**Recursive Filter Examples**

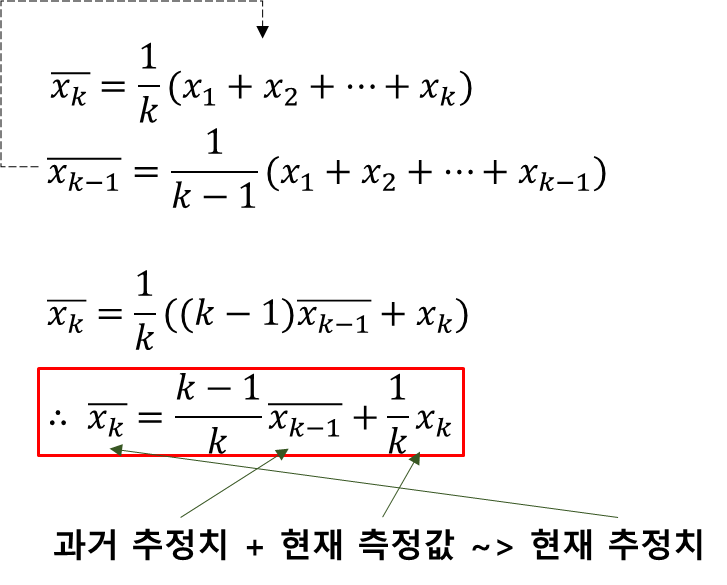
**Planning & Control**

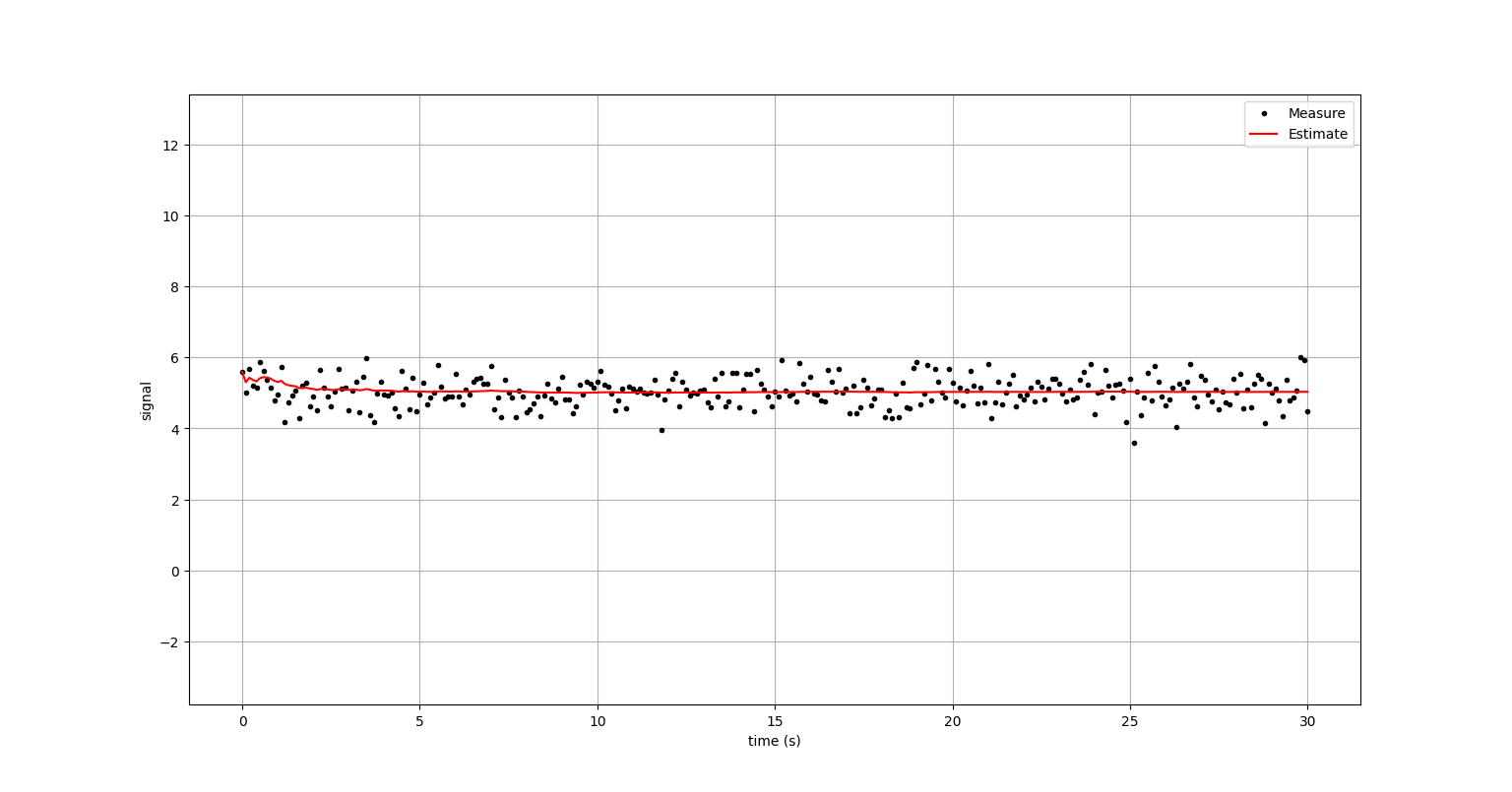
**[1기] 조환영**

**1. Average Filter**

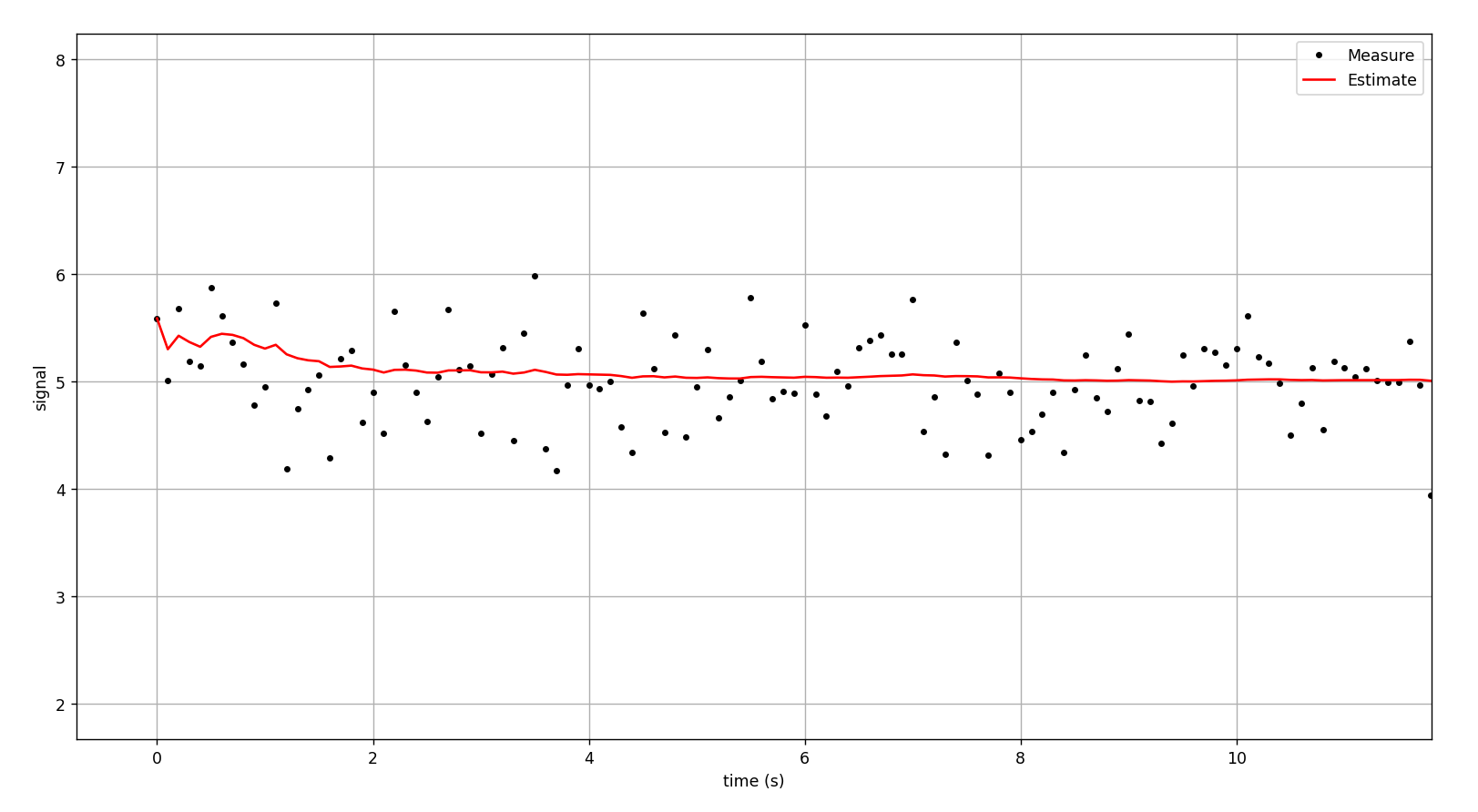
- 아주 간단한 재귀필터로 모든 데이터의 평균을 추정치로 사용한다.

- 모든 데이터를 사용하기 때문에 최근 데이터의 양상이 잘 반영되지 않는다.



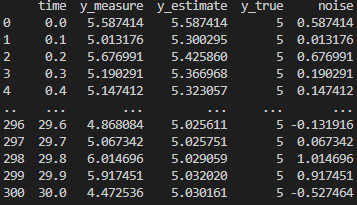


그래프 앞쪽을 확대해서 살펴보면,



초반엔 측정값의 추세를 잘 따라가다 시간이 지날수록 그냥 평균값에 머물러 있다. 이는 뒤로 갈수록 이전 데이터가 쌓여 최근 데이터의 비중이 약해지고 있음을 나타낸다.

y\_measure과 과거 y\_estimate로 계산된 y\_estimat들의 계산 결과이다.

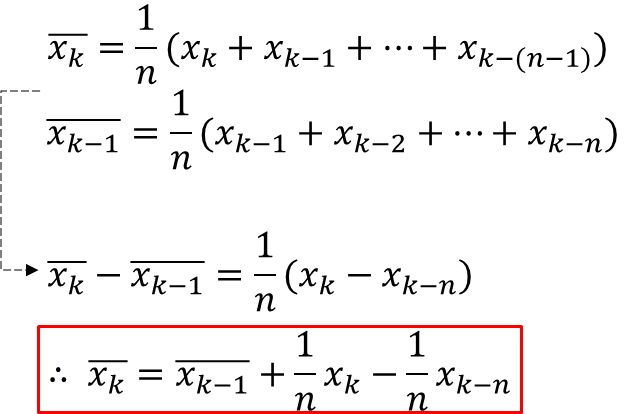
(그래프 상에서 빨간 선을 나타내는 값)

**2. Moving Average Filter**

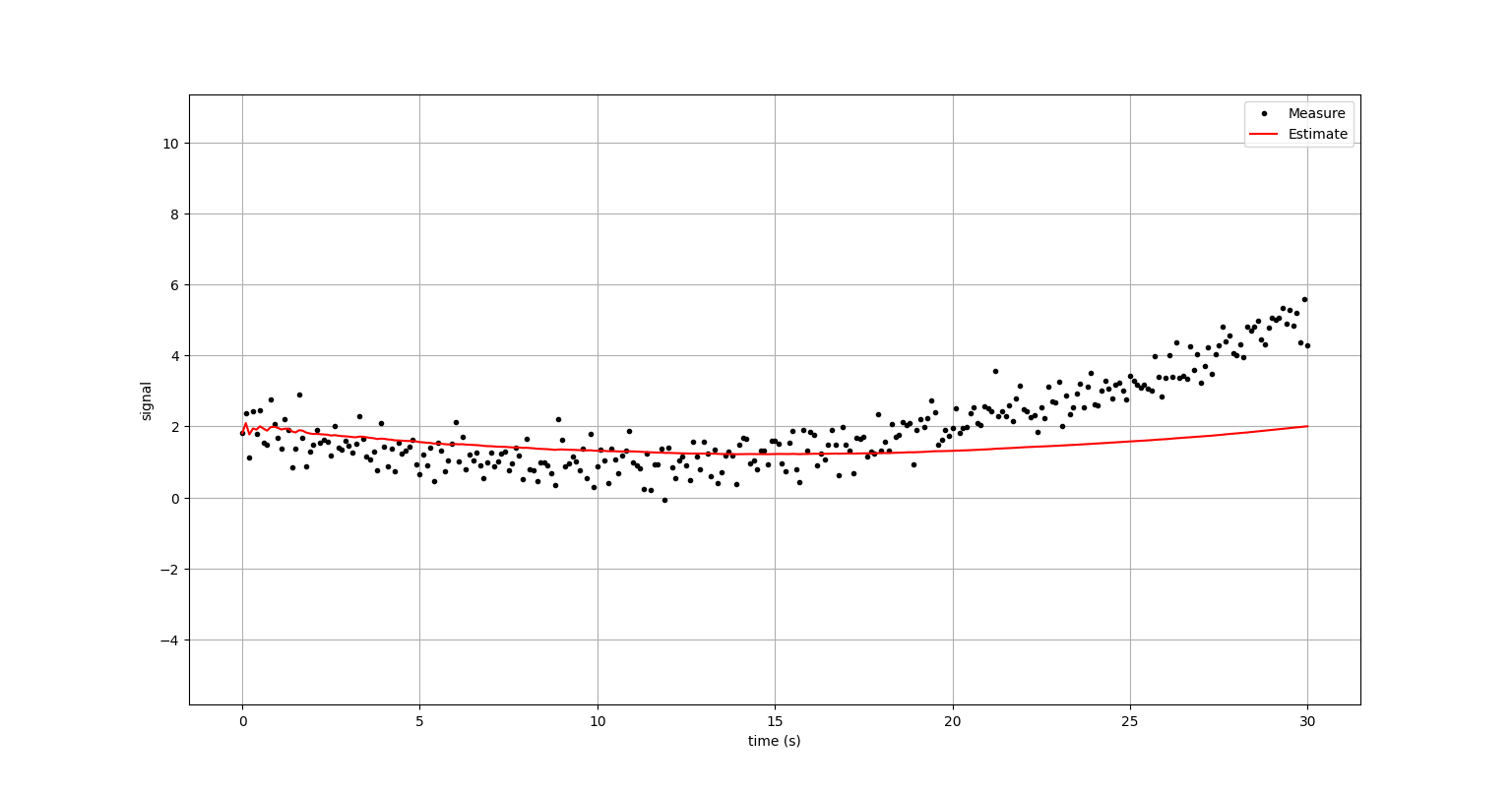
- 평균필터의 단점을 보완하고자, 시스템의 동적인 변화를 반영한 필터이다.

- 평균을 계산할 데이터의 구간을 정함으로써 과거 데이터를 어느 정도 무시할 수 있다.

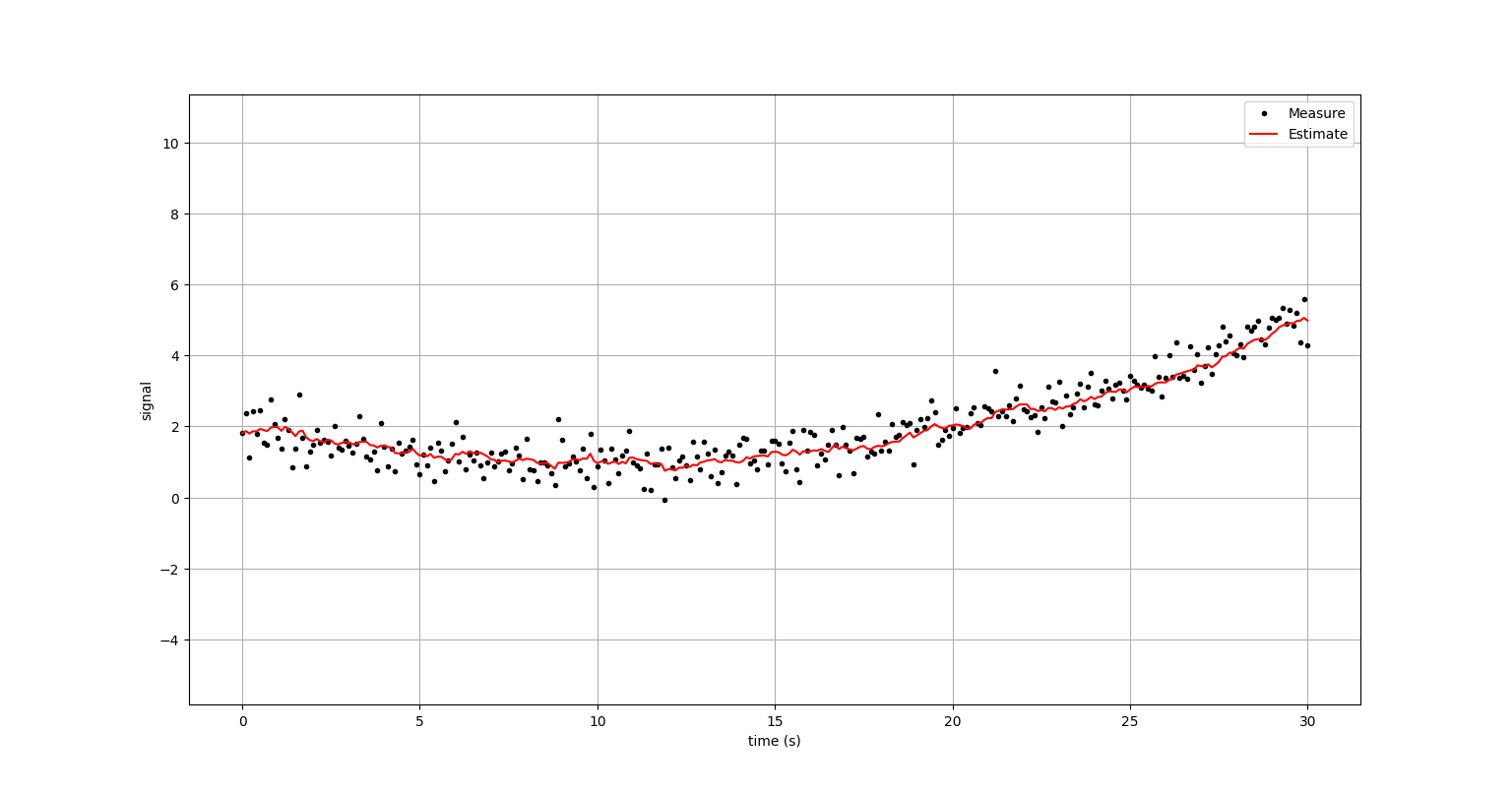
- 하지만 여전히 ‘평균’ 이기 때문에, 계산되는 모든 데이터의 가중치가 동일하다는 한계가 있다.



1. Average Filter를 썼을 때

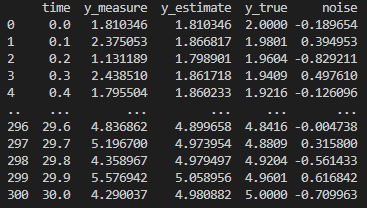


1. Moving Average Filter를 썼을 때



한눈에 다름을 알 수 있다.

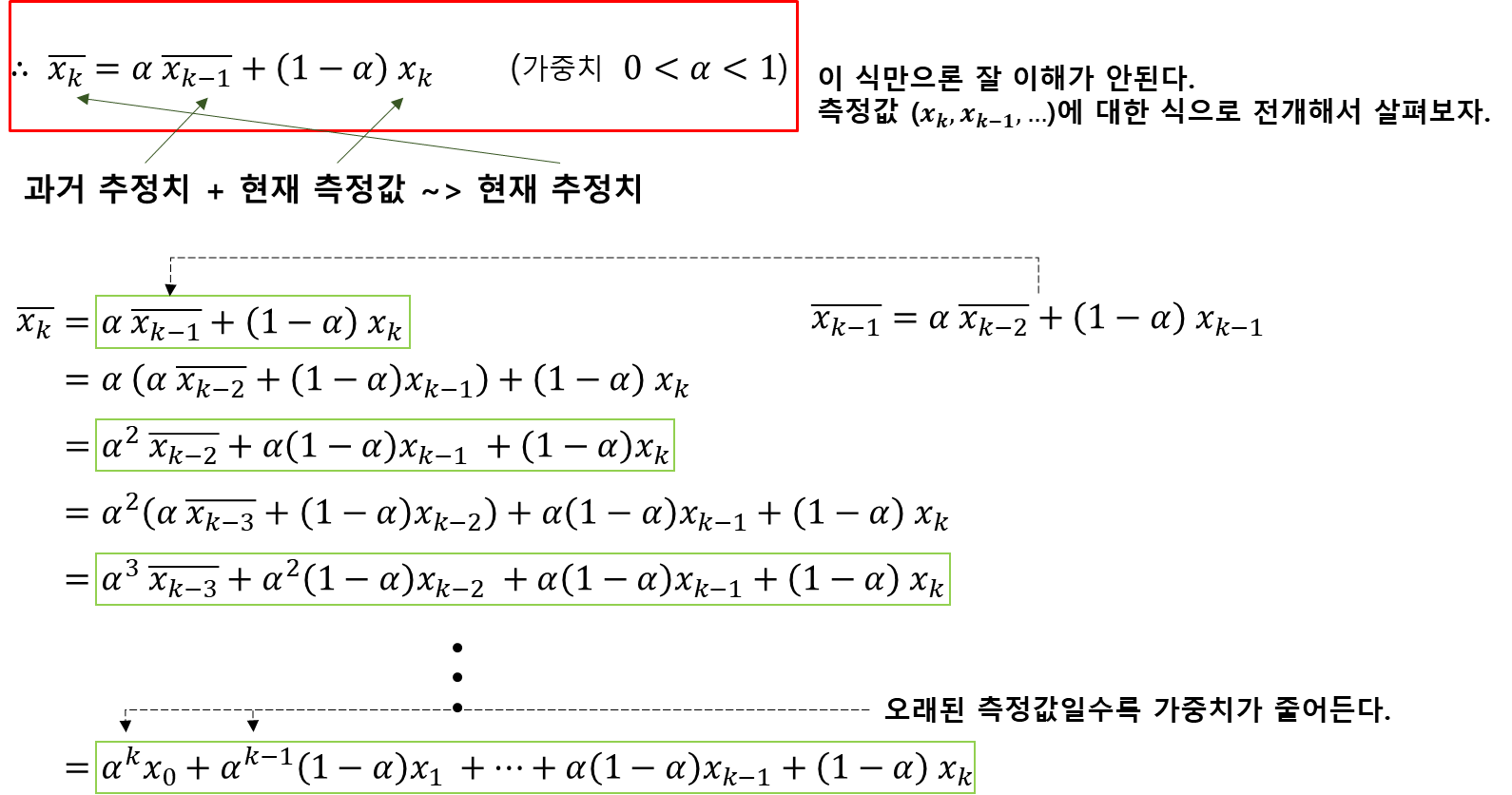
이동평균 필터가 확실히 최신 데이터의 경향을 더 잘 반영한다.



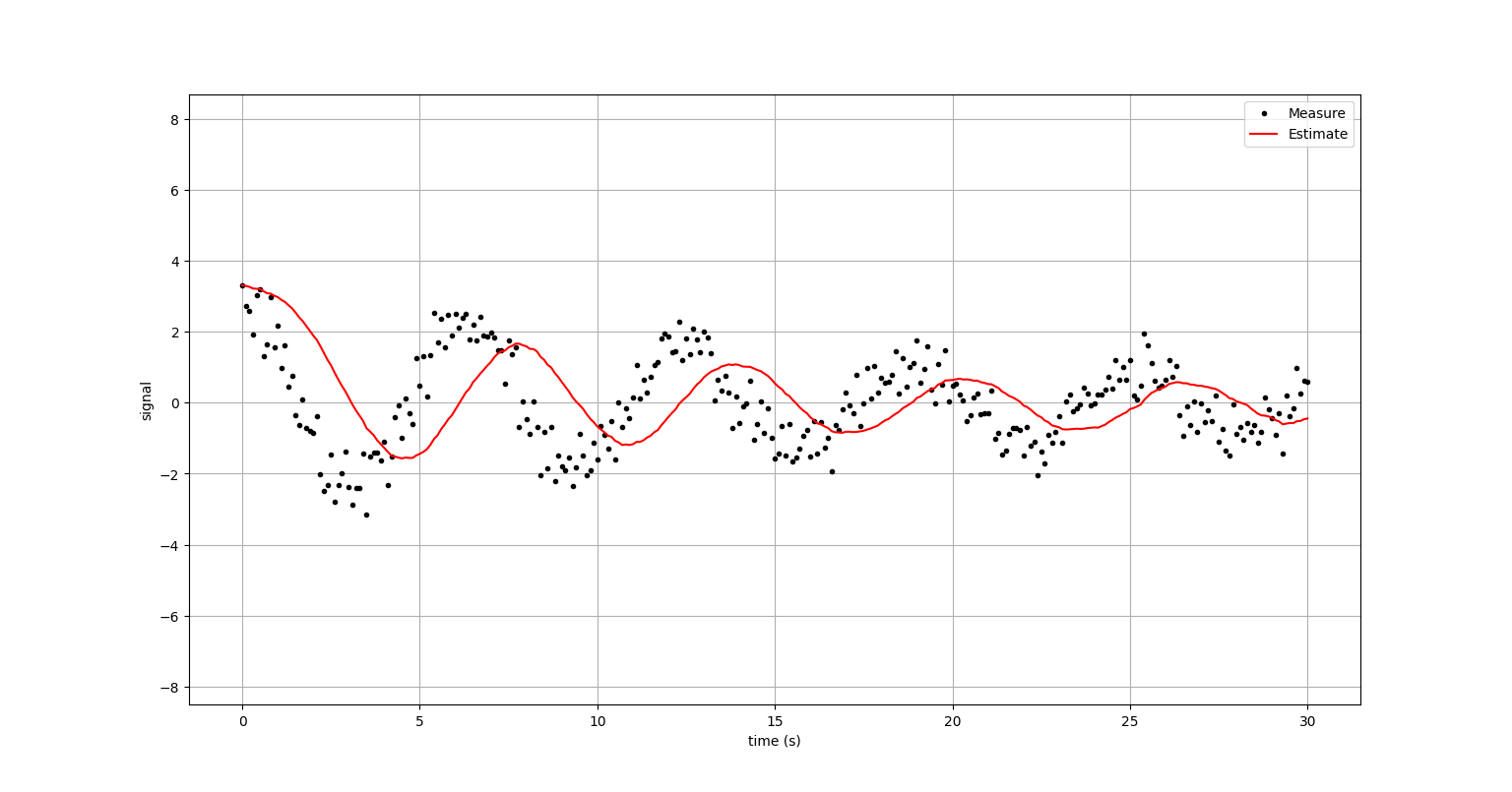
**3. Low Pass Filter**

- 파장이 짧은 고주파 노이즈 성분은 제거하고 저주파 성분을 통과시키는 필터이다.

- 평균 필터의 단점을 보완하고자 데이터의 가중치를 조절할 수 있도록 설계된 필터이다.



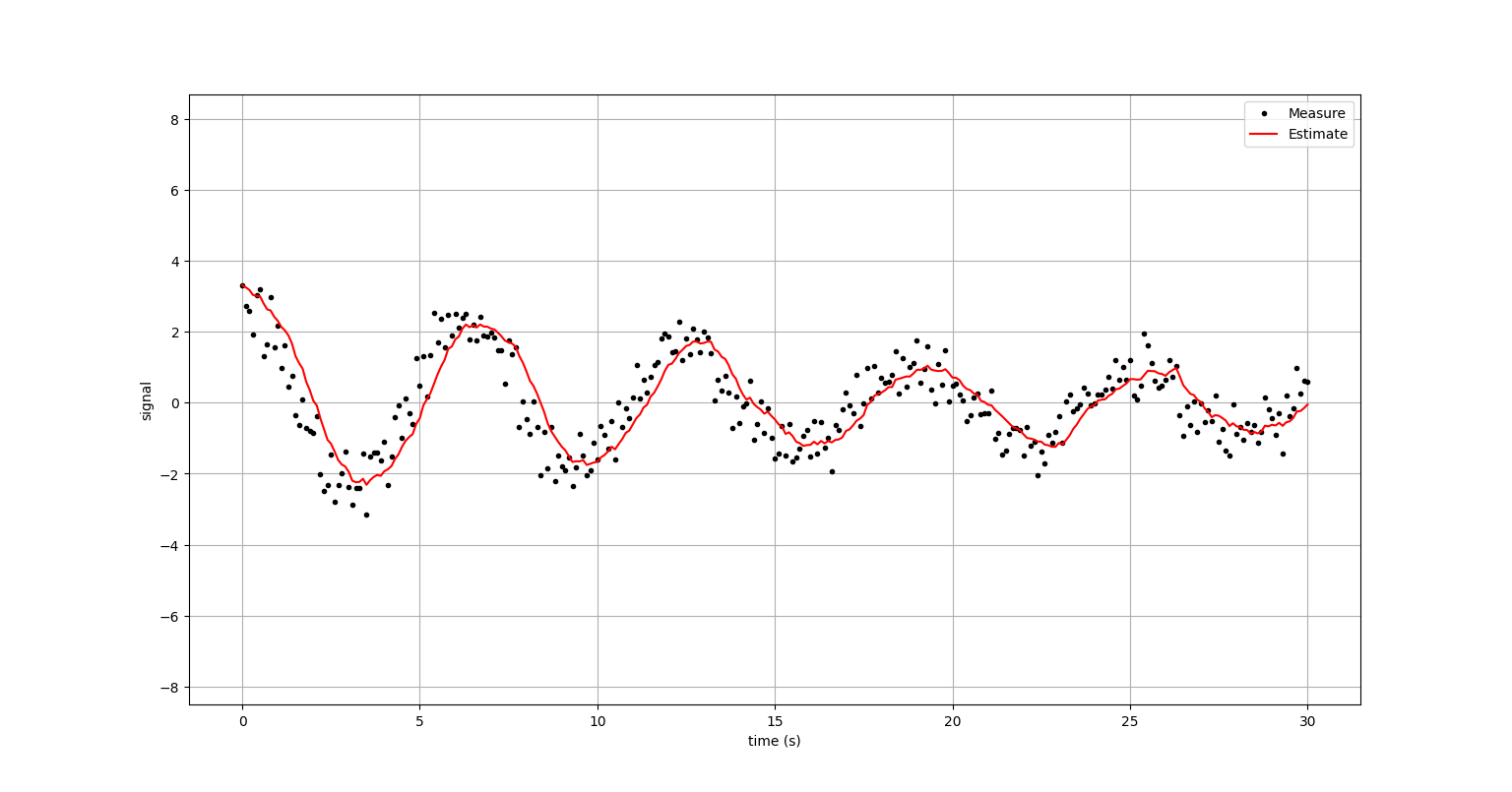
1) Moving Average Filter 썼을 때



차이를 좀 보이게 하기 위해 평균을 계산할 데이터 수 n=30으로 크게 잡았다.

확실히 추정치가 많이 밀리는 것이 보인다.

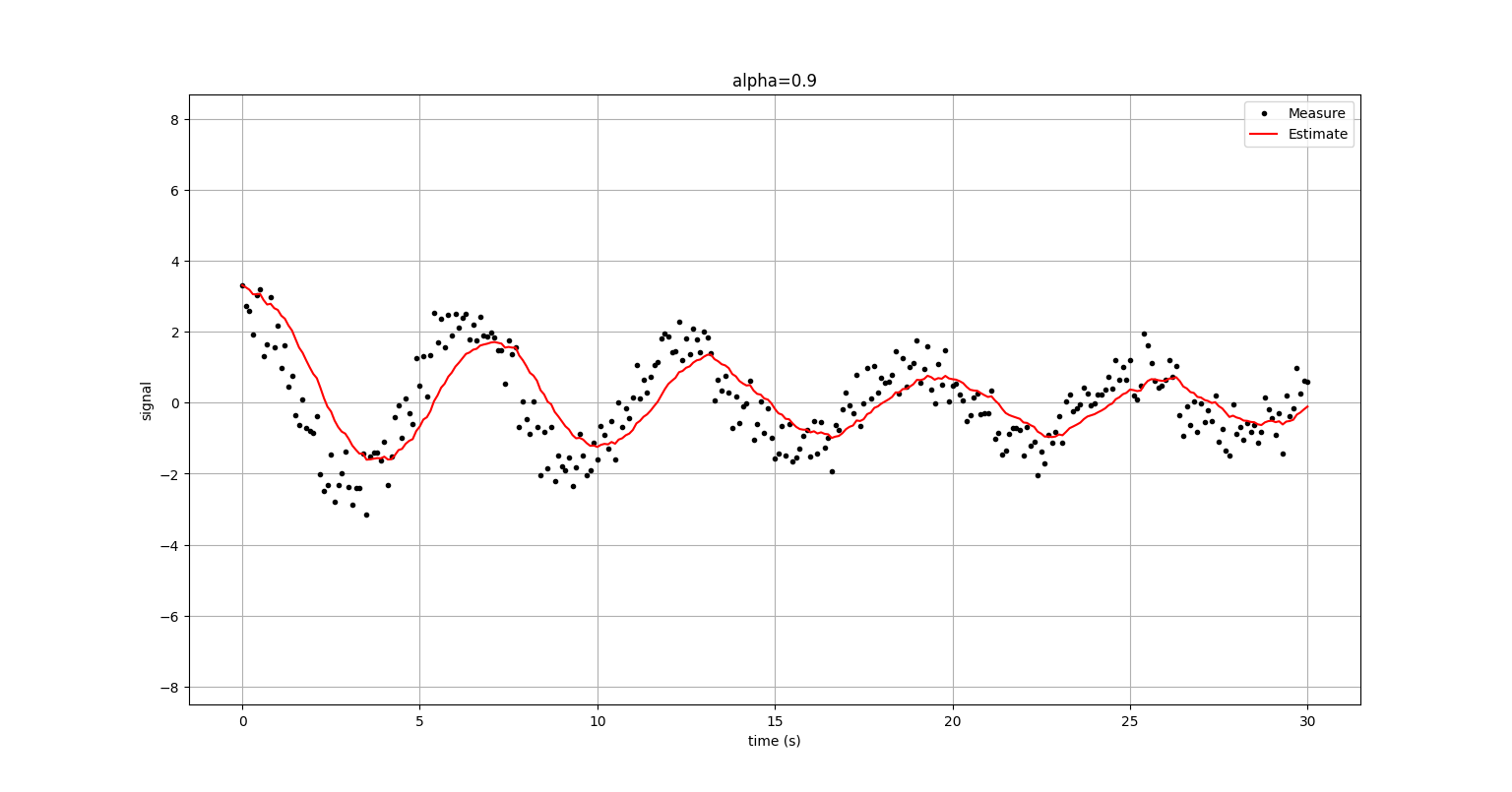
하지만, N을 10정도로 작게 잡으면 아래와 같이 좋은 결과가 나온다.



2) Low Pass Filter 썼을 때

얼핏 보면 이동평균필터보다 필터링이 더 잘 안된 것처럼 보이지만, 편차가 심한 노이즈 들을 더 잘 걸러내서 더 완만하고 안정적인 결과를 도출하고 있다고 해석할 수도 있을 것 같다.

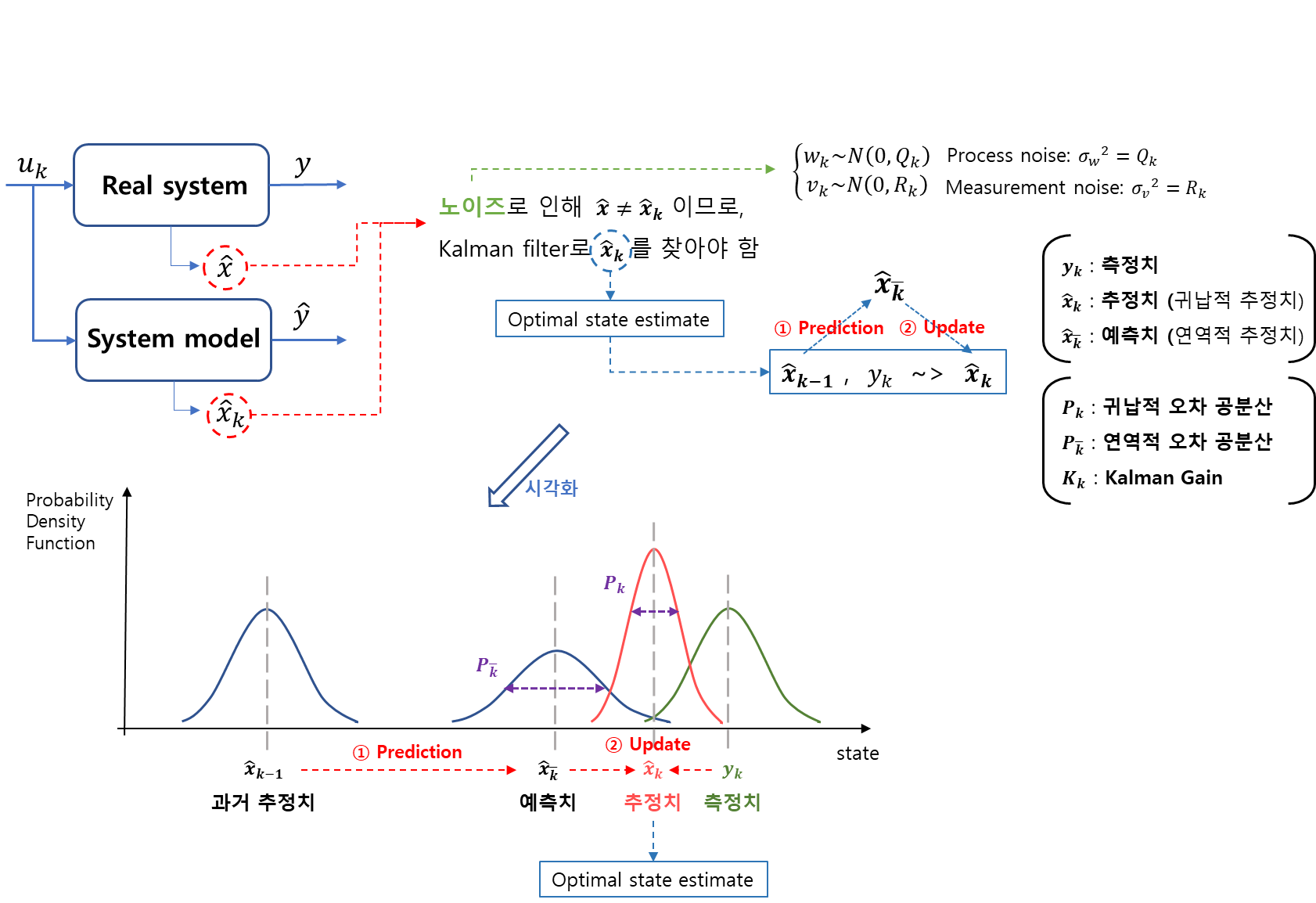
이건 필터의 사용 목적에 따라 다르지 않을까?

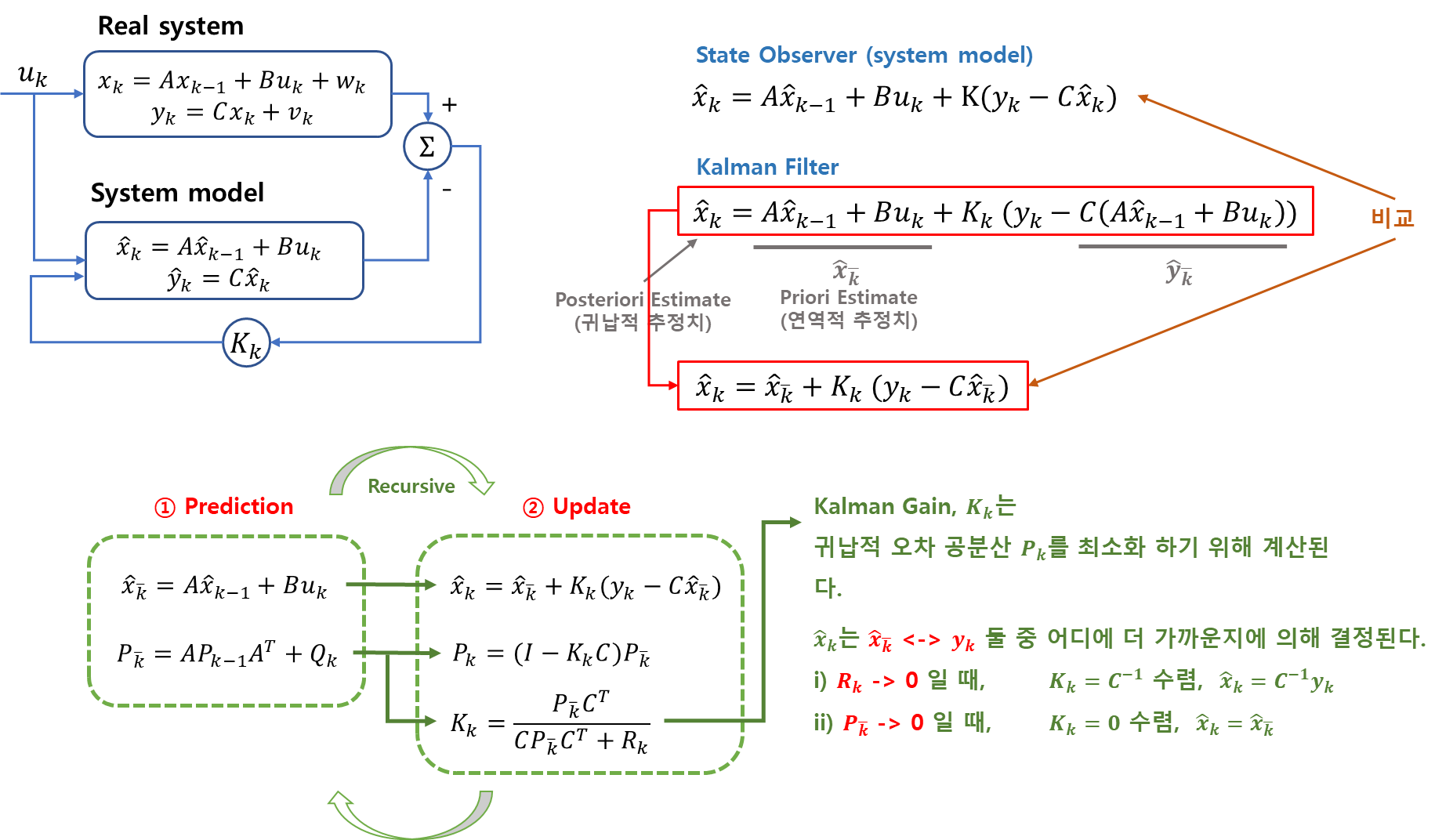


**4. Kalman Filter**

- 필터의 꽃, 가장 많이 사용되고 그만큼 중요한 필터라고 알고 있다.

- 시스템의 최적의 상태를 찾기 위해 확률 시스템으로 gain을 구하는 방식을 취한다.





- 내가 공부하면서 정리한 내용이다.

- 문제에서 주어진 시스템의 모델은

-> 측정값: 속도 dx\_k/dt

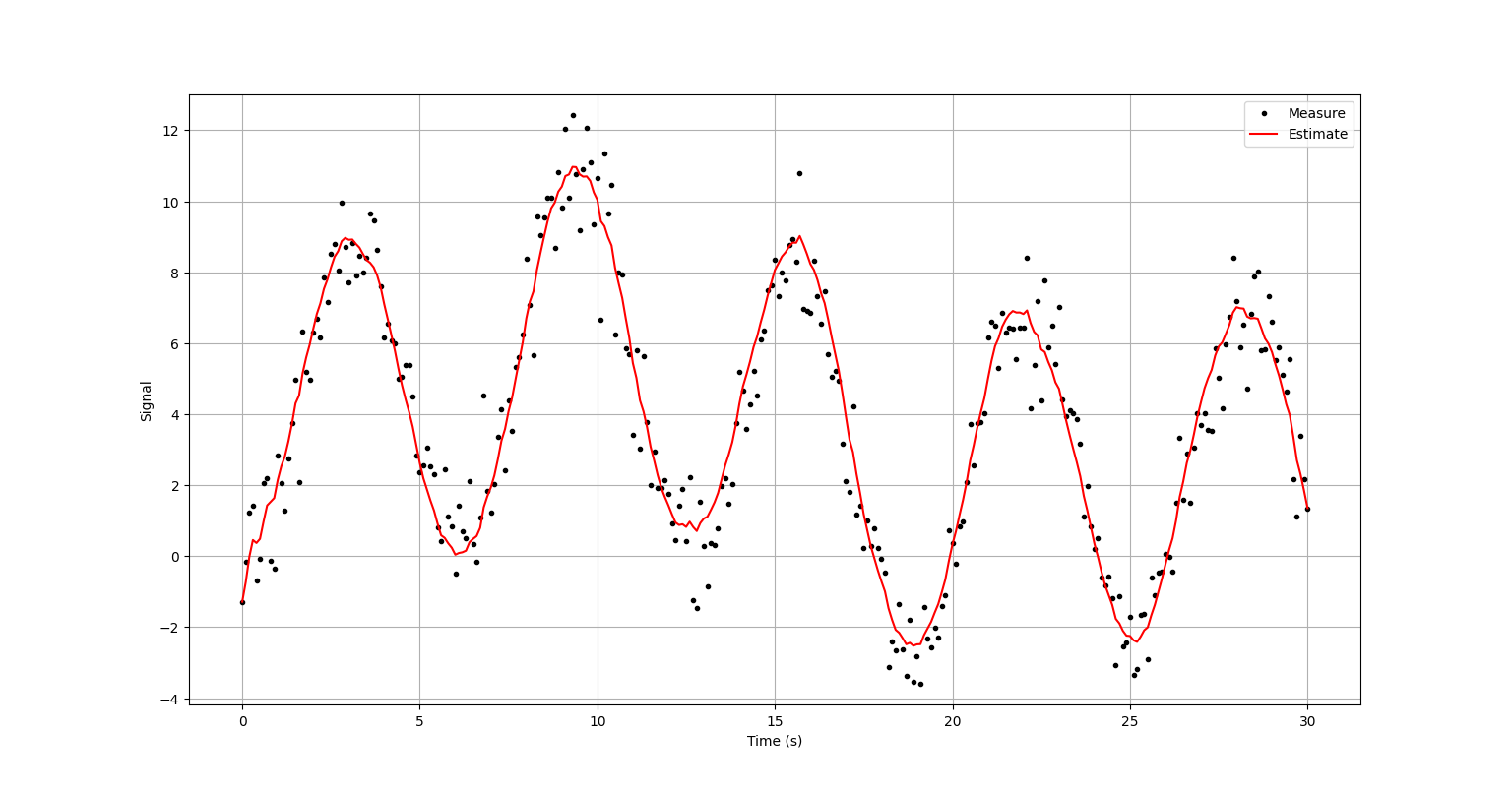
-> 상태변수: 위치 x\_k

-> 입력: f(t) = 엑셀에 주어진 u 값 (time step = 0.1)

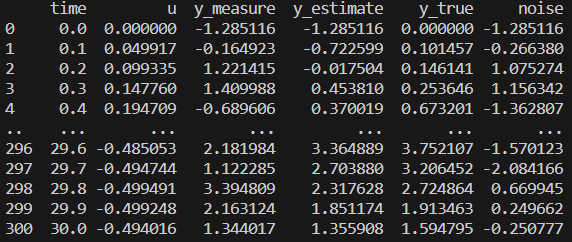
-> 출력: y\_k 이고, 이걸로 Kalman Filter 에서 사용하는 상태공간 방정식의 형태로 식을 세우면,

x\_(k+1)=x\_k + (del\_t/m)u, y\_k=dx\_k/dt 의 두 식이 나온다. 가해진 힘에 대한 속도를 측정하는 모델에서 각 A,B,C,D 는 상수로 표현할 수 있다. (A=1, B=time\_step/m=1, C=1, D=0)

이를 적용하면 python에 prediction, update 과정을 적용한 Kalman filter를 구현할 수 있다.

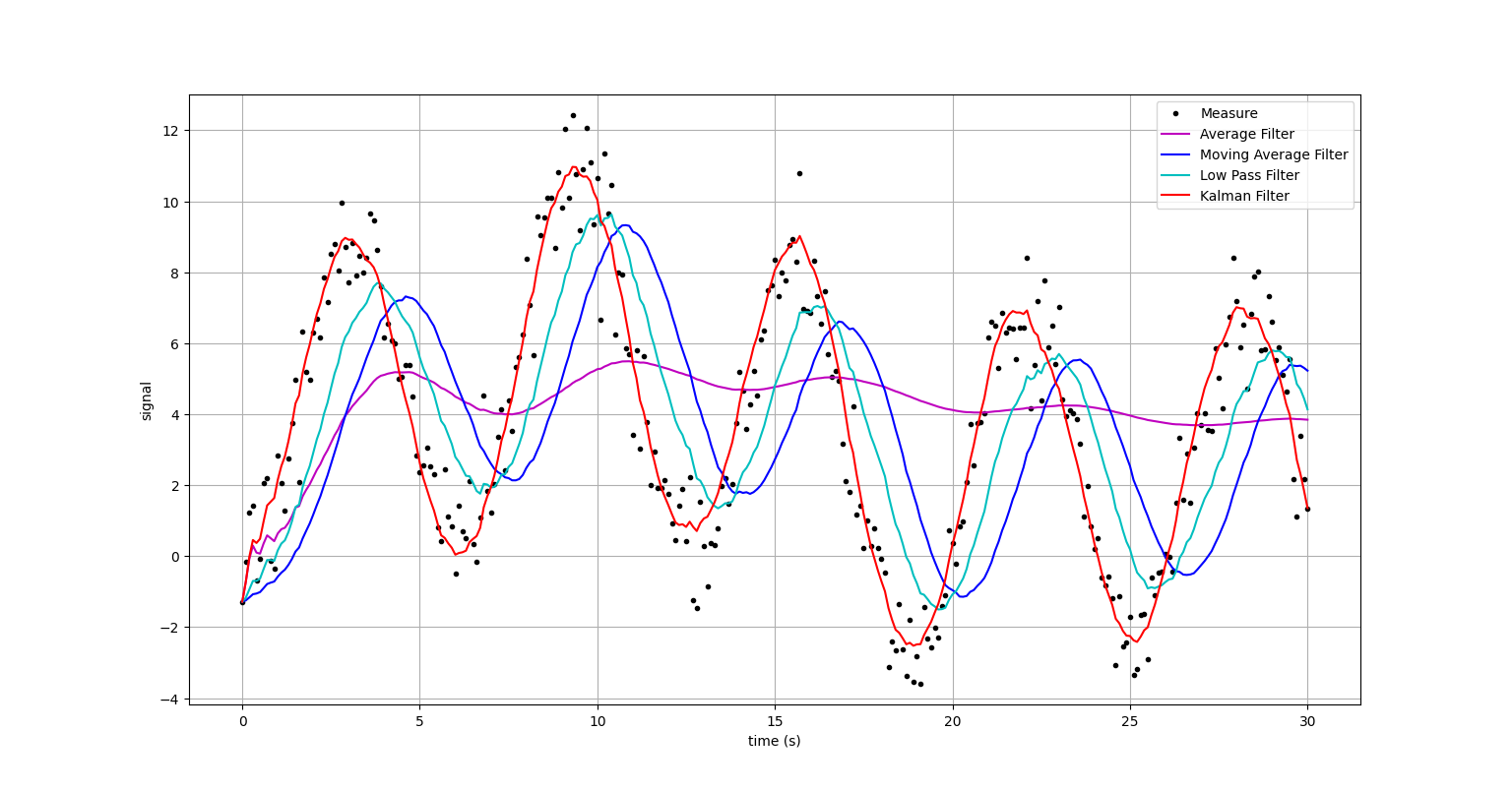


어찌어찌… 된 것 같기는 하다. 뒤에서 다른 필터들과 비교해 보자.



**5. Comparing Filters**

- 동일한 데이터에 지금까지의 4개의 필터를 모두 사용해서 결과를 비교해 보겠다.



- 필터는 예상하는 바와 같이 Kalman > LowPass > Moving Average > Average 순서로 좋은 성능을 보인다.

- 이거 하면서 numpy 라이브러리에 좋은 메소드를 하나 알았다. append(~~~.item()) 이라고 마지막에 item()을 붙여주면, 아무리 2차원 배열이라 해도 원소가 하나인 경우 배열의 원소을 전부scalar 값으로 바꿔준다.

=> plot 하는데 array가 2차원 배열이라 계속 오류가 나서 서칭하다가 찾았다 ㅎㅎㅎ 고마워요 chatGPT~

- 이제는 가장 성능이 좋은 Kalman Filter를 튜닝하여 필터의 성능을 올리는 것에 집중해봐야겠다.

**6. Tuning Kalman Filters**

- 4번 문제와 동일한 모델에서, 인풋과 노이즈를 튜닝하여 성능을 비교해보자

- 재밌는 결과가 나왔다. 오차 공분산 Px 값은 고정으로 두고 노이즈의 표준편차 Q, R만을 비교했을 때, 같은 모양의 그래프를 같은 색으로 표현해 보았다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Q** |  |  |  |  |  |
| **R** |  | **0.0001** | **0.001** | **0.01** | **0.1** | **1.0** |
|  | **0.01** |  |  |  |  |  |
|  | **0.1** |  |  |  |  |  |
|  | **1.0** |  |  |  |  |  |
|  | **10.0** |  |  |  |  |  |
|  | **100.0** |  |  |  |  |  |

- Px=10.0 이다.

- Q=0.01, R=1.0은 초기에 주어진 값으로 가장 일반적인 값이라 생각되어 기준으로 삼았다.

- 그래프가 가지는 특성은 아래와 같다.

1) 빨간 그래프를 1번, 검은 영역을 9번으로 하여 순서대로 번호를 부여했다.

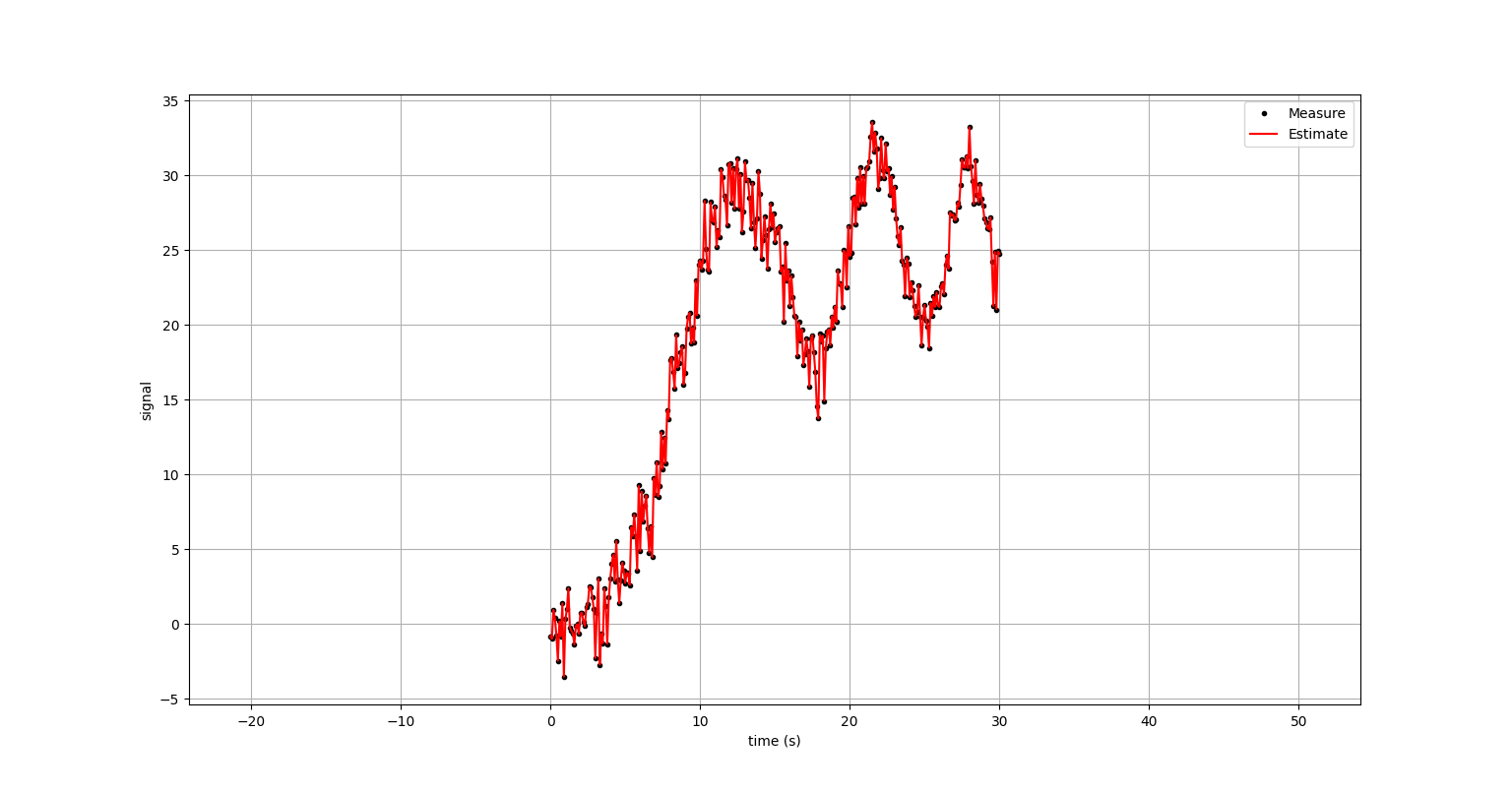
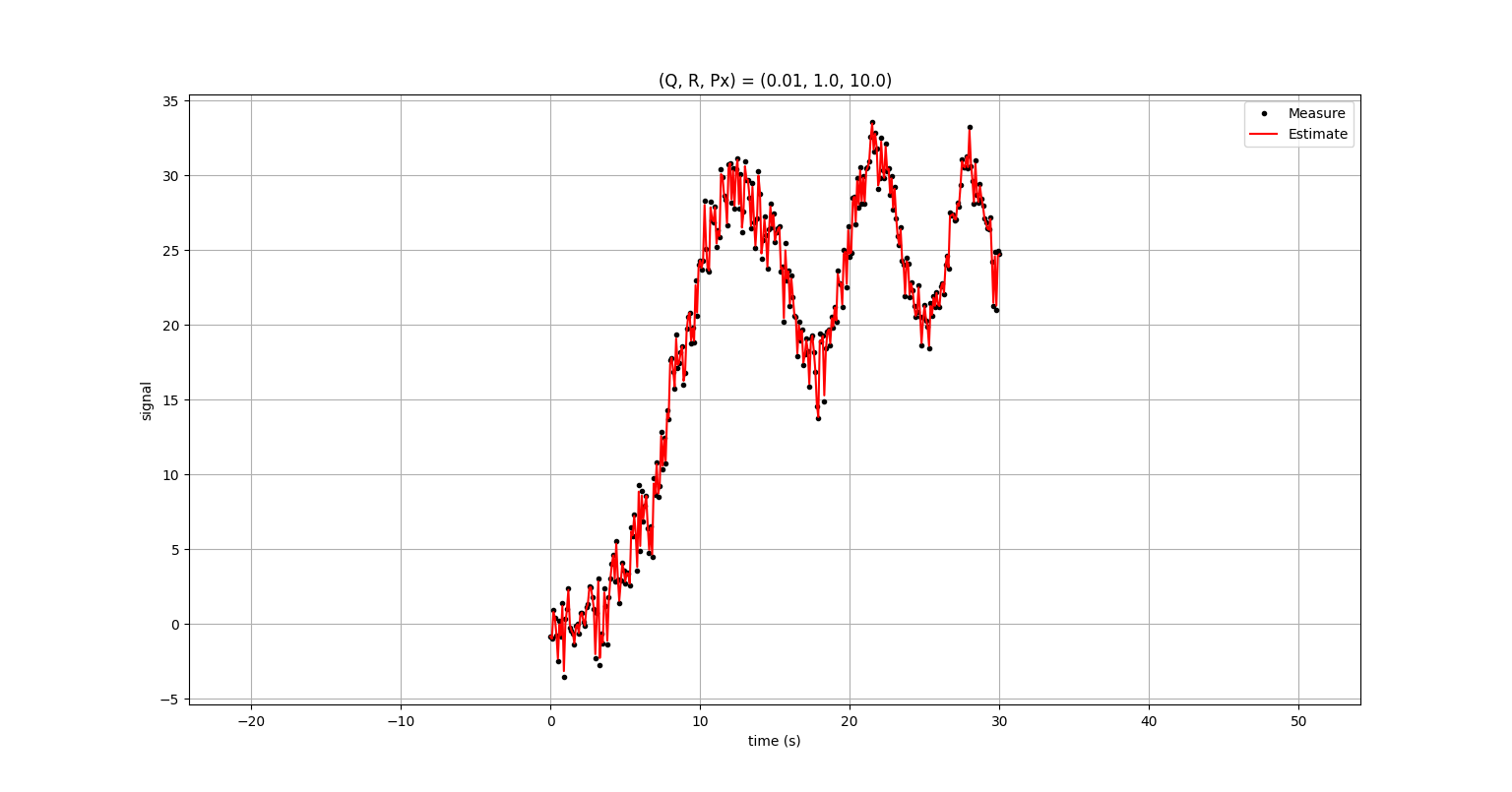
2) 1번에 가까운 그래프일수록 측정값(y\_measure)에 가까운 결과를 보였다. 즉, 노이즈 제거가 잘 되지 않아 주파수가 높은 형태의 변동이 심한 그래프를 보였다.

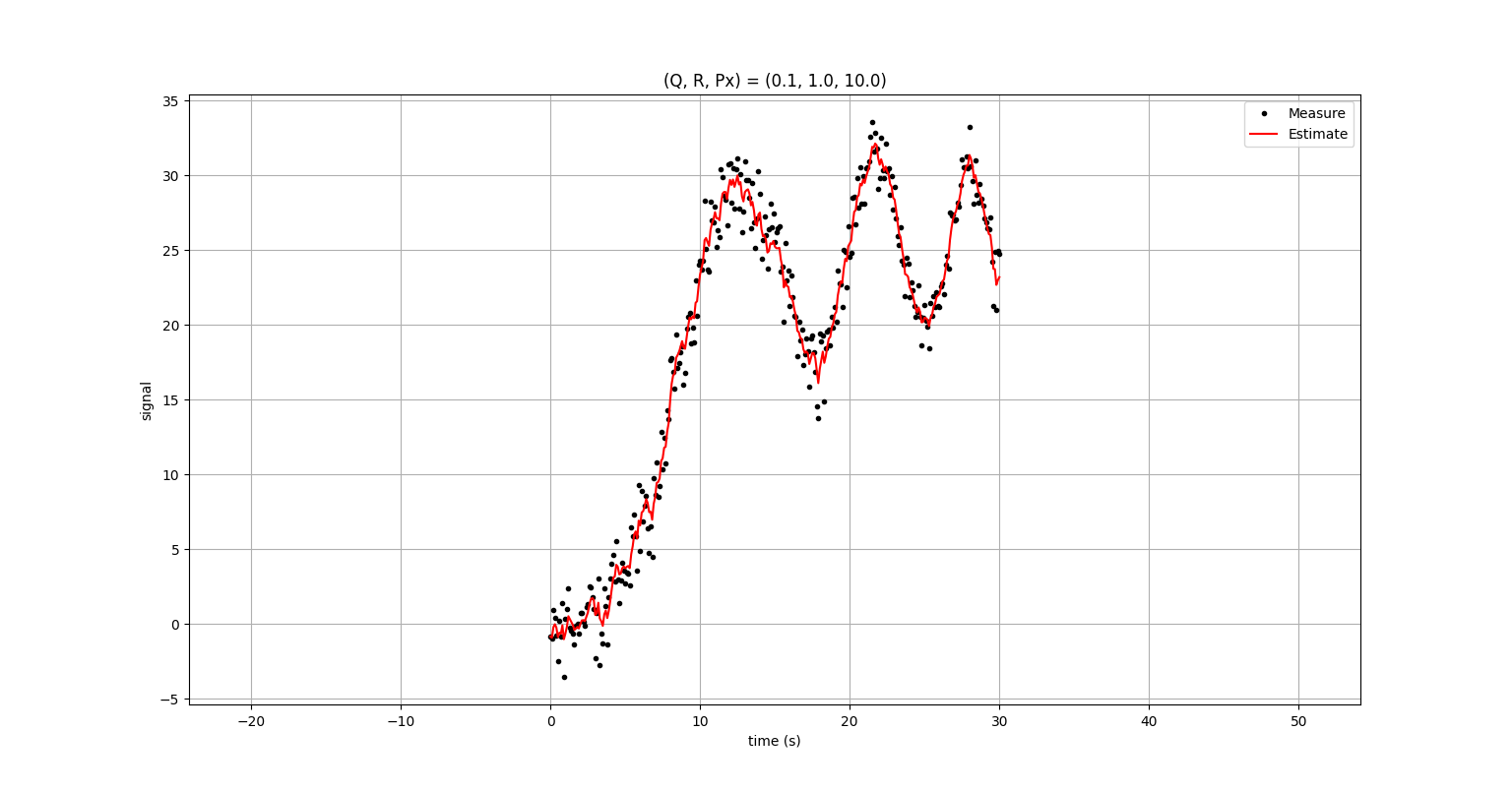
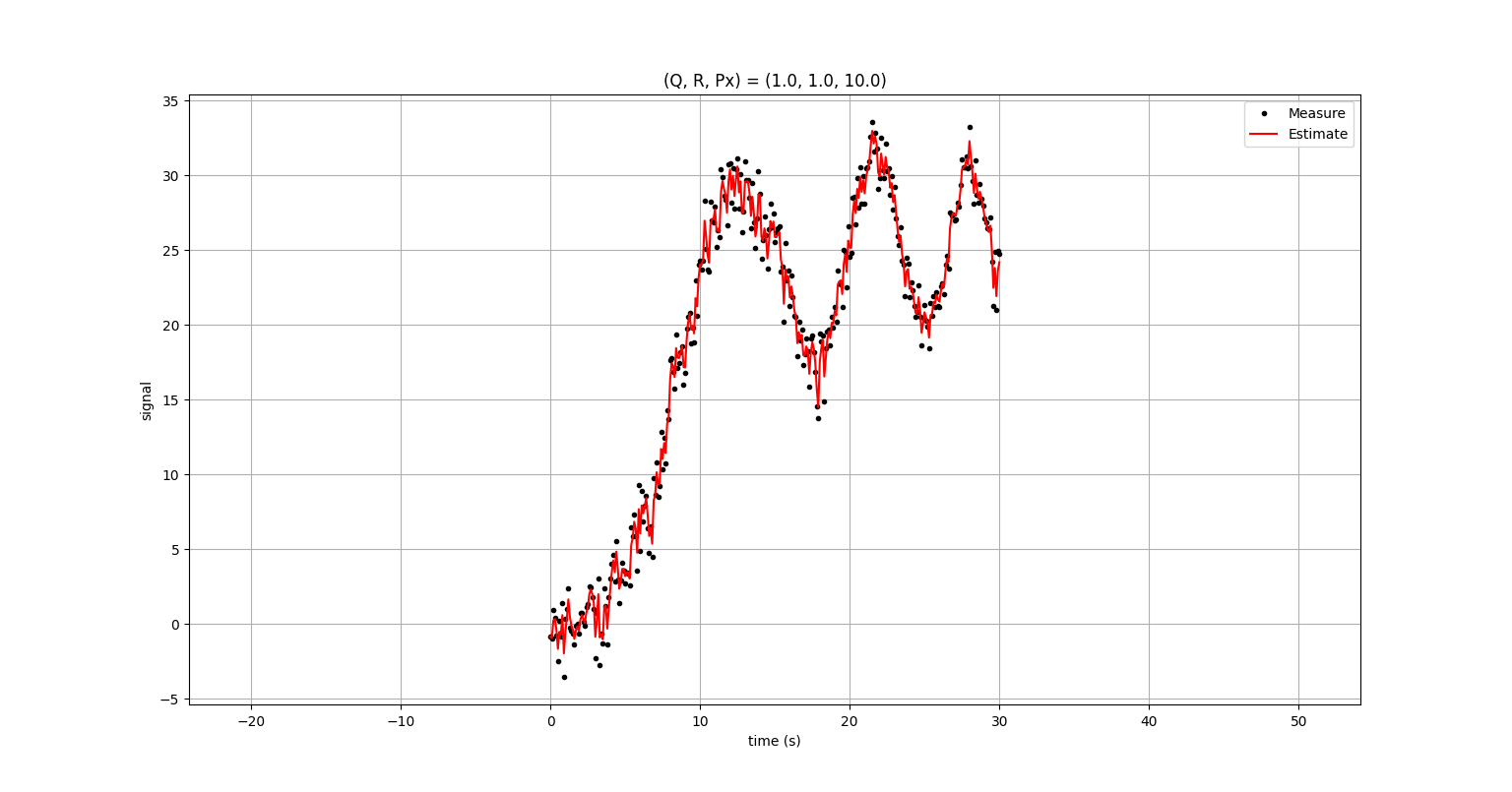
3) 반대로 9번에 가까운 그래프일수록 예측치에 가까운 결과를 보엿다. 즉, 노이즈 제거가 잘 되어 주파수가 낮은 형태의 완만한 그래프를 보였다.

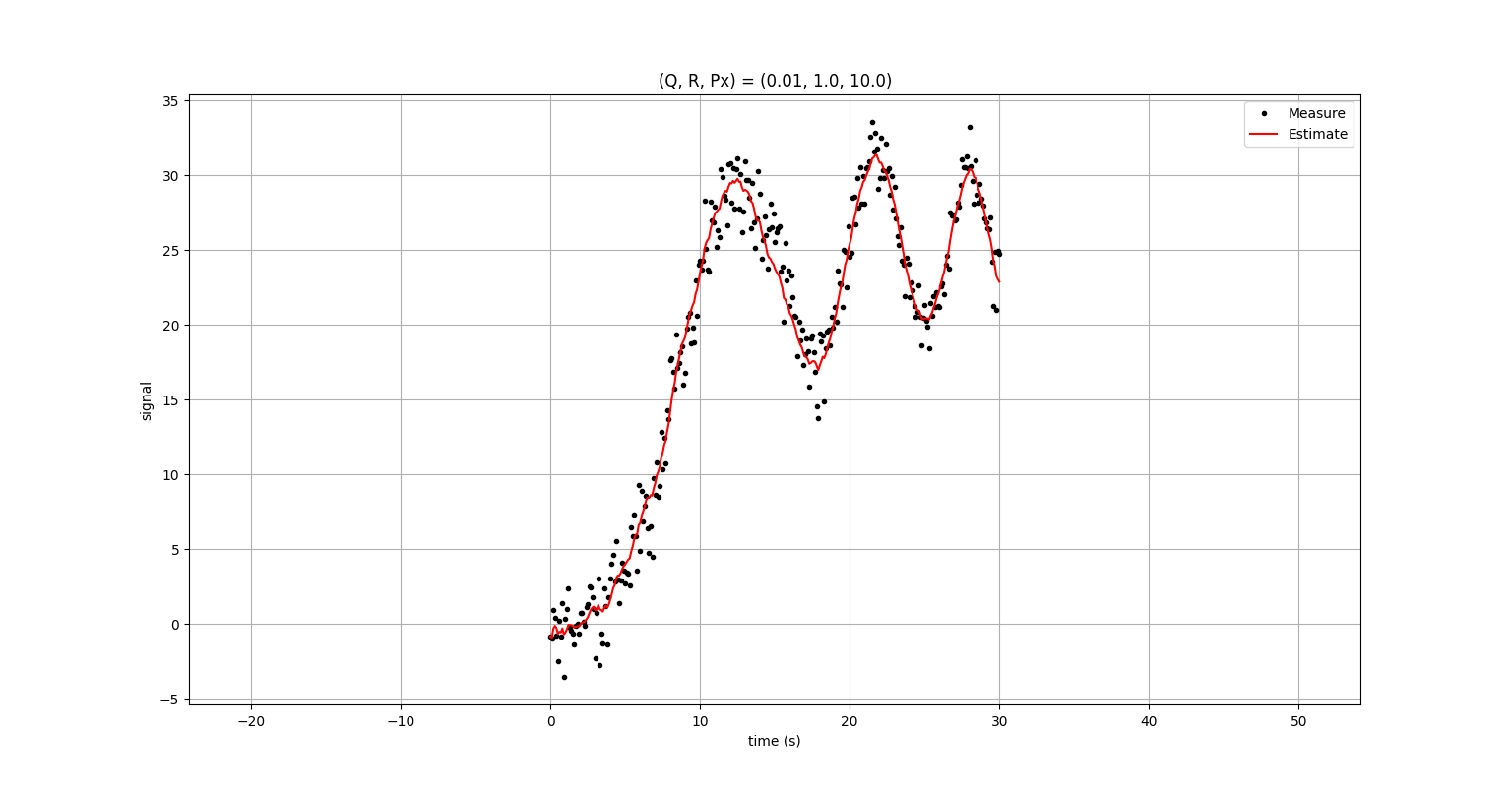
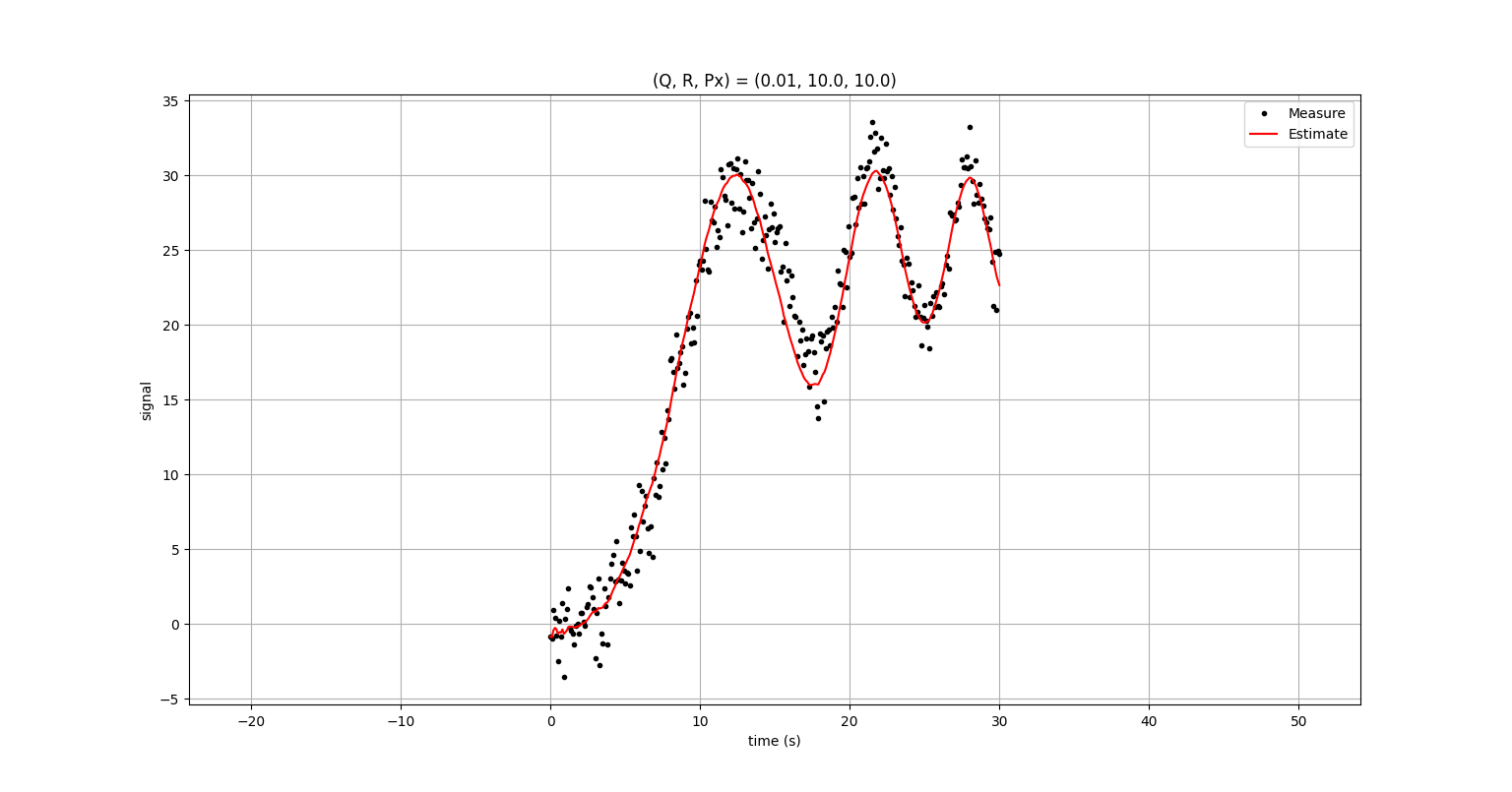
4) 이를 통해 R과 Q가 같은 비율로 커지거나 동시에 작아진다면 결국 같은 성능의 필터임을 직관적으로 알 수 있다.

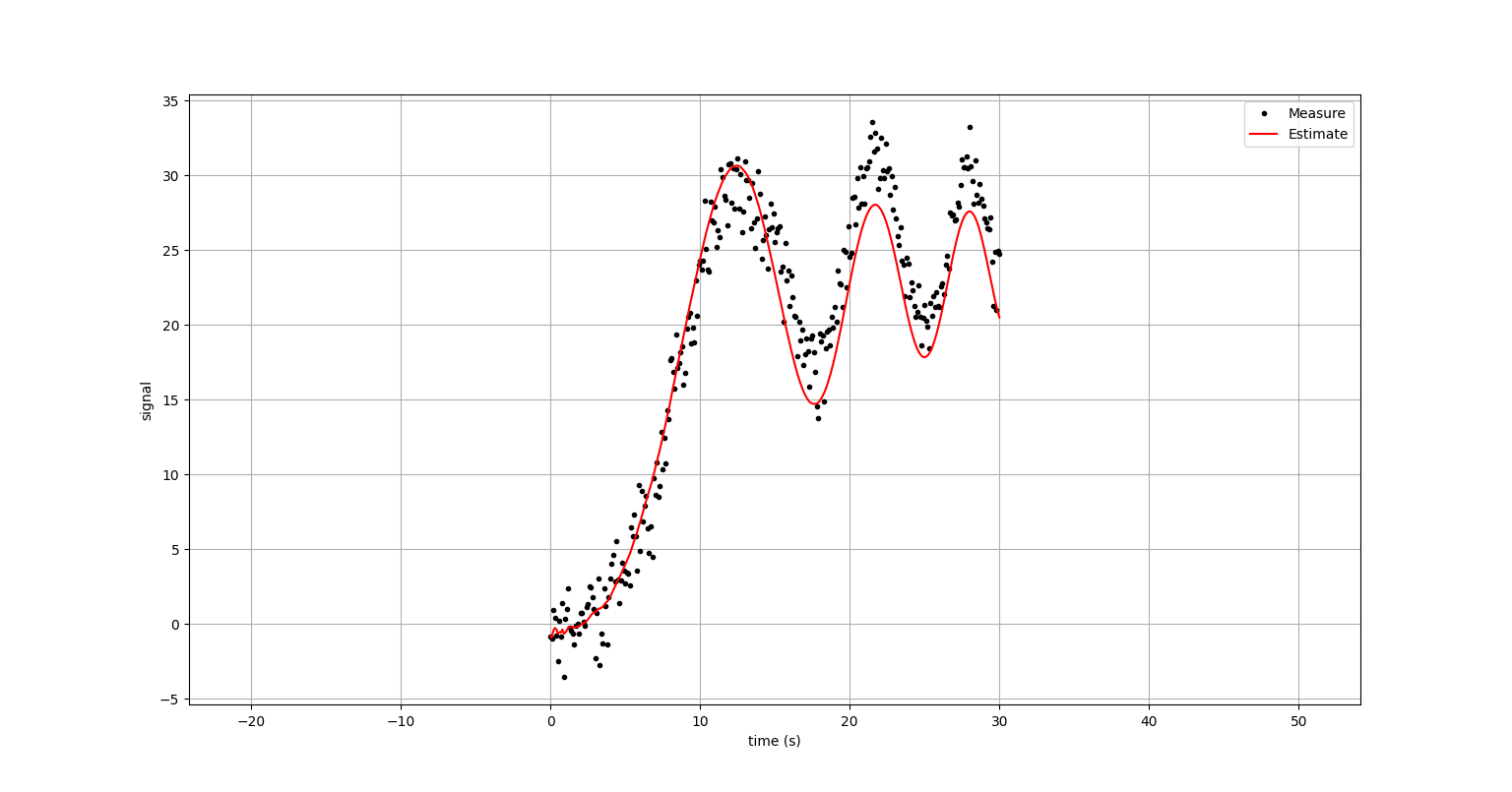
5) 필터의 경우의 수가 25개에서 9개로 줄어 원하는 성능의 필터를 효율적으로 찾을 수 있게 되었다.

- 아래 1번부터 9번까지의 그래프를 순서대로 쭉 나열해 보겠다. 한눈에 비교가 될 것이다.

**1)** **2)**

**3) 4)**

**5) 6)**

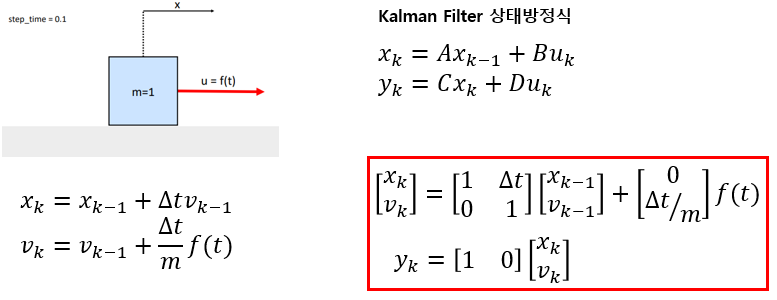
**텍스트, 라인, 도표, 그래프이(가) 표시된 사진

자동 생성된 설명7) 8)**

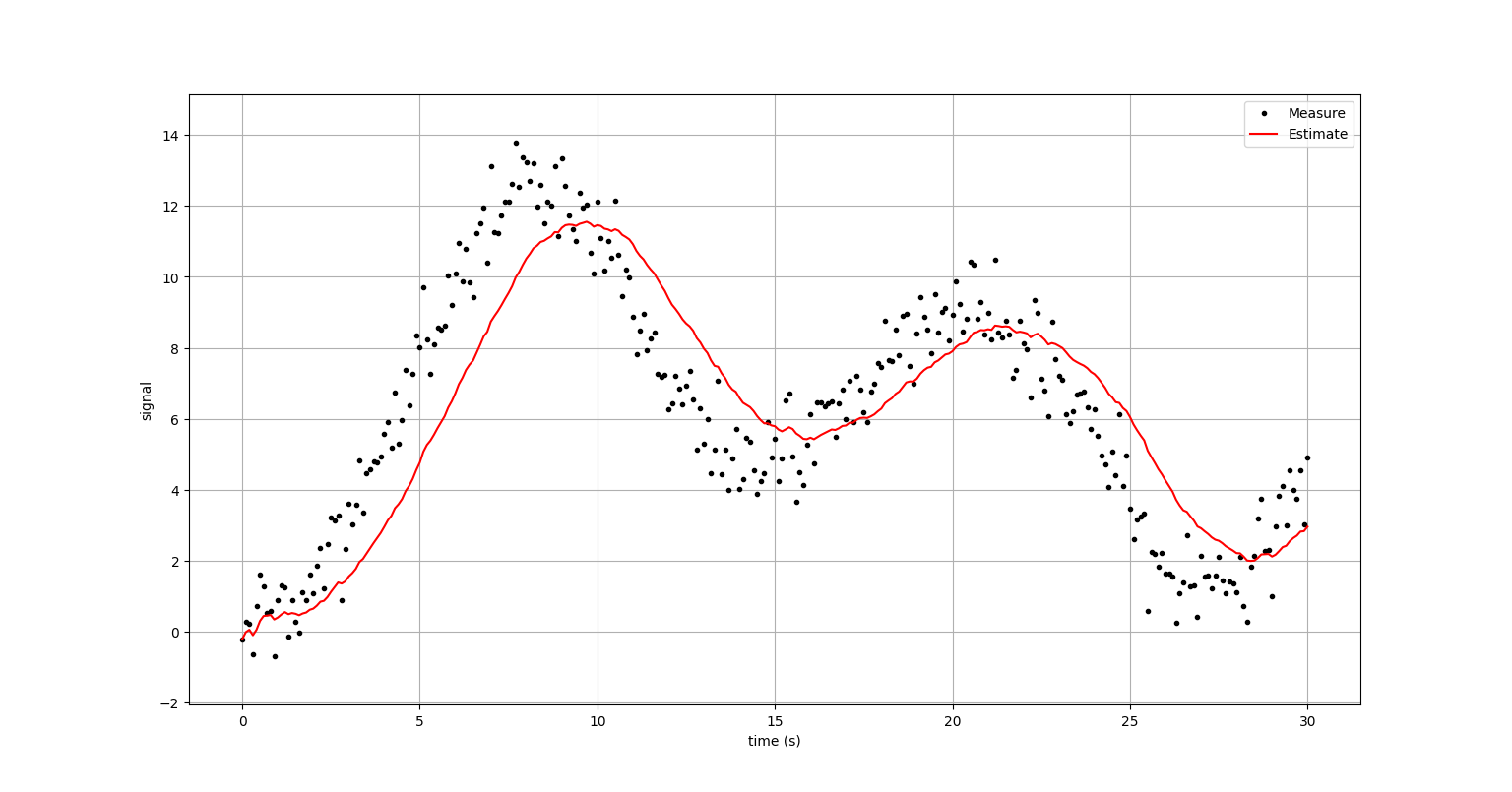
- 결론적으로 목적에 맞게, 실제 측정값에서 크게 벗어나지 않고 안정적인 결과를 보이는 필터를 선택하면 될 것 같다.

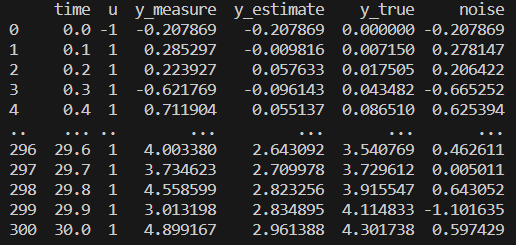
**7. Kalman Filter (State Space Equation,1)**

- 물체의 위치를 측정하는 모델에서 상태방정식을 다시 세우고 필터에 적용해 보자



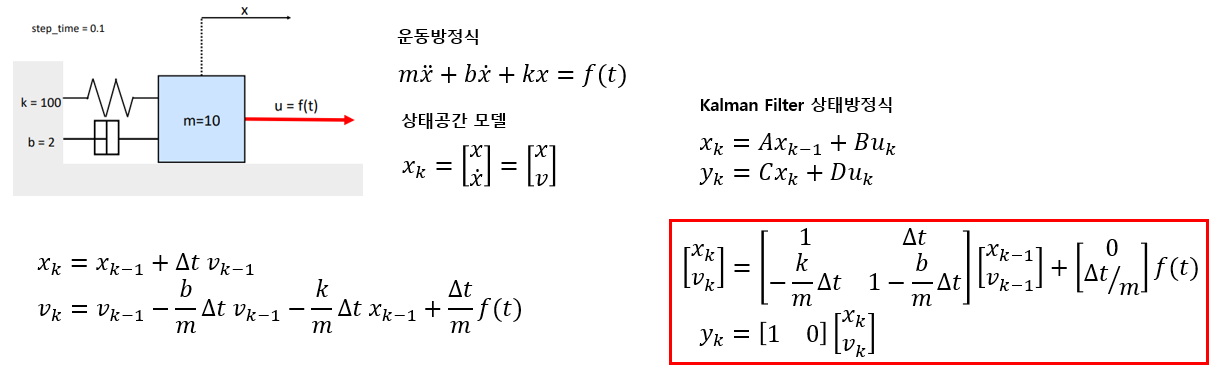
위 모델을 기반으로 필터를 적용시키면 아래와 같은 결과가 나온다.

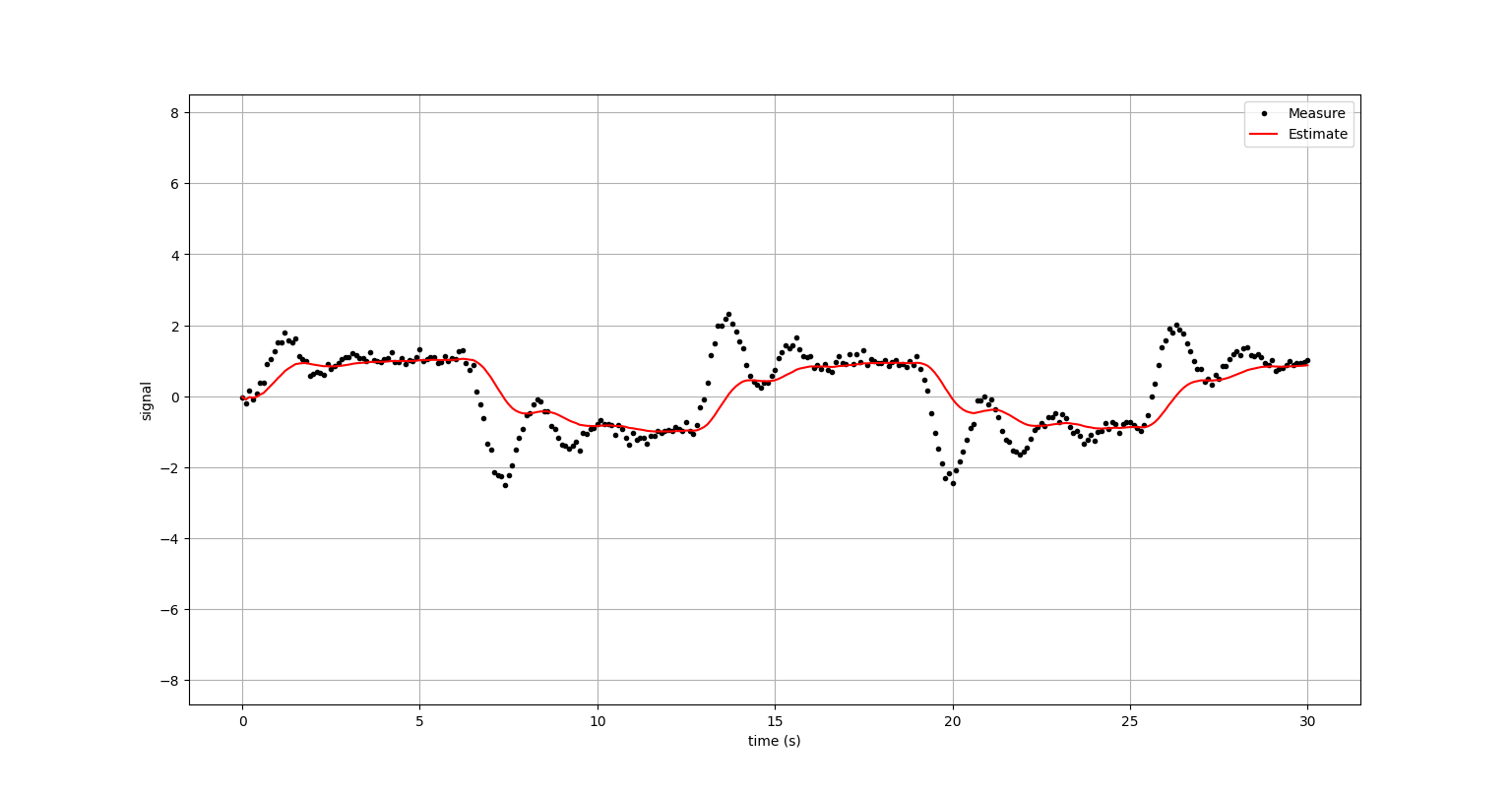




**8. Kalman Filter (State Space Equation,2)**

- 위치(x)를 측정하는 Spring-Damper 시스템의 모델식을 세우고 칼만 필터를 적용시켜 보자

 위 모델을 기반으로 필터를 적용시키면 아래와 같은 결과가 나온다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명