ANSWERS

U12.2 Fitting a model to data

Learning, and the capstone · University · about 35 min

What this lesson is about

Least squares in plain Python, and a robot that drives by the model it fitted to itself.

- The problem
- Collecting data on a robot that moves
- The dead band, and what a model is for
- Using the fit

Questions

7 marks in all

1. What does this least squares fit of speed against command print? [1 mark]

```
data = [(20, 3.1), (40, 7.4), (60, 11.2), (80, 15.3), (100, 19.0)]
n = len(data)
sx = sum(c for c, v in data)
sv = sum(v for c, v in data)
sxx = sum(c * c for c, v in data)
sxv = sum(c * v for c, v in data)
slope = (n * sxv - sx * sv) / (n * sxx - sx * sx)
intercept = (sv - slope * sx) / n
print(round(slope, 4), round(intercept, 3))
```

Answer:

```
0.1985 -0.71
```

Six sums and a division give a slope of 0.1985 cm/s per unit of command and an intercept of -0.71 cm/s. The negative intercept is a hint of the dead band below the data.

2. A fit gives v = 0.2 c - 0.8. How many seconds should the robot drive at command 60 to cover 80 cm open loop? Give 2 decimal places. [1 mark]

Answer: 7.14 (accept within 0.01). v = 0.2 x 60 - 0.8 = 11.2 cm/s, and 80 / 11.2 = 7.14 s.

3. Speeds are read 0.2 s after each command change. Why is that worse than random noise? [1 mark]

- ☐ **A** Every reading is taken on the ramp and is low by the same fraction, so the slope comes out quietly wrong
- ☐ **B** The readings become too noisy to average
- ☐ **C** The intercept becomes exactly zero
- ☐ **D** The fit fails to converge

Answer: A. With a 0.25 s lag the speed has not settled. A consistent bias does not average away; it moves the answer.

4. What does taking (forward - backward) / 2 at each command achieve? [1 mark]

- ☐ **A** It cancels any fixed offset in the sensor and reduces the noise, and the robot ends where it began
- ☐ **B** It doubles the speed range of the sweep
- ☐ **C** It removes the drive's dead band
- ☐ **D** It corrects the flow sensor's scale factor

> **Answer:** A. An offset adds to both directions and subtracts out. The flow sensor's scale error is multiplicative and stays in the model.

5. A straight line is fitted to commands from 5 to 100, including the region below 15 where the drive does not move. What is the sensible response? [1 mark]

- ☐ **A** Restrict the fit to where the model applies and state that domain, or use a model with a dead band in it
- ☐ **B** Keep all the points, because more data always gives a better fit
- ☐ **C** Force the intercept to zero
- ☐ **D** Fit a degree 5 polynomial to follow the bend

> **Answer:** A. Across the dead band the intercept is nonsense. v = a max(0, c - d) captures it, at the cost of a search, because it is not linear in d.

6. A proportional heading correction asks for 8 percent turn during the sweep, and the robot slowly turns anyway. Why? [1 mark]

- ☐ **A** 8 percent is below the 15 percent dead band, so the correction does nothing; it needs a minimum magnitude above it
- ☐ **B** The gyro is not being read
- ☐ **C** The correction has the wrong sign
- ☐ **D** Heading has no effect on flow readings

> **Answer:** A. The controller and the plot look right while the drive ignores the command. The lesson clamps the turn command to at least 18.

7. Driving for D / v open loop lands close to D despite the 0.25 s lag. Why? [1 mark]

- ☐ **A** The robot loses about v tau of travel while speeding up and gains about the same coasting after the stop
- ☐ **B** The lag only affects turning
- ☐ **C** The flow sensor corrects the distance during the run
- ☐ **D** The fit's intercept absorbs the lag

> **Answer:** A. The two effects cancel, so a good model gives accurate travel without a single sensor reading during the run.

The task: fit the drive

Sweep the command, fit speed against command by least squares, print `slope:` and `intercept:`, then use the fit to drive 80 cm open loop and stop there.

```
from bugbot import *
connect()

DT = 0.1
COMMANDS = [20, 30, 40, 50, 60, 70, 80, 90, 100]
TARGET = 80.0
```

The hint students can ask for: Sweep the command, and at each one let the robot settle before reading flow(). Run each command forwards and then backwards so the robot stays where it started, and hold the heading square throughout, remembering that a correction below the drive's dead band does nothing. Fit speed against command by least squares, then work out how long to drive at one command to cover 80 cm.

A solution

```
from bugbot import *
connect()

DT = 0.1
COMMANDS = [20, 30, 40, 50, 60, 70, 80, 90, 100]
TARGET = 80.0
CRUISE = 70

def turn_cmd():
    """Hold the robot square: a sweep takes half a minute, and a robot that turns during it
fits nonsense."""
    err = (imu()[0] + 180) % 360 - 180
    if abs(err) < 2.0:
        return 0
    cmd = max(18.0, min(30.0, abs(1.5 * err)))     # below 15 the drive does nothing at all
    return -cmd if err > 0 else cmd

def hold(cmd, ticks, collect=False):
    vs = []
    for i in range(ticks):
        drive(cmd, 0, turn_cmd())
        if collect:
            vs.append(flow()[1])
        wait(DT)
    return vs

def speed_at(cmd):
    """Mean forward speed at this command, once the motors have come up."""
    hold(cmd, 12)                  # the command reverses between samples, so give it five
time constants
    vs = hold(cmd, 3, True)
    return sum(vs) / len(vs)

# every command forwards and then backwards, so the robot ends where it began
data = []
for c in COMMANDS:
    out = speed_at(c)
    back = speed_at(-c)
```

```
    data.append((c, (out - back) / 2.0))
stop()
wait(0.3)

n = len(data)
sx = sum(c for c, v in data)
sv = sum(v for c, v in data)
sxx = sum(c * c for c, v in data)
sxv = sum(c * v for c, v in data)
slope = (n * sxv - sx * sv) / (n * sxx - sx * sx)
intercept = (sv - slope * sx) / n
print("slope:", round(slope, 4))
print("intercept:", round(intercept, 3))

# drive 80 cm open loop, on the model and nothing else
predicted = slope * CRUISE + intercept
seconds = TARGET / predicted
print("predicted speed:", round(predicted, 2), "so", round(seconds, 2), "seconds")
hold(CRUISE, int(round(seconds / DT)))
stop()
wait(0.5)
```

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