11/1/16

**Today:**
- Sampling Random #’s
- Continuous Distributions
- PDF’s
- CDF’s

**Recall:** Discrete Bayes Filter
- lets us see probability distributions of where robot could be
- Bad - large run time complexity

**Warm-up**
Assume we have a function `randint()` that returns an integer `x` with `0 \leq x \leq \text{RAND-MAX} (\text{some big number})` with equal probability.

- Computers give pseudorandom numbers

Say instead we want a random real # in `[0, 1]` with equal probability everywhere. How can we use `randint()` to do this?

→ No such thing as a real # on a computer (trick question)
  * represented as a floating point number

→ Solution: def randreal():
  
  ```python
  x = randint()
  return x/float(RAND-MAX)
  ```

Now let’s use `randreal()` to simulate flipping an unfair coin with

- \[ p(\text{Heads}) = 0.278 \]
- \[ p(\text{Tails}) = 0.722 \]
def coin_flip():
    # return 'heads' or 'tails' for unfair coin
    coin = random.real()
    if coin <= 0.278:
        return 'heads'
    else:
        return 'tails'

HARDER:

Simulate an unfair die with probabilities

def die_roll():
    x = random.real()
    if x <= 1.0/21:
        return 1
    elif x <= 3.0/21:
        return 2
    elif x <= 6.0/21:
        return 3
    elif x <= 10.0/21:
        return 4
    elif x <= 15.0/21:
        return 5
    else:
        return 6
With 100-sided die we could generate the array of possibilities in linear time. If already generated can shorten run time using Binary Search which has time complexity of \( \log_2(n) \).

**Probability Distributions**

- **Discrete**
  - \( x \in X \) (finite set)
  - \( 0 \leq p(x) \leq 1 \)
  - \( \sum x p(x) = 1 \)

- **Continuous**
  - \( x \in \mathbb{R} \)
  - \( p(x) \geq 0 \)
  - \( \int_{-\infty}^{\infty} p(x) \, dx = 1 \)

**Example:**

**Uniform Distribution on \([0, 1]\)**

\[
p(x) = \begin{cases} 
0 & \text{if } x < 0 \text{ or } x > 1 \\
1 & \text{otherwise}
\end{cases}
\]

Say I sample \( x = 0.1234567 \)

- Probability density = 1
- Probability = 0

→ If we took another 1,000 samples, NONE would equal that the probability of getting the same number is 0.

- Way to get probability: \( \int_{a}^{b} p(x) \, dx \)
- Integrates PDF between an interval
Integrating a PDF gives us an actual probability

\[ P(a \leq x \leq b) = \int_{a}^{b} P(x) \, dx \quad (\text{probability } x \text{ is in } [a, b]) \]

\[ P(0 \leq x \leq 0.278) = \int_{0}^{0.278} 1 \, dx = 0.278 \]

★ The Gaussian Distribution ★

- parametrized by mean \( \mu \) and variance \( \sigma^2 \) or standard deviation \( \sigma \)

\[ \mu \pm \sigma = 68\% \]
\[ \mu \pm 2\sigma = 95\% \]
\[ \mu \pm 3\sigma = 99\% \]

\[ p(x) = \frac{1}{\sqrt{2\pi} \sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]

Why do we like the Gaussian Model?

- easy to model
- Central Limit Theorem
  - Result of adding several random variables together will give a Gaussian Distribution

★ Cumulative Distribution Functions (CDF) ★

- given PDF \( P(x) \) define:

\[ \text{cdf}(b) = \int_{-\infty}^{b} P(x) \, dx \quad \text{area under the curve up to } b \]

\[ \text{cdf}(b) = p(x \in [-\infty, b]) = p(x \leq b) \]
The CDF is non-decreasing
- slope is pdf

CDF for uniform

$\text{cdf}(-\infty) = 0$
$\text{cdf}(\infty) = 1$
- $\text{cdf}(b) \in [0, 1] \Rightarrow$ splitting out probabilities

* Inverse CDF - (not guaranteed to be unique)

$\text{cdf}^{-1}(r) = b$ such that $\text{cdf}(b) = r$

> Useful for sampling, map uniform # in $[0, 1]$ to arbitrary pdf

$r = \text{rand\_float}()$

return $\text{cdf}^{-1}(r)$ # sample from $p(x)$
Today:
- Sampling vs. Evaluation a PPF
- Particle Filter
- Demo

Current News: Septa Strike

Random sampling $\rightarrow$ numpy
- `numpy.random.random(size)`
  - could also sample many distributions: `normal`, `uniform` (`numpy.random.normal()`)  
- `import numpy as np`
- `import matplotlib.pyplot as plt`  
  $\rightarrow$ `plt.plot(sample, 'color')`

Plotting Histogram:
```python
n, bins, patches = plt.hist(samples, bins=##)  
```

Sampling is simulating the distribution.
Evaluating is computing $p(x) = N \cdot e^{-\frac{(x-u)^2}{2\sigma^2}}$

The Particle Filter

Idea: Continuous state spaces are too big to run Bayes Filters on.
Instead of discretizing, represent distribution as collection of samples.

$P(x) \rightarrow x$  
Particle  
Bayes  
$\leftarrow$  
$N$ samples

$N$ bins
Like points on a number line.
Once again we have a motion model & a measurement model
\[ P(x' | x, u) \quad P(z | x) \]

- This time, \( x \) is continuous so motion model is a pdf (usually \( z \) is too)
  - Bayes Filter evaluates both distributions
  - Particle Filter samples the motion model, but evaluates the measurement model
  - \( \text{Sampling motion: drawing random numbers to simulate noisy motion of the robot} \)

Potential Shortcoming: - Bayes easy to guess position \( \rightarrow \) highest bin
  - Particle very good with uncertainty, but hard to tell position

* Particle Algorithm *

Start with \( n \) particles drawn from some initial distribution
\[ \mathcal{P} = \{ x_1, x_2, x_3, \ldots, x_n \} \]

1. \( \textbf{Motion Update:} \) Given control \( u \), update the set of particles to
   \[ \mathcal{P}' = \{ x'_1, x'_2, x'_3, \ldots, x'_n \} \]
   - with each \( x'_i \) sampled from \( P(x'_i | x_i, u) \)
   - add some random noise

2. \( \textbf{Measurement Update:} \) Observe \( z \) then compute for each \( x'_i \):
   \[ w_i = \eta \cdot p(z | x'_i) \]
   - weight comes from evaluating model pdf
   \[ \eta = \frac{1}{\sum p(z | x'_i)} \quad \text{so} \quad \sum w_i = 1 \]
   - So, \( w_i \) is the probability that \( x'_i \) is a “good particle.”

\( \rightarrow \) Good particles align well with sensor measurements
Now Resample:

* Assemble \( P = \sum x_i \) by sampling with replacement each \( x_i \) from \( P' \) with probability \( w_i \).

* Good particles have a better chance of making it to next level or next generation (Darwinian - Survival of the Fittest)

* Go Back to 0

Big O notation: \( O(n) \)

* Demo

\[
Z = \begin{bmatrix}
Z_1 \\
Z_2 \\
Z_3 \\
Z_4
\end{bmatrix} = \begin{bmatrix}
\text{angle to beacon 1} \\
\text{angle to beacon 2} \\
\text{angle to beacon 3} \\
\text{angle to beacon 4}
\end{bmatrix}
\]

\[
P(Z|x) = \prod_{i=1}^{4} p(Z_i|x)
\]

\[
p(Z_i|x) = \mathcal{N}(x_i - \| \pi_i - x ||^2 / 2 \sigma_z^2
\]

* Motion step \( \rightarrow \) spreads particles out

* Measurement model \( \rightarrow \) condenses particles down

- Sigma: how confident are you in the sensors: particles closer / farther if \( \sigma \) = low / high

2 beacons \( \rightarrow \) Solution: Triangulate: 3 beacons

* over time might think it's in 2 places

* will improve guess