ANSWERS

U6.3 The Kalman gain

State estimation · University · about 35 min

What this lesson is about

Where the weighting comes from: two variances, one line of arithmetic, no taste involved.

- The scalar filter, in full
- Reading the gain
- The steady state gain

Questions

6 marks in all

1. A scalar Kalman filter has p = 100 and R = 25. What is the Kalman gain k? [1 mark]

> **Answer:** 0.8 (accept within 0.001). k = p / (p + R) = 100 / 125 = 0.8. The estimate is far less certain than the measurement, so it takes most of it.

2. One full predict and correct of the scalar filter. What does it print? [1 mark]

```
x, p = 0.0, 100.0
v, dt, Q, R = 20.0, 0.1, 0.5, 25.0
measured = 5.0
x = x + v * dt
p = p + Q
k = p / (p + R)
x = x + k * (measured - x)
p = (1 - k) * p
print(round(k, 3), round(x, 3), round(p, 3))
```

> **Answer:**
>
> ```
> 0.801 4.402 20.02
> ```
>
> Predict: x = 2, p = 100.5. Then k = 100.5 / 125.5 = 0.801, x = 2 + 0.801 x 3 = 4.402, and p = (1 - k) x 100.5 = 20.02. One measurement cut the variance by a factor of five.

3. The steady state gain from the lesson's formula, checked by running the filter. What does it print? [1 mark]

```
Q, R = 0.5, 25.0
p = (Q + (Q ** 2 + 4 * Q * R) ** 0.5) / 2
print(round(p / (p + R), 3))

p = 100.0
for i in range(200):
    p += Q
    k = p / (p + R)
    p = (1 - k) * p
print(round(k, 3))
```

Answer:

```
0.132
0.132
```

The formula gives p = (0.5 + sqrt(50.25)) / 2 = 3.79, so k = 3.79 / 28.79 = 0.132, and the iterated filter settles on the same value.

4. Q stays at 0.5 and R is doubled from 25 to 50. What happens to the steady state gain? [1 mark]

- ☐ **A** It falls, because the measurement is now trusted less
- ☐ **B** It rises, because there is more noise to correct
- ☐ **C** It halves exactly
- ☐ **D** It is unchanged

Answer: A. R is how noisy the measurement is. A larger R makes k = p / (p + R) smaller, so the filter moves less towards each measurement. It does not exactly halve, because the settled p changes too.

5. The filter is started with p = 0.01 while the estimate is in fact badly wrong. What happens for the first second or so? [1 mark]

- ☐ **A** The gain is near 0, so it largely ignores the measurements
- ☐ **B** The gain is near 1, so it jumps to each measurement
- ☐ **C** It diverges immediately
- ☐ **D** Nothing different from starting at p = 100

Answer: A. A tiny p claims the estimate is almost certain, so k is tiny: 2 percent on the first tick with Q = 0.5 and R = 25. Q grows p every tick, so the gain reaches 12 percent within a second and the filter starts to listen.

6. Once the gain has settled, which statements are true? [1 mark]

Tick every answer that is true.

- ☐ **A** The filter behaves like an exponential filter with alpha equal to the settled gain
- ☐ **B** The gain can be computed once and hard-coded, skipping the variance arithmetic
- ☐ **C** The filter has stopped using the measurements
- ☐ **D** The gain is always 0.5 at steady state

Answer: A, B. With constant Q and R, k converges to a fixed value, so the update is exactly an exponential filter with that alpha. Many shipped systems hard-code it.

The task: a Kalman filter in one dimension

Run the scalar filter while driving at the wall. Plot `measured`, `estimate` and `gain`, with the gain as a percentage so that it shows. Print `my y:` and `final gain:`, the gain itself as a number between 0 and 1. No `position()`.

```
from bugbot import *
connect()

DT, START_GAP = 0.1, 110.0
R = 25.0
Q = 0.5
est, p = 0.0, 100.0
```

The hint students can ask for: Carry a variance p as well as the estimate. Predict: p += Q. Correct: k = p / (p + R), then estimate += k * (measured - estimate), then p = (1 - k) * p. R is the measurement's variance, which is sigma squared, and sigma here is about 5 cm.

A solution

```
from bugbot import *
connect()

DT = 0.1
START_GAP = 110.0
R = 25.0            # the depth sensor's variance: sigma 5 cm, squared
Q = 0.5             # how much the model can be wrong by in one tick

est = 0.0
p = 100.0           # we start quite unsure
k = 0.0

forward(60)
for i in range(80):
    est += flow()[1] * DT                    # predict the state
    p += Q                                   # and the uncertainty grows
    measured = START_GAP - distance()
    k = p / (p + R)                          # the Kalman gain
    est += k * (measured - est)              # correct
    p = (1 - k) * p                          # and the uncertainty shrinks
    plot("measured", measured)
    plot("estimate", est)
    plot("gain", k)
    if est > 62:
        stop()
    wait(DT)
stop()
print("my y:", round(est, 1))
print("final gain:", round(k, 3))
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.