JOHN GUTTAG: Hi, everybody. Welcome back to class. I have a lot to cover today, so I'm just going to get right to it. So you may remember at the end of last lecture, I talked about the Empirical Rule and said that there were a couple of assumptions underlying it. That one is the mean estimation error is zero.

And second, that the distribution of errors will be normally distributed. I didn't probably mention at the time, but we often call this distribution Gaussian after the astronomer Carl Gauss. And it looks like that.

Normal distributions are very easy to generate in Python. I have a little example here of generating, not a real normal distribution, but a discrete approximation of one. And the thing to really notice about it here is this line random.gauss. So that's a built in function of the random library.

The first argument is the mean. And the second argument is the standard deviation or a mu and sigma as they're usually called. Every time I call that, I will get a different-- or usually a different random value drawn from a Gaussian with the mean, in this case of 0 and a standard deviation of 100.

I'm then going to produce a plot of those so you can see what it looks like. And I'm going to do that using some things we haven't seen before. So first of all, we've seen histograms. So pylab.hist produces a histogram. Bins tells us how many bins we want in the histogram. I said we want 100. The default is 10. Dist is the values it will use for it.

And then this is something we haven't seen before, weights. Weights is a keyword argument. So normally when we produce a histogram, we take all of the values, the minimum to the maximum, and in this case, we would divide them into 100 bins, because I said, bins equals 100.

So the first bin might be, well, let's say we only had values ranging from 0 to 100. The first bin

would be all the 0's, all the 1's up to all the 99's. And it weights each value in the bin by 1. So if the bin had 10 values falling in it, the y-axis would be a 10. If the bin had 50 values falling in it, the y-axis would go up to 50.

You can tell it how much you want to weight each bin, the elements in the bins. And say, no, I don't want them each to count as 1, I want them to count as a half or a quarter, and that will change the y-axis. So that's what I've done here.

What I've said is I've created a list and I want to say for each of the bins-- in this case I'm going to weigh each of them the same way-- the weight is going to be 1 over the number of samples. I'm multiplying it by the len of dist, that will be how many items I have. And that will tell me how much each one is being weighted.

So for example, if I have, say, 1,000 items, I could give 1,000 values and say, I want this item weighted by 1, and I want this item over here weighted by 12 or a half. We rarely do that. Usually, what we want is to give each item the same weight.

So why would I want it not to be weighted at just one? Because I want my y-axis to be more easily interpreted, and essentially give me the fraction of the values that fell in that bin. And that's what I'm doing here.

The other new thing I'm doing here is the plotting commands, including pylab.hist, many of them return values. Usually, I just ignore that value when we just say pylab.plot or pylab.hist. Here I am taking the value. The value in this case for a histogram is a tuple of length 2. The first element is a list or an array, giving me how many items are in each bin. And the second is the patches used to produce the beautiful pictures we're use to seeing.

So here what I'm going to do is take this value so that I can do this at the end. Now, why would I want to look at the fraction within approximately 200 of the mean? What is that going to correspond to in this case? Well, if I divide 200 by 2 I get 100. Which happens to be the standard deviation.

So in this case, what I'm going to be looking at is what fraction of the values fall within two standard deviations of the mean? Kind of a check on the empirical rule, right? All right, when I run the code I get this. So it is a discrete approximation to the probability density function.

You'll notice, unlike the previous picture I showed you which was nice and smooth, this is jaggedy. You would expect it to be. And again, you can see it's very nice that the peak is what

we said the mean should be, 0. And then it falls off.

And indeed, slightly more than 95% fall within two standard deviations of the mean. I'm not even surprised that it's a little bit more than 95% because, remember the magic number is 1.96, not 2. But since this is only a finite sample, I only want it to be around 95. I'm not going to worry too much whether it's bigger or smaller. All right?

So random.gauss does a nice job of giving us Gaussian values. We plotted them and now you can see that I've got the relative frequency. That's why I see fractions in the y-axis rather than counts. And that would be because of the way I use the weights command.

All right, let's return to PDFs. So as we said last time, distributions can be defined by Probability Density Functions, and that gives us the probability of some random variable lying between two values. It defines a curve where the values in the x-axis lie between the minimum and maximum values of the variable. And it's the area under the curve-- and we'll come back to this-- between those two points that give us the probability of an example falling in that range.

So now let's look at it for a normal distribution. You may recall from the last lecture this rather exotic looking formula which defines a PDF for a normal distribution. And here's some code that's as straight forward an implementation as one could imagine of this formula, OK. So that now is a value that, given a mu and a sigma and an x, gives me the x associated with that mu and sigma, OK? And you'll notice there's nothing random about this. All right, it is giving me the value.

Now, let's go down here. And I'm going to, for a set of x's, get the set of y's corresponding to that and then plot it. I'm going to set mu to 0 and sigma to 1, the so-called standard normal distribution. And I'm going to look at the distribution from minus 4 to 4. Nothing magic about that other than, as you'll see, it's kind of a place where it asymptotes near 0.

So while x is less than 4, I'll get the x-value, I'll get the y-value corresponding to that x by calling Gaussian, increment x by 0.05 and do that until I'm done. And then simply, I'll plot the x-values against the y-values and throw a title on it using pylab.title. All right, code make sense?

Well, this is where I got that beautiful picture we've looked at before. When I plotted here, it looks, actually quite smooth. It's not, it's really connecting a bunch of tiny little lines but I made

the points close enough together that it looks at least here smooth at this resolution. So we know what the values on the x-axis are. Those are the values I happen to want to look at, from minus 4 to 4. What are the values on the y-axis?

We kind of would like to interpret them as probabilities, right? But we could be pretty suspicious about that and then if we take this one point that's up here, we say the probability of that single point is 0.4. Well, that doesn't make any sense because, in fact, we know the probability of any particular point is 0 in some sense, right?

So furthermore, if I chose a different value for sigma, I can actually get this to go bigger than 1 on the y-axis. So if you take sigma to be say, 0.1-- I think the y-axis goes up to something like 40. So we know we don't have probabilities in the range 40.

So if these aren't probabilities, what are they? What are the y values? Well, not too surprising since I claimed this was a probability density function, they're densities. Well, what's a density? This makes sense. I'll say it and then I'll try and explain it. It's a derivative of the cumulative distribution function.

Now, why are we talking about derivatives in the first place? Well, remember what we're trying to say. If we want to ask, what's the probability of a value falling between here and here, we claim that that was going to be the area under this curve, the integral. Well, as you know from 18.01, there's a very clear relationship between derivatives and integrals.

And so if we interpret each of these points as a derivative, in some sense the slope here, then we can look at this as the area just by integrating under there. So to interpret a PDF, we always do it mathematically. Actually, I do it just by looking at it. But the only really interesting mathematical questions to ask have to do with area. Once we have the area, we can, then, talk about the probabilities of some value falling within a region of the curve.

So what's interesting here is not the numbers per se on the y-axis but the shape of the curve, because those numbers have to be related to the numbers on the x-axis, dx, dy right? We're looking at derivatives.

All right, so now, we have to talk about integration. I promise you'll only hear about it for another few minutes then we'll leave the topic. So I mentioned before SciPy as a library that contains a lot of useful mathematical functions. One of them is integrate.quad.

Well, the integrate part is obvious. It means integration. Quad is telling you the algorithm it's choosing to do the integration. All of these integrals are going to be actual approximations to the real integral. SciPy is not doing some clever mathematics to get an analytical solution. It's using a numerical technique to approximate the integral. And the one here happens to be called quadrature, it doesn't matter.

All right, you can pass it up to four arguments. You must pass it to function to be integrated, that makes sense. A number representing the lower limit of the integration-- you need to give it that. A number representing the upper limit-- you need to give it that. And then the fourth argument is a tuple supplying all values for the arguments, except the first of the function you are integrating.

I'll show you an example of that on the next slide. And we'll see that it returns a tuple, an approximation to the result, what it thinks the integral is, and an estimate of how much error there might be in that one. We'll ignore the error for the moment.

All right, let's look at the code. So we start by importing scipy.integrate. That gives us all the different integration methods. I'm not going to show you the code for Gaussian since I showed it to you a couple of minutes ago. But I wanted you to remember that it takes three arguments, x, mu, and sigma.

Because when we get down here to the integration, we'll pass at the function Gaussian and then the values that we want to integrate over. So those will be the values that x can take upon. And that will change as we go from mu minus the number of standard deviations times sigma to mu plus the number of standard deviations times sigma.

And then, this is the optional fourth argument, the tuple, mu, and sigma. Why do I need to pass that in? Because Gaussian is a ternary argument, or a function that takes three values. And I'm going to integrate over values of x so I have to fix mu and sigma to constants, which is what I'm doing down here. And then I'll take the zeroth value, which is its estimate of the integral. All right, so that's the new thing.

The rest of the code is all stuff you've seen. For t and range number of trials, I'm going to choose a random mu between minus 10 and 10 and a random sigma between 1 and 10. It doesn't matter what those constants are. And then for the number of standard deviations in 1, 1.96, and 3, I'm going to integrate Gaussian over that range. And then we're just going to see how many of them fall within that range.

In some sense, what we're doing is we're checking the empirical rule. We're saying, take the Gaussian. I don't care what mu and sigma are. It doesn't matter. The empirical rule will still hold, I think. But we're just checking it here, OK?

Well, here are the results. So from mu equals 9 and sigma equals 6, I happened to choose those, we'll see the fracture within 1, fraction within 1.96 and 3. And so for these random mus and sigmas, you can see that all of them-- and you can set them to whatever you want when you get your hand them the code. Essentially, what we have is, whoops, the empirical rule actually works. One of those beautiful cases where you can test the theory and see that the theory really is sound. So there we go.

So why am I making such a big deal of normal distributions? They have lots of nice mathematical properties, some of which we've already talked about. But all of that would be irrelevant if we didn't see them. The good news is they're all over the place. I've just taken a few here. Up here we'll see SAT scores. I would never show that to high school students, or GREs to you guys. But you can see that they are amazingly well-distributed along a normal distribution.

On down here, this is plotting percent change in oil prices. And again, we see something very close to a normal distribution. And here is just looking at heights of men and women. And again, they clearly look very normal. So it's really quite impressive how often they occur.

But not everything is normal. So we saw that the empirical rule works for normal distributions. I won't say I proved it for you. I illustrated it for you with a bunch of examples. But are the outcomes of the spins of a roulette wheel normal? No. They're totally uniform, right?

Everything is equally probable-- a 4, a 6, an 11, a 13, double-0 if you're in Las Vegas. They're all equally probable. So if I plotted those, I'd basically just get a straight line with everything at 1 over however many pockets there are.

So in that case, why does the empirical rule work? We saw that we were doing some estimates about returns and we used the empirical rule, we checked it and, by George, it was telling us the truth. And the reason is because we're not reasoning about a single spin of the wheel but about the mean of a set of spins.

So if you think about it, what we were reasoning about was the return of betting. If we look at

one spin-- well, let's say we bet $1. The return is either minus 1 because we've lost our dollar. Or if we get lucky and our pocket happens to come up, it was 36, I think, or 35. I forget which, OK? But that's all. So if we plotted a histogram, we would see a huge peak at minus 1 and a little bump here at 36 and nothing in the middle. Clearly, not a normal distribution.

But what we're reasoning about is not the return of a single spin but the return of many spins. If we played 1,000 spins, what is our expected return? As soon as we end up reasoning, not about a single event but about the mean of something, we can imply something called the Central Limit Theorem. And here it is.

It's actually for something so important, very simple. It says that given a sufficiently large sample-- and I love terms like sufficiently large but we'll later put a little meat on that-- the following three things are true. The means of the samples in a set of samples, the so-called sample means will be approximately normally distributed.

So that says if I take a sample-- and remember, a sample will have multiple examples. So just to remind people. A population is a set of examples. A sample is a subset of the population. So it too is a set of examples, typically.

If this set is sufficiently large-- certainly 1 is not sufficiently large-- then it will be the case that the mean of the means-- so I take the mean of each sample and then I can now plot all of those means and so take the mean of those, right-- and they'll be normally distributed.

Furthermore, this distribution will have a mean that is close to the mean of the population. The mean of the means will be close to the mean of the population. And the variance of the sample means will be close to the variance of the population divided by the sample size. This is really amazing that this is true and dramatically useful.

So to get some insight, let's check it. To do that, postulate that we have this kind of miraculous die. So instead of a die that when you roll it you get a number 1, 2, 3, 4, 5, or 6, this particular die is continuous. It gives you a real number between 0 and 5, or maybe it's between 1 and 6, OK? So it's a continuous die.

What we're going to do is roll it a lot of times. We're going to say, how many die? And then, how many times are we going to roll that number of die? So the number of die will be the sample size-- number of dice will be the sample size. And then we'll take a bunch of samples which I'm calling number of rolls. And then we'll plot it and I'm just choosing some bins and

some colors and some style and various other things just to show you how we use the keyword arguments.

Actually, I said the number of rolls is the number of trials. But it isn't quite that because I'm going to get the number of trials by dividing the number of rolls by the number of dice. So if I have more dice, I get to have fewer samples, more dice per sample, all right? Then we'll just do it. So it will be between 0 and 5 because random.random returns a number between 0 and 1 and I'm multiplying it by 5.

And then we'll look at the means and we'll plot it all. Again, we're playing games with weights just to make the plot a little easier to read. And here's what we get. If we roll one die, the mean is very close to 2.5. Well, that's certainly what you'd expect, right?

It's some random number between 0 and 5. 2.5 is a pretty good guess as to what it should average. And it has a standard deviation of 1.44. And that's a little harder to guess that that's what it would be. But you could figure it out with a little math or as I did here with the simulation.

But now, if I roll 50 dice, well, again, the mean is close to 2.5. It's what you'd expect, right? I roll 50 die, I get the mean value of those 50. But look how much smaller the standard deviation is. More importantly, what we see here is that if we look at the value, the probability is flat for all possible values between 0 and 5 for a single die. But if we look at the distribution for the means, it's not quite Gaussian but it's pretty close.

Why is it not Gaussian? Well, I didn't do it an infinite number of times. Did it quite a few, but not an infinite number. Enough that you didn't want to sit here while it ran. But you can see the amazing thing here that when I go from looking at 1 to looking at the mean of 50, suddenly I have a normal distribution. And that means that I can bring to bear on the problem the Central Limit Theorem.

We can try it for roulette. Again I'm not going to make you sit through a million trials of 200 spins each. I'll do it only for fair roulette. And again, this is a very simple simulation, and we'll see what we get. And what we see is it's not quite normal, again, but it definitely has that shape.

Now, it's going to be a little bit strange because I can't lose more than one, if I'm betting one. So it will never be quite normal because it's going to be truncated down on the left side,

whereas the tail can be arbitrarily long. So again, mathematically it can't be normal but it's close enough in the main region, where most of the values lie, that we can get away with applying the empirical rule and looking at answers. And indeed as we saw, it does work.

So what's the moral here? It doesn't matter what the shape of the distribution of the original values happen to be. If we're trying to estimate the mean using samples that are sufficiently large, the CLT will allow us to use the empirical rule when computing confidence intervals. Even if we go back and look at this anomaly over in the left, what do you think would happen if I, instead of had 200, have, say, 1,000?

What's the probability of the average return being minus 1 of 1,000 bets? Much smaller than for 100 bets. To lose 1,000 times in a row is pretty unlikely. So to get all the way to the left is going to be less likely, and, therefore, the thing will start looking more and more normal as the samples get bigger. All right, and so we can use the CLT to justify using the empirical rule when we compute confidence intervals.

All right, I want to look at one more example in detail. This is kind of an interesting one. You might think that randomness is of no use for, say, finding the value of pi because there's nothing random about that. Similarly, you might think that randomness was of no use in integrating a function, but in fact, the way those numerical algorithms work is they use randomness.

What you're about to see is that randomness, and indeed things related to Monte Carlo simulations, can be enormously useful, even when you're computing something that is inherently not random like the value of pi here. And we won't ask you to remember that many digits on the quiz and the exam.

All right, so what's pi? Well, people have known about pi for thousands and thousands of years. And what people knew was that there was some constant, we'll call it pi. It wasn't always called that. Such that it was equal to the circumference of a circle divided by the diameter, and furthermore, the area was going to be pi r squared. People knew that way back to the Babylonians and the Egyptians. What they didn't know is what that value was.

The earliest known estimate of pi was by the Egyptians, on something called the Rhind Papyrus pictured here. And they estimated pi to be 4 times 8 over 9 squared, or 3.16. Not so bad. About 1,100 years later an estimate of pi appeared in the Bible, Kings 7:23. It didn't say pi is but it described-- I think this was a construction of King Solomon's Temple. "He made a

molten sea, ten cubits from one brim to the other." it was round. It was a circle. "His height was five cubits and a line of 30 cubits did compass it round about."

Well, if you do the arithmetic, what does this imply the value of pi to be? 3, and I'm sure that's what Mike Pence thinks it is. About 300 years later, Archimedes did a better job of it. He estimated pi by constructing a 96-sided polygon. There's a picture of one. It looks a lot like a circle. On the left is one with fewer sides.

And what he did is he then-- since it was a polygon he knew how to compute its area or it's circumference and he just counted things up and he said, well, pi is somewhere between 223/71 and 22/7. And that turns out to actually-- if you take the average of that it will be a really good estimate.

2000 years after Archimedes, the French mathematicians Buffon and Laplace proposed finding the value of pi using what we would today call a Monte Carlo simulation. They did not have a computer to execute this on. So what they proposed-- it started with this mathematics. They took a circle and inscribed it in a square, a square of two units of whatever it is per side.

And therefore we know that the area of the square is 2 times 2 or 4. We know that the area of the circle is pi r squared. And since we know that r is 1-- because that's the square it's inscribed in-- we know that the area of a circle is exactly pi. What Buffon then proposed was dropping a bunch of needles at random-- kind of like when Professor Grimson was sorting things-- and seeing where they land.

Some would land in the square but not in the circle. Some would land in the circle. You would ignore any that landed outside the square. And then they said, well, since they're falling at random, the ratio of the needles in the circle to needles in the square should exactly equal the area of the square over the area of the circle, exactly, if you did an infinite number of needles. Does that make sense?

Now, given that, you can do some algebra and solve for the area of the circle and say it has to be the area of the square times the number of needles in the circle divided by the needles in the square. And since we know that the area of the circle is pi that tells us that pi is going to equal four times the needles in the circle. That's 4 is the area of the square divided by the number of needles in the square.

And so the argument was you can just drop a bunch of these needles, see where they land,

add them up and from that you would magically now know the actual value of pi. Well, we tried a simulation one year in class but rather than using needles we had an archer and we blindfolded him so he would shoot arrows at random and we would see where they ended up. There's a video of this here if you want to see it. I'm going to play only the very end of the video which describes the results.

Maybe we should just use Python to build a Monte Carlo simulation instead.

So that's what happened. After it was all over, Anna came up and gave me some sound advice. Yeah, I was very proud of that particular shot with the apple. I had hoped the student would put it on his or her head but no one volunteered. That was not done blindfolded.

Any rate, so here is the simulation. So first we have throwing the needles. For needles in range 1 to number of needles plus 1, we're going to choose the random x or random y and we're going to see whether or not it's in the circle. And there we will use the Pythagorean theorem to tell us that, all right? And we'll just do this, and then we're going to return exactly the formula we saw in the previous slide.

Now, comes an interesting part. We need to get an estimate. So we start with the number of needles and the number of trials. For t in range number of trials, we'll get a guess by throwing number of needles. And then we'll append that guess to our list of estimates.

We'll then compute the standard deviation, get the current estimate which will be the sum of the estimates divided by the len of the estimate, just the mean of the estimate. And then we'll print it, all right? So given a number of needles and a number of trials, we'll estimate pi and we'll give you the standard deviation of that estimate.

However, to do that within a certain precision, I'm going to have yet another loop. And I'm showing you this because we often structure simulations this way. So what I have here is the number of trials, which is not so interesting, but the precision.

I'm saying, I would like to know the value of pi and I would like to have good reason to believe that the value you give me is within 0.005, in this case, of the true value. Or maybe more precisely, I would like to know with a confidence of 95% that the true value is within a certain range.

So what this is going to do is it's just going to-- and I should probably use 1.96 instead of 2, but oh, well-- it's just going to keep increasing the number of needles, doubling the number of

needles until it's confident about the estimate, confident enough, all right? So this is a very common thing. We don't know how many needles we should need so let's just start with some small number and we'll keep going until we're good.

All right, what happens when we run it? Well, we start with some estimates that when we had 1,000 needles it told us that pi was 3.148 and standard deviation was 0.047 et cetera. So a couple of things to notice. One, are my estimates of pi getting monotonically better? You'd like to think, as I add more needles, my estimates are getting more accurate. So that's question one. Are they getting more accurate? Not monotonically, right?

If we look at it, where are they getting worse? Well, let's see. All right, 3.148, well, 3.13 is already worse, right? So I double the number of needles and I get a worse estimate. Now, the trend is good. By the time I get here, I've got a pretty good estimate.

So overall as I look at larger samples, a bigger subset of the population, my estimates are trending towards better but not every time. On the other hand, let's look at the standard deviations. What we see here is that the standard deviations are, indeed, getting monotonically better.

Now, there's nothing mathematical guaranteeing that, right? Because there is randomness here. But I'm increasing the number of needles by so much that it kind of overrides the bad luck of maybe getting a bad random set. And we see those are getting better. So the important thing to see here is not that I happened to get a better estimate, but that I know more about the estimate. I can have more confidence in the estimate because it's closing in on it.

So the moral here is it's not sufficient to produce a good answer. I've said this before. We need to have reason to believe that it is close to the right answer. So in this case, I'm using the standard deviation and say, given that it's gotten quite small-- you know, 1.96 times 0.002 is indeed a small number.

I could make it smaller if I wanted. I have good reason to believe I'm close to the right value of pi. Everyone agree with that? Is that right? Not quite, actually. It would be nice if it were true. But it isn't.

So let's look at some things. Is it correct to state that 95% of the time we run this simulation you'll get an estimate of pi between these two values? I don't expect you to do the arithmetic in your head but the answer is yes. So that is something we believe is true by the math we've

been looking at for the last two lectures.

Next statement, with a probability of 0.95, the actual value of pi is between these two things. Is that true? In fact, if I were to say, with a probability of 1, the actual value is pi is between those two values, would it be true? Yes. So they are both true facts.

However, only the first of these can be inferred from our simulation. While the second fact is true, we can't infer it from the simulation. And to show you that, statistically valid is not the same as true, we'll look at this. I've introduced a bug in my simulation.

I've replaced the 4 that we saw we needed by 2, now, an easy kind of mistake to make. And now, if we go to the code-- well, what do you think will happen if we go to the code and run it? We'll try it. We'll go down here to the code. We'll make that a 2. And what you'll see as it runs is that once again we're getting very nice confidence intervals, but totally bogus values of pi.

So the statistics can tell us something about how reproducible our simulation is but not whether the simulation is an actually, accurate model of reality. So what do you need to do? You need to do something like a sanity check. So here you might look at a polygon and say, well, clearly that's a totally wrong number. Something is wrong with my code.

OK, so just to wrap-up. What we've shown is a way to find pi. This is a generally useful technique. To estimate the area of any region r, you pick an enclosing region, call it e, such that it's easy to estimate the area of e, and r lies within it. Pick some random sets of points within e, let f be the fraction and fall within r, multiply e by f and you're done.

So this for example, is a very common way to do integration. I promised you we'd talk about integration. So here's a sine of x. If I want to integrate the sine of x over some region, as done here, all I need to do is pick a bunch of random points, red and black in this case, and look at the ratio of one to the other.

So showing how we can use randomness to again, compute something that is not inherently random. This is a trick people use over and over and over again when confronted with some situation where it's not easy to solve for things mathematically. You just do a simulation and if you do it right, you get a very good answer. All right, we will move on to a different topic on Wednesday.