Module 6 Exercises | Joshua Shapiro | 24 May 2017

5. Suppose Pr[heads] = p in the example above. Write down Pr[X = i], i ∈ {0,1,2,3}, in terms of p.

- **TESTING**

8. CODE: If p = 0.6, what is the probability that the first head appears on the 3rd flip? Verify your answer using Coin.java and CoinExample.java.

9. Suppose I compare two parameter values for the Geometric distribution: p = 0.6 and p = 0.8. For which of the two values of p is Pr[X = 3] higher?

10. Compute (by hand) Pr[X > k] when X ~ Geometric(p).

11. CODE: Suppose we flip a coin n times and count the number of heads using a coin for which Pr[H] = p.

   - Write code to compute Pr[X = k] using the formula $Pr[X = k] = \binom{n}{k} p^k (1-p)^{n-k}$. Write your code in Binomial.java.
   - Plot a graph of Pr[X = k] vs. k for the case n = 10, p = 0.6 and for the case n = 10, p = 0.2.
   - Write a simulation to estimate Pr[X = 3] when n = 10, p = 0.6. You can use Coin.java and CoinExample2.java for this purpose. Verify the estimate using the earlier formula.

12. CODE: Add code to Poisson.java to compute Pr[X = k] and plot a graph of Pr[X = k] vs. k when γ = 2. Use the Taylor series for e^x to prove that $\sum_k$ Pr[X = k] adds up to 1.

13. CODE: Download BusStop.java and BusStopExample3.java, and modify the latter to estimate the probability that exactly three buses arrive during the interval [0,2]. Compare this with Pr[X = 3] when X ~ Poisson(2).

14. Consider the distribution for the 3-coin-flip example:

   - Pr[X = 0] = 0.064
   - Pr[X = 1] = 0.288
   - Pr[X = 2] = 0.432
   - Pr[X = 3] = 0.216
   - Sketch the CDF on paper.

15. What is an example of a continuous rv associate with the QueueControl.java application?

16. CODE: The program GaussianCDF.java estimates the CDF of a Gaussian rv. Execute the program to plot the CDF. Then, use this CDF to compute the following probabilities:

   - Pr[0 < X ≤ 2]
   - Pr[X > 0]

17. CODE: Modify UniformCDF.java and GaussianCDF.java to compute the derivative of each. What is the shape of F'(y) in each case?

18. If X denotes the first interarrival time in the bus-stop problem, estimate the CDF of X as follows:

   - Assume that values fall in the range [0,3] (i.e., disregard values outside this range).
   - Use ExponentialCDF.java as a template, and add modified code from UniformCDF.java
   - Next, compute the derivative of this function and display it.

19. Complete the calculation above. What would you get if Pr[H] = 0.5?

20. How does this relate to the 3-coin-flip example?

21. What does $\frac{n_k}{n}$ become in the limit? Unfold the sum for the 3-coin-flip example to see why this is true.

22. CODE: Download Coin.java and CoinExample3.java and let X = the number of heads in 3 coin flips.

   - Compute the average value of X using $\frac{1}{n} S_n$

- Estimate $\Pr[X = k]$ using $\frac{n_k}{n}$
- Compute $\sum_k k \frac{n_k}{n}$ using the estimate of $\frac{n_k}{n}$
- Compare with the $E[X]$ calculation you made earlier.

23. CODE: Use Coin.java and CoinExample4.java and let X = the number of flips needed to go get the first heads when $\Pr[\text{Heads}] = 0.1$. Compute the average value of X using $\frac{1}{n}S_n$ as you did in the previous exercise. Compare with the $E[X]$ calculation from earlier.

24. CODE: Try this computation with the uniform, Gaussian, and exponential distributions using UniformCDF2.java, GaussianCDF2.java, and ExponentialCDF2.java. Explore what happens when more intervals are used in the expectation computation than in the CDF estimation.

25. CODE: Estimate the density of the time spent in the system by a random customer in the QueueControl example. To do this, you need to build a density histogram of values of the variable timeInSystem in QueueControl.java.

26. Suppose X ~ Exponential(γ) with CDF F(x). Write down an expression for $F^{-1}(y)$, the inverse of F.

27. CODE: Add code to DiscreteGenExample.java to implement the above generator, and to test it by building a histogram.

28. CODE: Add code to ExponentialGenerator.java to implement the above idea. Use the inverse-CDF you computed earlier. The test code is written to produce a histogram. Use your modified version of PropHistogram.java to make a density histogram. Compare the result with the actual density (using $\gamma = 4$). How do you know your code worked?

29. CODE: Recall the winch example: Winch.java. Try different torque values to lift the load to the desired height. You will also need Function.java and SimplePlotPanel.java. Upon clicking "stop" you will see how the distance y varies with time.

- *alt*

2. CODE: Implement proportional control for the horizontal case in the nextStep() method of Winch2.java. Experiment with different values of the constant Kp, for example Kp = 5, 10, 50. What do you observe? Where have you seen this dynamic behavior before? What is the value of yMax in the code?

- **When Kp= 5 it is extremely slow, and seems to oscillate after it reaches the goal. As Kp increases the speed increases, but so does the oscillation. This behavior is similar to the behavior of a sine or cosine function. yMax in the code is 250.**

3. CODE: Implement proportional-derivative control for the horizontal case in Winch3.java. Experiment with different values of the constants Kp and Kd. For example: Kd = 10, 100, 1000 (using previous values of Kp). Does any value of Kd fix the problem?

- **Using derivative control allows the oscillation to minimize over time, however it never completely stops oscillating, and it takes a while to minimize. Higher Kd's seemed to have better results, though the Kp had to be increased as well. Kp=500 and Kd=1000 seemed to work well, almost completely stopping oscillation.**

4. CODE: Apply PD-control to the vertical case, writing that version in Winch4.java. Again, experiment with different values of Kp. What do you observe?

- **The Kp has to be higher for the vertical case. Kp=1000 and Kd=500 had promising results.**

5. CODE: Apply PID-control to the vertical case, writing your code in Winch5.java. Experiment with different Ki values.

- **Kp=1000, Kd=700, Ki=15 seemed to work well.**

6. CODE: Download and unpack carSim.jar. Use java CarGUI manual on the command-line to run in manual mode, then select the Unicycle and "basic sensor". Drive around some obstacles (scene-3 or scene-4) and observe how the sensors work.

- **There are 8 "ultrasonic" sensors on the car that provide distances in the N, NE, E, SE, ... directions.**

7. CODE: Examine the file BasicSensorPack.java. You can access the variable sonarDistance[] to get these distances. Here, sonarDistance[0] is the distance in front (North of the vehicle). Next, use this template to write a simple car controller for Scene-3 that goes towards the obstacle, stops just before it and turns left to face North. Don't forget to select "Basic sensor" in carGUI when testing.

- **DONE**

8. CODE: Implement the above algorithm in your controller using $\alpha = 10, \delta = 40$ and a forward velocity of 10, with Scene-3 as the setting. The NE distance is sonarDistance[7] whereas the SE distance is sonarDistance[5] (going anti-clockwise). Recall that we need to set the steering angle $\phi > 0$ to turn left and $\phi < 0$ to turn right. What do you notice when the unicycle tries to turn right at the obstacle corner?

- **The unicycle begins to shake because the NE distance value keeps changing between very large and very small.**

9. CODE: Implement the above algorithm in your controller. Use vel=2 as the reduced speed. Does it solve the problem?

- **It still shakes but since the car is always moving forward it eventually turns the corner.**

10. CODE: Implement proportional control in your controller. Use a small (e.g. vel=2) velocity throughout. Experiment with different values of the proportionality constant Kp. What do you observe when the vehicle turns a corner?

- **It still shakes around the corner but seems to be faster.**

11. CODE: Add derivative control to your controller. Experiment with different values of the two proportionality constants Kp and Kd. What do you observe?

- **Turning through the first corner is smoother, though for the second corner it still shakes and will sometimes turn slightly in the wrong direction due to the oscillation. Still seems faster than without derivative component.**

12. CODE: See if bounding the sensor values (as inputs to the algorithm) makes a difference. Observe what happens as the vehicle turns near the third obstacle corner (near the x-axis, as it's headed down).

- **The oscillations when turning are more pronounced, but it seems that the car is turning faster. I can probably tune the Kd value more to have less pronounced oscillations. As it is heading into the 3rd turn, my car works fine (though has a bit of shaking). However, I could see a situation where the car will crash into the corner since the NE and SE sensors show large distances but the E sensor effectively shows a distance of 0.**

13. CODE: Implement the above algorithm and see if it works. Can you suggest improvements?

- **While this does solve the problem of hitting the obstacle, it causes issues when first turning. Since the same delta is used for North distance and East distance, the first turn results in turning past 90 degrees. This could be solved by using a different delta for East and North. Also, to prevent the shaking, it may be better to average multiple measurements or use more than just the NE and SE sensors for PD control. Also implementing the integral part of PID may help.**

14. CODE: Implement the above in MySimpleCarController2.java and see if it works. What do you observe?

- **It does not work since it never stays in the correct state for more than one move call. This can be solved by using states instead of if/else statements.**

15. CODE: Implement the above algorithm and see if it works.

- **It does, but it never finds the goal again after following an obstacle.**

16. CODE: Implement the above in MySimpleCarController3.java and see if it works. The open-to-goal test has been implemented for you.

- **DONE**

17. How can you determine whether it's closer to turn clockwise vs. anticlockwise?

- **You can use the sensors you are checking distances with to determine which direction to turn. For instance, if openToGoal is checking between E and SE for the goal, then it will be closer to turn clockwise. If openToGoal is checking anything that contains W, then it will be closer to turn counter clockwise.**

18. What happens if we are too close to an obstacle on the left side? What changes are needed for obstacle avoidance on the left?

- **The same planning with the NE and SE sonar should be done with NW and SW to handle obstacle avoidance on the left.**

19. Assuming obstacle avoidance works, draw on paper a scenario on paper where the above goal-seeking algorithm fails to reach the goal. Suggest a better algorithm.

- *alt*
- **In the above example the car will get to a point where N will never sense a wall and the delta value of W and E will allow the car to try to fit through the gap. The car will not fit and it will crash. A better algorithm would use more sensors, or take averages of the current sensors to detect when there are small openings that the car can't fit through.**