ANSWERS

U7.5 Resampling

Localisation · University · about 30 min

What this lesson is about

Keeping the good guesses without losing the diversity that lets the filter recover.

- Low variance resampling
- Particle deprivation
- Watching it work

Questions

6 marks in all

1. What does this program print? [1 mark]

```
particles = ["A", "B", "C", "D"]
weights = [0.05, 0.5, 0.3, 0.15]
N = 4
step = 1.0 / N
r = 0.12
c = weights[0]
i = 0
fresh = []
for m in range(N):
    u = r + m * step
    while u > c and i < N - 1:
        i += 1
        c += weights[i]
    fresh.append(particles[i])
print(" ".join(fresh))
```

Answer:

```
B B C D
```

The pointers are 0.12, 0.37, 0.62 and 0.87 against cumulative weights 0.05, 0.55, 0.85, 1.0. B gets two copies (0.5 x 4 = 2), C and D one each, and A, with 0.05 x 4 = 0.2, gets none.

2. With low variance resampling and N = 500, a particle has weight 0.013. How many copies does it get? [1 mark]

- ☐ **A** 6 or 7
- ☐ **B** Exactly 7
- ☐ **C** Any number from 0 to 500, depending on luck
- ☐ **D** 13

Answer: A. w x N = 6.5, and low variance resampling gives either floor(wN) or ceil(wN) copies, so 6 or 7. Naive independent draws could give 0 or 15.

3. After many rounds of resampling without jitter, the filter is confidently and permanently wrong. What has happened? [1 mark]

- ☐ **A** Particle deprivation: every particle is an exact copy of one ancestor, so nothing is left elsewhere to rescue it
- ☐ **B** The weights have underflowed to zero and the floor has taken over
- ☐ **C** The motion noise was too large, so the cloud spread off the mat
- ☐ **D** The effective sample size has grown above N

> **Answer:** A. Copies are exact, so resampling spends diversity. Once the cloud is a single point, a wrong point can never be corrected.

4. Which of these defend against particle deprivation? [1 mark]

Tick every answer that is true.

- ☐ **A** Add a small amount of jitter to each copy
- ☐ **B** Resample only when neff falls below N/2
- ☐ **C** Inject a few particles scattered over the whole map each tick
- ☐ **D** Resample on every tick whatever neff is
- ☐ **E** Use a smaller sigma in the measurement model

> **Answer:** A, B, C. Jitter, resampling only when needed and injecting scattered particles all keep diversity. Resampling every tick spends it faster, and a smaller sigma makes the collapse worse.

5. Put the steps of low variance resampling in order. [1 mark]

Number the lines 1 to 5 to put them in the right order.

___ `Set c to the first weight and i to 0`
___ `For each m, set the pointer u = r + m * step`
___ `Copy particles[i], plus a little jitter, into the new set`
___ `Set step = 1/N and draw a single number r between 0 and step`
___ `While u > c, move i on and add weights[i] to c`

> **Answer:**
>
> ```
> Set step = 1/N and draw a single number r between 0 and step
> Set c to the first weight and i to 0
> For each m, set the pointer u = r + m * step
> While u > c, move i on and add weights[i] to c
> Copy particles[i], plus a little jitter, into the new set
> ```
>
> One draw, then equal strides through the cumulative weights. Because the pointer only moves forwards the whole pass is O(N).

6. Why is low variance resampling preferred to drawing N independent samples? [1 mark]

- ☐ **A** It is less noisy, never gives zero copies to a particle that deserves one, and runs in O(N)
- ☐ **B** It needs no random numbers at all, so it is deterministic
- ☐ **C** It keeps the old weights, so no information is lost
- ☐ **D** It guarantees the resampled cloud has no duplicate particles

> **Answer:** A. It uses one random number, not none, and it does produce duplicates, which is why jitter is still needed.

The task: resample

Weight a scattered cloud against one reading, then resample it. Print `before:` and `after:`, the effective sample size each side, and `spread:`, the standard deviation of the resampled cloud.

```
from bugbot import *
import math, random
connect()

N, SIGMA = 500, 3.0
particles = [random.uniform(0, 200) for i in range(N)]
```

The hint students can ask for: Low variance resampling: one random start, then step through the cumulative weights in equal strides. Every particle comes out with weight 1/N, so the effective sample size is back to N, and the cloud is concentrated where the weight was.

A solution

```
from bugbot import *
import math
import random
connect()

N = 500
SIGMA = 3.0
particles = [random.uniform(0, 200) for i in range(N)]

readings = []
for i in range(10):
    readings.append(distance())
    wait(0.1)
measured = sum(readings) / len(readings)

weights = []
for y in particles:
    d = (200 - y) - measured
    weights.append(math.exp(-d * d / (2 * SIGMA * SIGMA)) + 1e-12)
total = sum(weights)
weights = [w / total for w in weights]
print("before:", round(1.0 / sum(w * w for w in weights), 1))

# low variance resampling: one random start, then equal strides through the cumulative
weight
step = 1.0 / N
r = random.uniform(0, step)
c = weights[0]
i = 0
fresh = []
for m in range(N):
    u = r + m * step
    while u > c and i < N - 1:
        i += 1
        c += weights[i]
    fresh.append(particles[i])
particles = fresh

mean = sum(particles) / N
```

```
spread = (sum((y - mean) ** 2 for y in particles) / N) ** 0.5
print("after:", round(float(N), 1))
print("spread:", round(spread, 2))
```

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