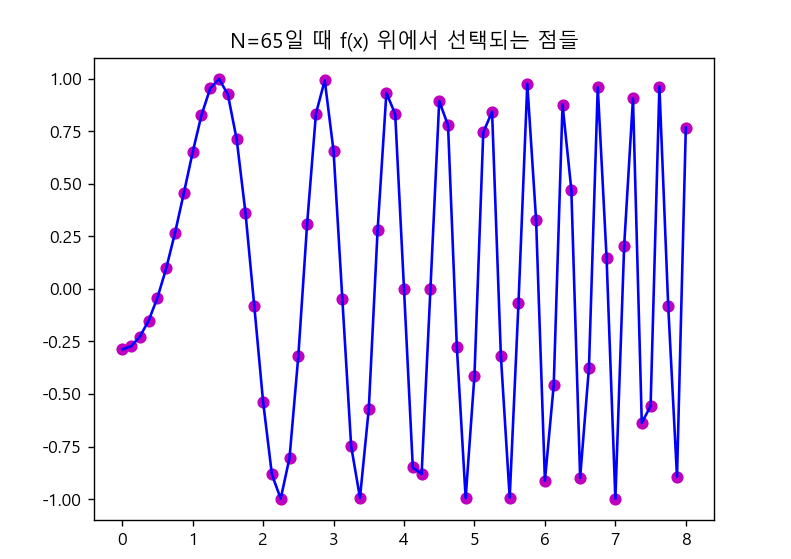




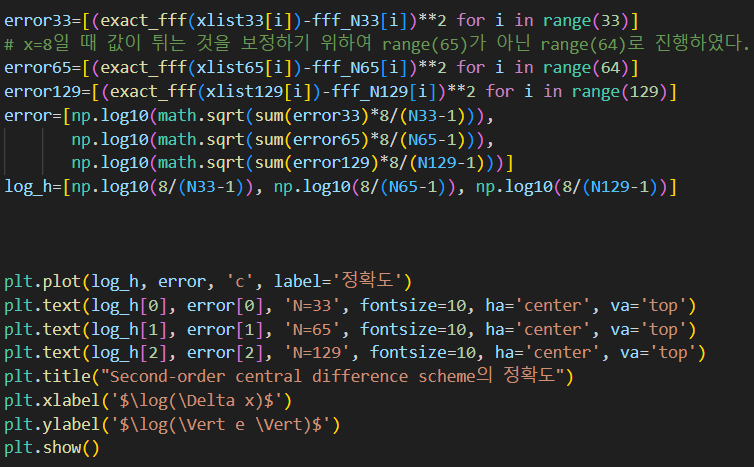
그래프를 통해 N=65인 경우 x=8인 지점에서 함숫값이 302로 튀는 것을 확인하였다. Exact solution의 해는 -195로 매우 큰 차이를 보인다. 이 오차가 아래의 오차 검증에 큰 영향을 미쳐 오차 계산 시 이 점을 제외하고 진행하였다.

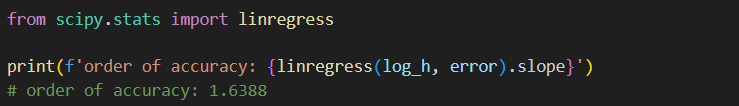
함숫값이 튀는 원인을 확인하기 위해 f(x)에 N=65인 경우 골라지는 점들을 찍어 확인해 보았다



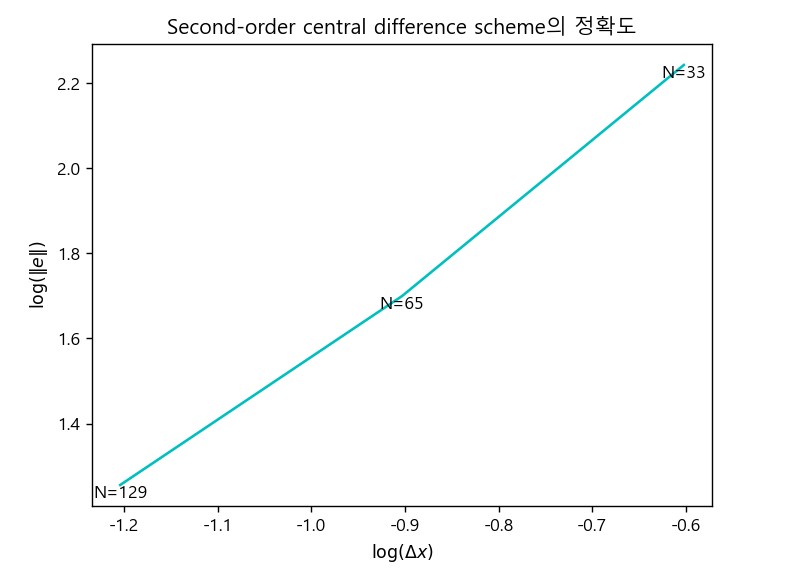


사용한 전진 차분의 식은 (2f(x)-5f(x-h)+4f(x-2h)-f(x-3h))/h^2이다. 가장 가중치가 큰 f(7.75)의 값이 극단적인 것이 주요 원인으로 보이나 직접 값을 확인해 보기 전까지는 이를 예상하기 어려울 것으로 보인다.





결과



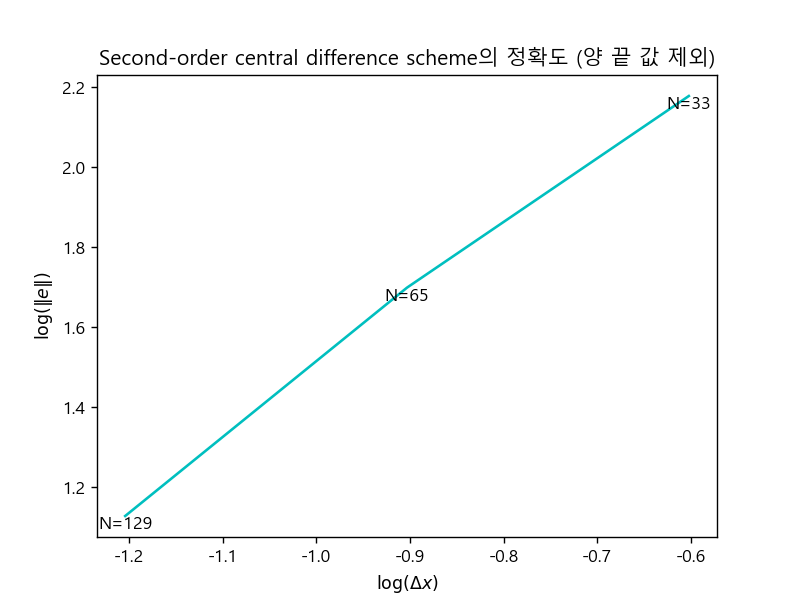
Order of accuracy: 1.6388

함수의 기울기가 유의미하지 않아 전진 차분, 후진 차분으로 만든 부분을 빼고 다시 error을 살펴보았다.





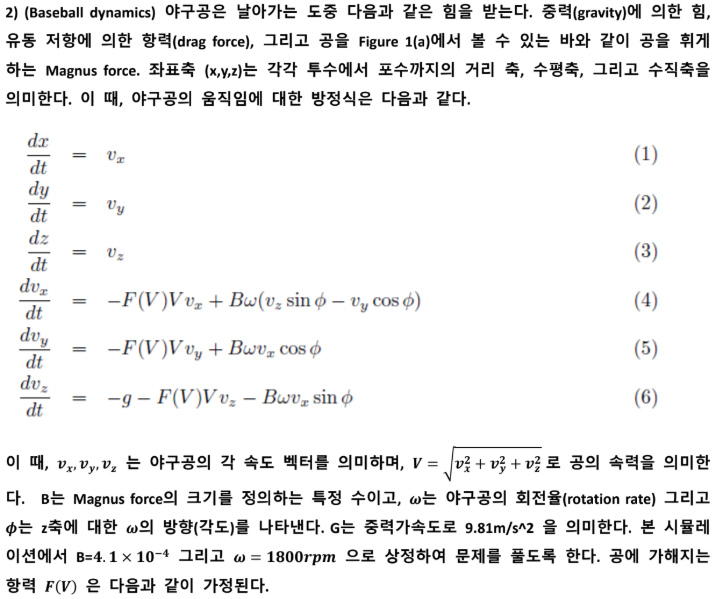
결과

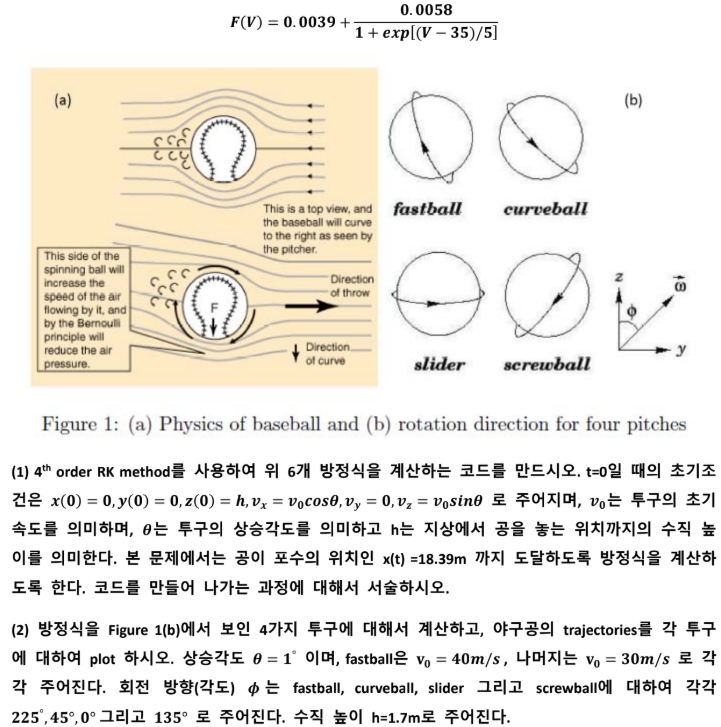


Order of accuracy: 1.7441

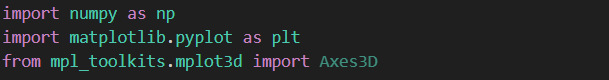
함수의 기울기가 2와 유사함을 확인하였다. 이를 통해 오차가 Δx^2에 비례하여 감소함을 알 수 있다.

2)

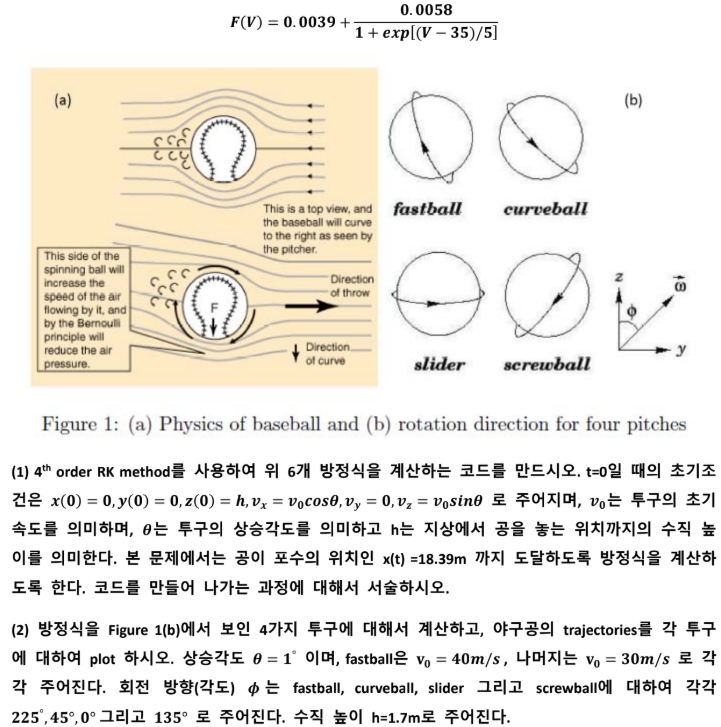




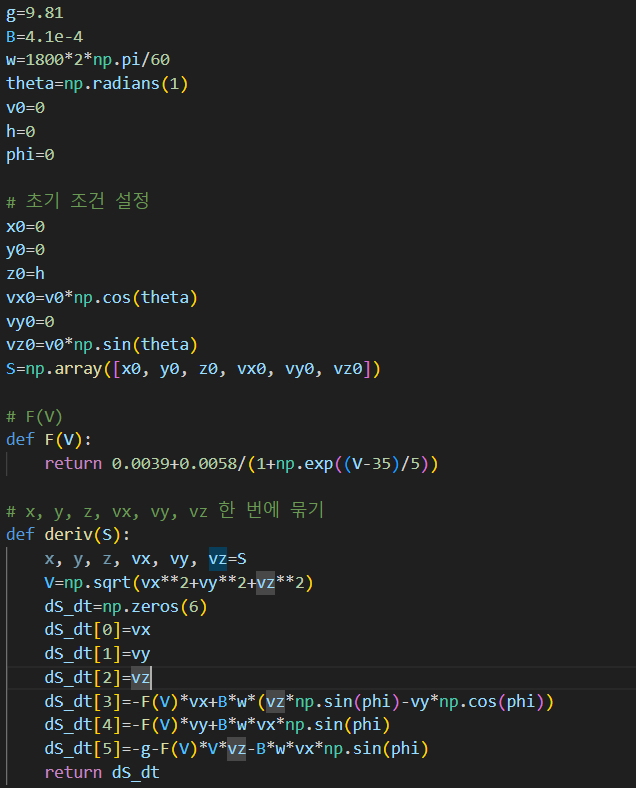
공통적으로 아래의 환경에서 코딩하였다.



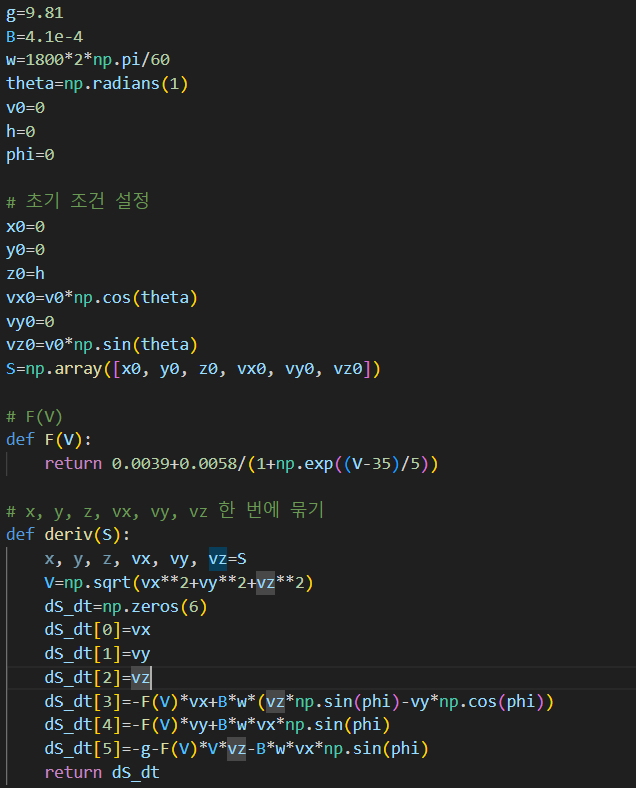
2-1



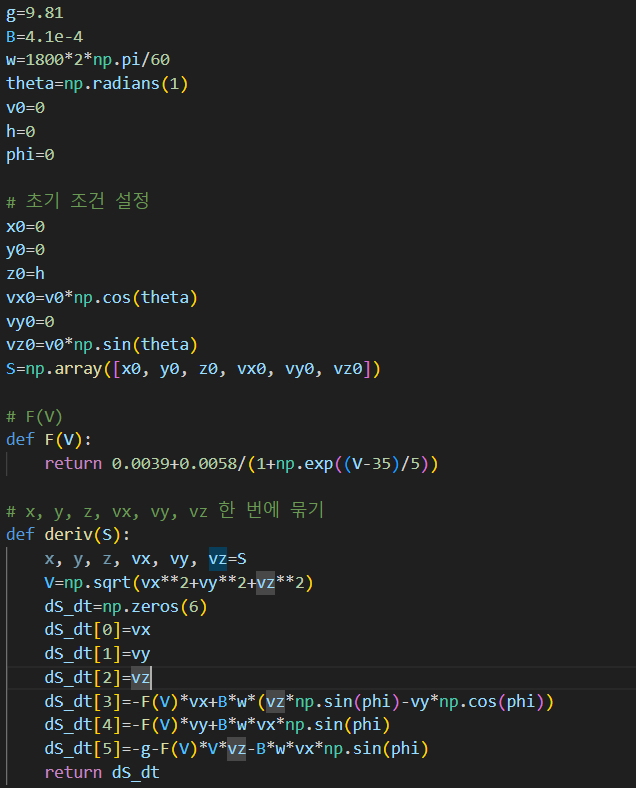
가장 먼저 상수를 정의한다. v0, h, phi는 아직 주어지지 않았으므로 0을 넣었다.



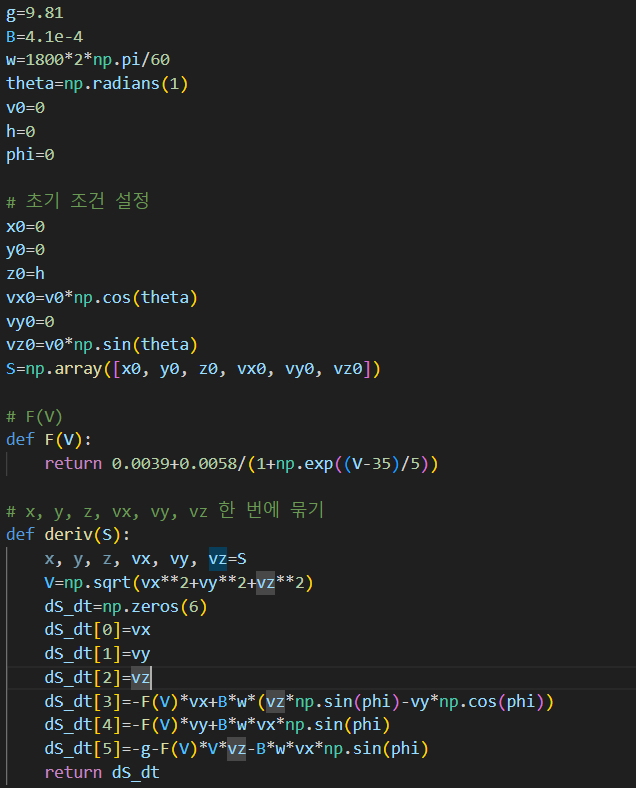
초기 조건을 설정한다. 식으로 주어진 6개의 변수의 초기 조건을 설정한 후, 넘파이 S로 묶었다.



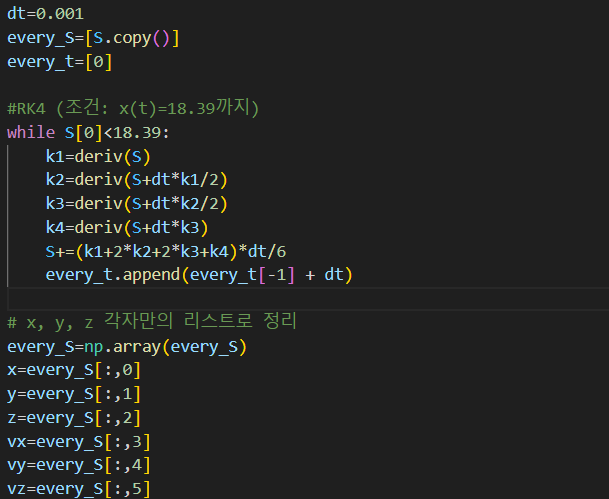
주어진 함수식인 F(V)를 정의한다.



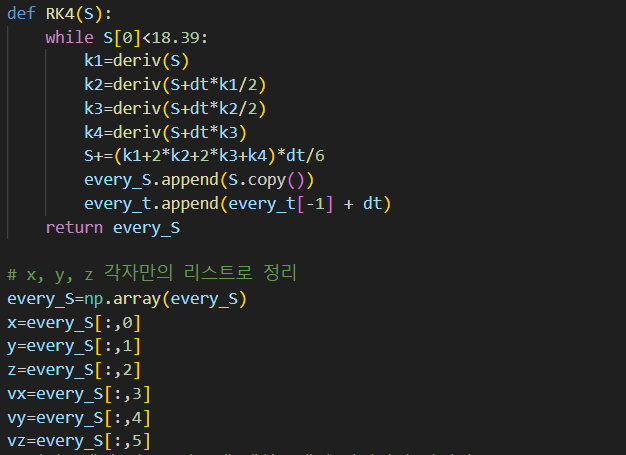
함수 deriv(S)를 정의한다. 먼저 dS/dt를 6개의 0이 있는 넘파이 배열로 정의한 후, 각각의 0를 문제에서 주어진 6개의 식에 대응되도록 하였다. 변수 6개의 초기 조건을 묶어 주었던 S를 deriv(S)의 미지수로 이용함으로써, S를 변화시키며 deriv(S)에 넣어주면 S 속 x, y, z, vx, vy, vz가 주어진 6개의 식에 순서대로 대응되도록 하였다.



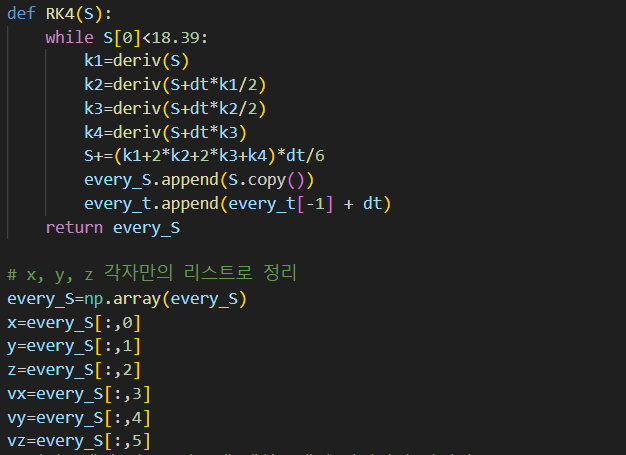
시간을 설정하고, every\_S를 정의하여 각 시간에 따른 S를 보존한다.

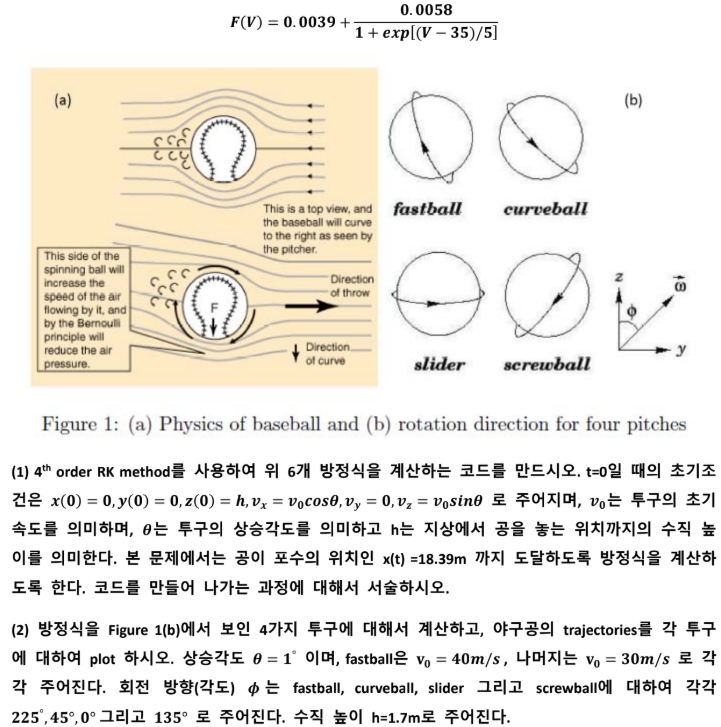


S와 deriv를 4th order RK method의 형식으로 연결하고, 반복 횟수를 정의한다. 시간에 따른 6개의 변수값은 묶여 every\_S에 들어가게 된다. 또한 every\_t에 멈추는 순간까지의 t를 저장하여 every\_S와 같은 번째를 뽑으면 해당 시간에서의 6개의 변수값을 알 수 있도록 하였다. S[0]가 18.39보다 커지는 순간에 멈추도록 함으로써, 포수 위치에 가장 가까운 때의 값을 마지막으로 반복문이 끝나도록 하였다.



마지막으로 이렇게 모인 every\_S 속 값을 변수에 따라 묶어주면 6개의 변수 각각의 시간에 따른 값들이 모이게 된다.





코드의 큰 틀은 2-1과 같다. 다만, 마지막에 every\_S으로부터 얻어야 할 값은 x, y, z 뿐이기 때문에 이것만을 모으도록 하였고, 이 x, y, z를 모아 궤적을 그려주는 함수 xyz\_3d를 정의하였다.

또한 포수에게 가장 가까운 두 좌표, 즉 18.39의 직전과 직후의 (x, y, z) 값을 나타내어 공이 어느 좌표에 도착하는지 알 수 있게 하였다.

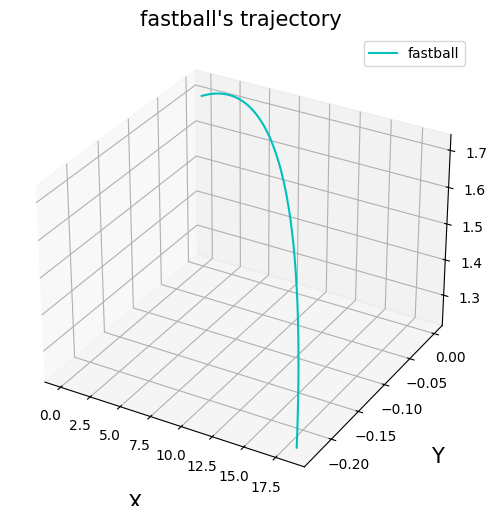
* Fastball의 경우



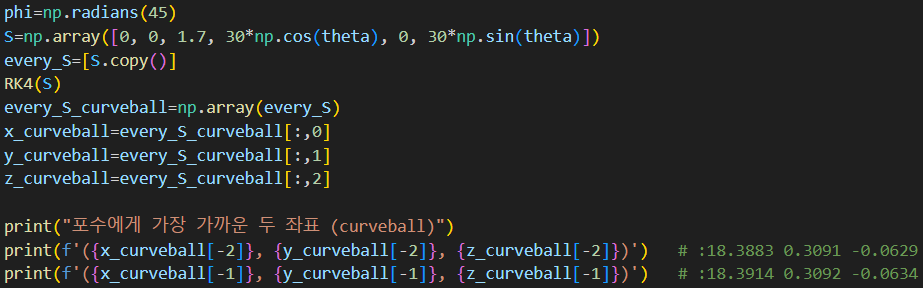


Fast ball의 그래프 코드와 결과



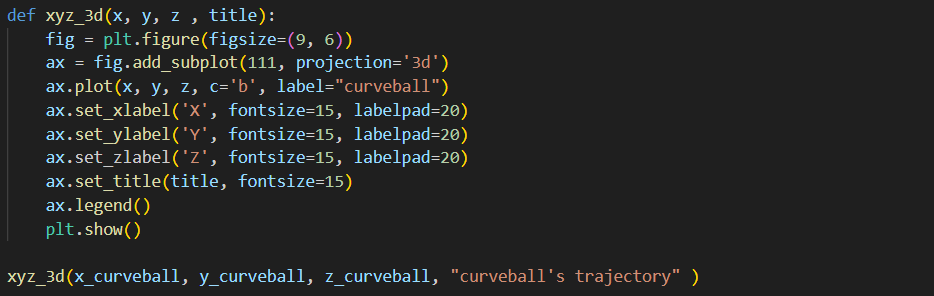
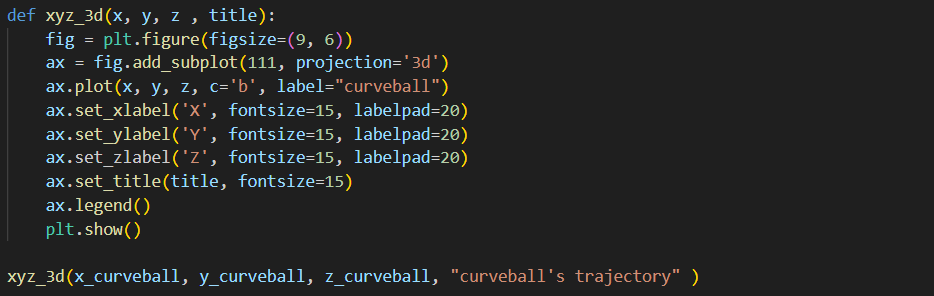


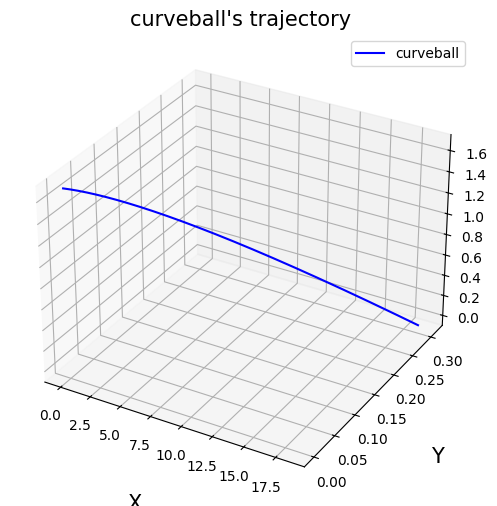
* Curveball의 경우



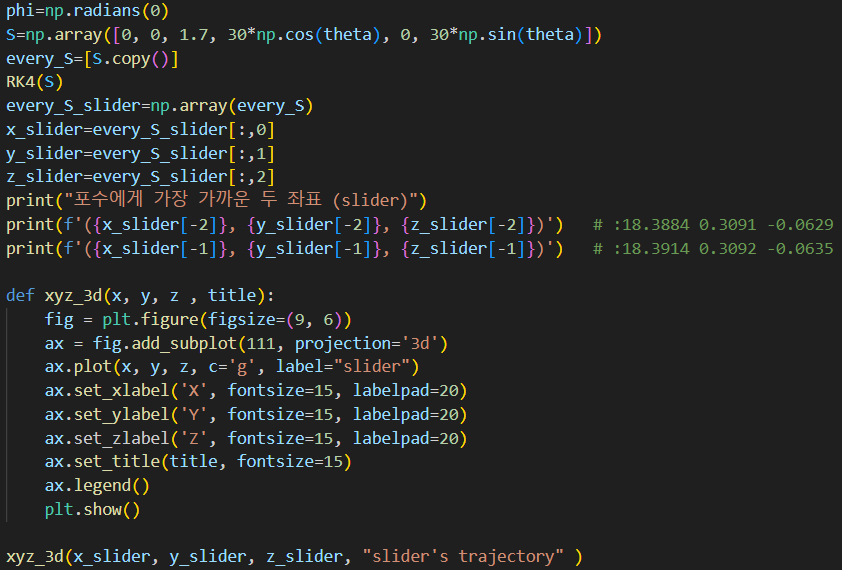


Curveball의 그래프 코드와 결과



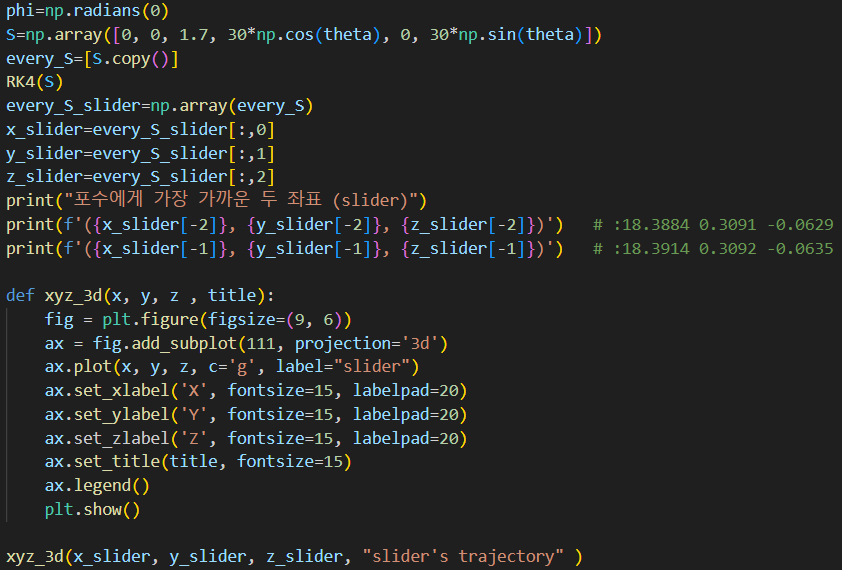


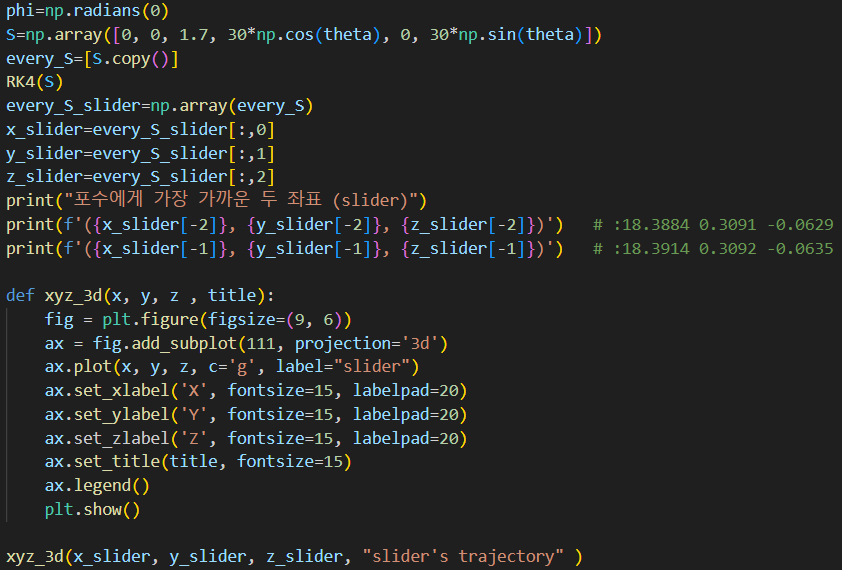
* Slider의 경우

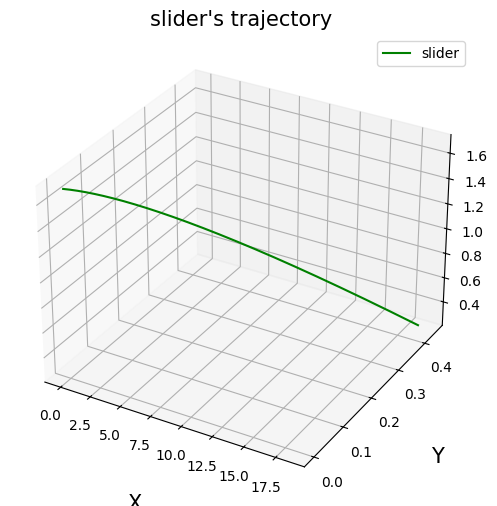




Slider의 그래프 코드와 결과







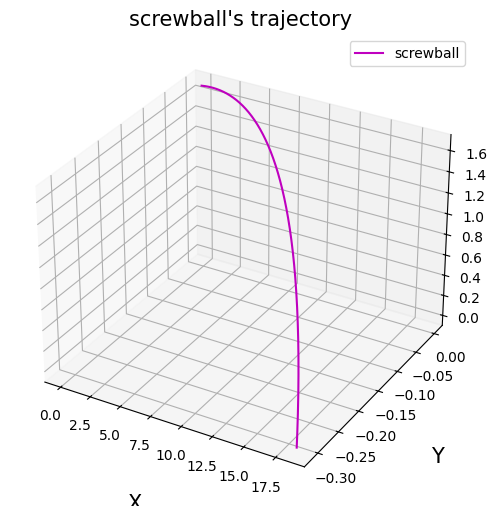
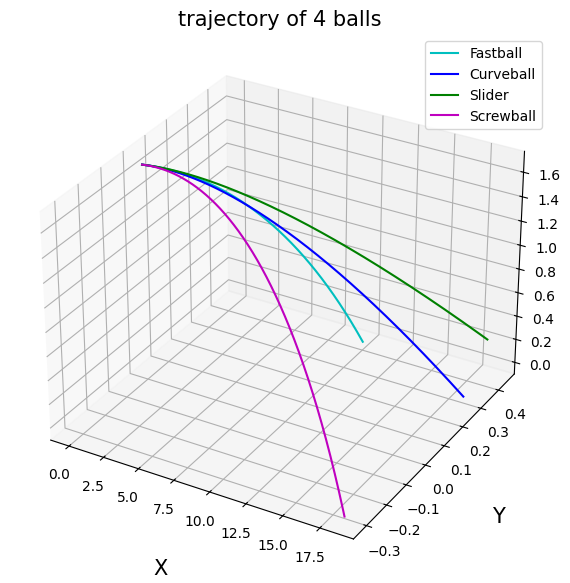
* Screwball의 경우





Screwball의 그래프 코드와 결과





마지막으로 네 가지 공의 궤적을 전부 함께 살펴보면 이와 같다.