PROFESSOR: I want to go back to where I stopped at the end of Tuesday's lecture, when you let me pull a fast one on you. I ended up with a strong statement that was effectively a lie. I told you that when we drop a large enough number of pins, and do a large enough number of trials, we can look at the small standard deviation we get across trials and say, that means we have a good answer. It doesn't change much. And I said, so we can tell you that with 95% confidence, the answer lies between x and y, where we had the two standard deviations from the mean.

That's not actually true. I was confusing the notion of a statistically sound conclusion with truth. The utility of every statistical test rests on certain assumptions. So we talked about independence and things like that. But the key assumption is that our simulation is actually a model of reality.

You can recall that in designing the simulation, we looked at the Buffon-Laplace mathematics and did a little algebra from which we derived the code, wrote the code, ran the simulation, looked at the results, did the statistical results, and smiled. Well, suppose I had made a coding error. So, for example, instead of that 4 there-- which the algebra said we should have-- I had mistakenly typed a 2. Not an impossible error.

Now if we run it, what we're going to see here is that it converges quite quickly, it gives me a small standard deviation, and I can feel very confident that my answer that pi is somewhere around 1.569. Well, it isn't of course. We know that that's nowhere close to the value of pi. But there's nothing wrong with my statistics. It's just that my statistics are about the simulation, not about pi itself.

So what's the moral here? Before believing the results of any simulation, we have to

have confidence that our conceptual model is correct. And that we have correctly implemented that conceptual model. How can we do that? Well, one thing we can do is test our results against reality. So if I ran this and I said pi is about 1.57, I could go draw a circle, and I could crudely measure the circumference, and I would immediately know I'm nowhere close to the right answer. And that's the right thing to do. And in fact, what a scientist does when they use a simulation model to derive something, they always run some experiments to see whether their derived result is actually at least plausibly correct.

Statistics are good to show that we've got the little details right at the end, but we've got to do a sanity check first. So that's a really important moral to keep in mind. Don't get seduced by a statistical test and confuse that with truth.

All right, I now want to move on to look at some more examples that do the same kind of thing we've been doing. And in fact, what we're going to be looking at is the interplay between physical reality,-- some physical system, just in the real world-- some theoretical models of the physical system, and computational models.

Because this is really the way modern science and engineering is done. We start with some physical situation-- and by physical I don't mean it has to be bricks and mortar, or physics, or biology. The physical situation could be the stock market, if you will,-- some real situation in the world. We use some theory to give us some insight into that, and when the theory gets too complicated or doesn't get us all the way to the answer, we use computation.

And I now want to talk about how those things relate to each other. So imagine, for example, that you're a bright student in high school biology, chemistry, or physics-- a situation probably all of you who have been in. You perform some experiment to the best of your ability. But you've done the math and you know your experimental results don't actually match the theory. What should you do? Well I suspect you've all been in this situation. You could just turn in the results and risk getting criticized for poor laboratory technique. Some of you may have done this. More likely what you've done is you've calculated the correct results and turned those in, risking

some suspicion that they're too good to be true. But being smart guys, I suspect what all of you did in high school is you calculated the correct results, looked at your experimental results, and met somewhere in between to introduce a little error, but not look too foolish.

Have any of you cheated that way in high school? Yeah well all right. We have about two people who would admit it. The rest of you are either exceedingly honorable, or just don't want to admit it. I confess, I had fudged experimental results in high school. But no longer, I've seen the truth.

All right, to do this correctly you need to have a sense of how best to model not only reality, but also experimental errors. Typically, the best way to model experimental errors-- and we need to do this even when we're not attempting to cheat-- is to assume some sort of random perturbation of the actual data. And in fact, one of the key steps forward, which was really Gauss' big contribution, was to say we can typically model experimental error as normally distributed, as a Gaussian distribution.

So let's look at an example. Let's consider a spring. Not the current time of year, or a spring of water, but the kind of spring you looked at in 8.01. The things you compress with some force then they expand, or you stretch, and then they contract.

Springs are great things. We use them in our cars, our mattresses, seat belts. We use them to launch projectiles, lots of things. And in fact, as we'll see later, they're frequently occurring in biology as well.

I don't want to belabor this, I presume you've all taken 8.01. Do they still do springs in 8.01? Yes, good, all right. So as you know, in 1676-- maybe you didn't know the date-- the British physicist, Robert Hooke, formulated Hooke's Law to explain the behavior of springs. And the law is very simple, it's f equals minus kx.

In other words, the force, f, stored in the spring is linearly related to x, the distance the spring has been either compressed or stretched. OK. so that's Hooke's law, you've all seen that. The law holds true for a wide variety of materials and systems

including many biological systems. Of course, it does not hold for an arbitrarily large force. All springs have an elastic limit and if you stretch them beyond that the law fails. Has anyone here ever broken a Slinky that way. Where you've just taken the spring and stretched it so much that it's no longer useful. Well you've exceeded its elastic limit.

The proportionate of constant here, k, is called the spring constant. And every spring has a constant, k, that explains its behavior. If the spring is stiff like the suspension in an automobile, k is big. If the spring is not stiff like the spring in a ballpoint pen, k is small. The negative sign is there to indicate that the force exerted by the spring is in the opposite direction of the displacement. If you pull a spring bring down, the force exerted by the spring is going up.

Knowing the spring constant of a spring is actually a matter of considerable practical importance. It's used to do things like calibrate scales,-- one can use to weigh oneself, if one wants to know the truth-- atomic force microscopes, lots of kinds of things. And in fact, recently people have started worrying about thinking that you should model DNA as a spring, and finding the spring constant for DNA turns out to be of considerable use in some biological experiments.

All right, so generations of students have learned to estimate springs using this very simple experiment. You've probably most of you have done this. Get a picture up here, all right. So what you do is you take a spring and you hang it on some sort of apparatus, and then you put a weight of known mass at the bottom of the spring, and you measure how much the spring has stretched. You then can do the math, if f equals minus kx. We also have to know that f equals m times a, mass times acceleration. We know that on this planet at least the acceleration due to gravity is roughly 9.81 meters per second per second, and we can just do the algebra and we can calculate k. So we hang one weight in the spring, we measure it, we say, we're done. We now know what k is for that spring. Not so easy, of course, to do this experiment if the spring is a strand of DNA. So you need a slightly more complicated apparatus to do that.

This would be all well and good if we didn't have experimental error, but we do. Any experiment we typically have errors. So what people do instead is rather than hanging one weight on the spring, they hang different weights-- weights of different mass-- they wait for the spring to stop moving and they measure it, and now they have a series of points. And they assume that, well I've got some errors and if we believe that our errors are normally distributed some will be positive, some will be negative. And if we do enough experiments it will kind of all balance out and we'll be able to actually get a good estimate of the spring constant, k.

I did such an experiment, put the results in a file. This is just a format of the file. The first line tells us what it is, it's the distance in meters and a mass in kilograms. And then I just have the two things separated by a space, in this case. So my first experiment, the distance I measured was 0.0865 and the weight was 0.1 kilograms.

All right, so I've now got the data, so that's the physical reality. I've done my experiment. I've done some theory telling me how to calculate k. And now I'm going to put them together and write some code. So let's look at the code. Think we'll skip over this, and I'll comment this out, so we don't see get pi get estimated over and over again.

So the first piece of code is pretty simple, it's just getting the data. And again, this is typically the way one ought to structure these things. I/O, input/output, is typically done in a separate function so that if the format of the data were changed, I'd only have to change this, and not the rest of my computation. it opens the file, discards the header, and then uses a split to get the x values and the y values, all right. So now I just get all the distances and all the masses-- not the x's and the y's yet, just distances and masses. Then I close the file and return them. Nothing that you haven't seen before. Nothing that you won't get to write again, and again, and again, similar kinds of things.

Then I plot the data. So here we see something that's a little bit different from what we've seen before. So the first thing I do is I got my x and y by calling, GetData. Then I do a type conversion. What GetData is returning is a list. I'm here going to

convert a list to another type called an array. This is a type implemented by a class supplied by PyLab which is built on top of something called NumPy, which is where it comes from. An array is kind of like a list. It's a sequence of things. There's some list operations methods that are not available, like append, but it's got some other things that are extremely valuable. For example, I can do point-wise operations on an array. So if I multiply an array by 3, what that does is it multiplies each element by 3. If I multiply one array by another, it does the cross products. OK, so they're very valuable for these kinds of things.

Typically, in Python, one starts with a list, because lists are more convenient to build up incrementally than arrays, and then converts them to an array so that you can do the math on them. For those of you who've seen MATLAB you're very familiar with the concept of what Python calls an array. Those of you who know C or Pascal, what it calls an array has nothing to do with what Python or PyLab calls an array. So can be a little bit confusing. Any rate, I convert them to arrays. And then what I'll do here, now that I have an array, I'll multiply my x values by the acceleration due to gravity, this constant 9.81. And then I'm just going to plot them. All right, so let's see what we get here.

So here I've now plotted the measure displacement of the spring. Force in Newtons, that's the standard international unit for measuring force. It's the amount of force needed to accelerate a mass of 1 kilogram at a rate of 1 meter per second per second. So I've plotted the force in Newton's against the distance in meters. OK. Now I can go and calculate k. Well, how am I going to do that? Well, before I do that, I'm going to do something to see whether or not my data is sensible.

What we often do, is we have a theoretical model and the model here is that the data should fall on a line, roughly speaking, modular experimental errors. I'm going to now find out what that line is. Because if I know that line, I can compute k. How does k relate to that line? So I plot a line. And now I can look at the slope of that line, how quickly it's changing. And k will be simply the inverse of that.

How do I get the line? Well, I'm going to find a line that is the best approximation to

the points I have. So if, for example, I have two points, a point here and a point here, I know I can quote, fit a line to that curve-- to those points-- it will always be perfect. It will be a line. So this is what's called a fit.

Now if I have a bunch of points sort of scattered around, I then have to figure out, OK, what line is the closest to those points? What fits it the best? And I might say, OK, it's a line like this. But in order to do that, in order to fit a line to more than two points, I need some measure of the goodness of the fit. Because what I want to choose here is the best fit. What line is the best approximation of the data I've actually got? But in order to do that, I need some objective function that tells me how good is a particular fit. It lets me compare two fits so that I can choose the best one.

OK, now if we want to look at that we have to ask, what should that be? There are lots of possibilities. One could say, all right let's find the line that goes through the most points, that actually touches the most points. The problem with that is it's really hard, and may be totally irrelevant, and in fact you may not find a line that touches more than one point. So we need something different. And there is a standard measure that's typically used and that's called the least squares fit. That's the objective function that's almost always used in measuring how good any curve-- or how well, excuse me, any curve fits a set of points.

What it looks like is the sum from L equals 0 to L equals the len of the observed points minus 1, just because of the way things will work in Python. But the key thing is what we're summing is the observed at point L minus the predicted at point L-squared. And since we're looking for the least squares fit, we want to minimize that. The smallest difference we can get. So there's some things to notice about this. Once we have a quote fit, in this case a line for every x value the fit predicts a y value. Right? That's what our model does. Our model in this case will take the independent variable, x, the mass, and predict the dependent variable, the displacement.

But in addition to the predicted values, we have the observed values, these guys.

And now we just measure the difference between the predicted and the observed, square it, and notice by squaring the difference we have discarded whether it's above or below the line-- because we don't care, we just care how far it's from the line. And then we sum all of those up and the smaller we can make that, the better our fit is. Makes sense?

So now how do we find the best fit? Well, there's several different methods you could use. You can actually do this using Newton's method. Under many conditions there are analytical solutions, so you don't have to use approximation you can just compute it. And the best news of all, it's built into PyLab So that's how you actually do it. You call the PyLab function that does it for you. That function is called Polyfit.

Polyfit takes three arguments. It takes all of the observed X values, all of the observed Y values, and the degree of the polynomial. So I've been talking about fitting lines. As we'll see, polyfit can be used to fit polynomials of arbitrary degree to data. So you can fit a line, you can fit a parabola, you can fit cubic. I don't know what it's called, you can fit a 10th order polynomial, whatever you choose here. And then it returns some values. So if we think about it being a line, we know that it's defined by the y value is equal to ax plus b. Some constant times the x value plus b, the y-intercept.

So now let's look at it. We see here in fit data, what I've done is I've gotten my values as before, and now I'm going to say, a,b equals pyLab.polyfit of xVals, y values and 1. Since I'm looking for a line, the degree is 1. Once I've got that, I can then compute the estimated y values, a times pyLab.array. I'm turning the x values into an array, actually I didn't need to do that since I'd already done it, that's okay-- plus b. And now I'll plot it and, by the way now that I've got my line, I can also compute k. And let's see what we get.

All right, I fit a line, and I've got a linear fit, and I said my spring constant k is 21 point -- I've rounded it to 5 digits just so would fit nicely on my plot. OK. The method that's used to do this in PyLab is called a linear regression. Now you might think it's called linear regression because I just used it to find a line, but in fact that's not why

it's called linear regression. Because we can use linear regression to find a parabola, or a cubic, or anything else. The reason it's called linear, well let's look at an example. So if I wanted a parabola, I would have y equals ax-squared plus bx plus c. We think of the variables, the independent variables, as x-squared and x. And y is indeed a linear function of those variables, because we're adding terms. Not important that you understand the details, it is important that you know that linear regression can be used to find polynomials other than lines.

All right, so we got this done. Should we be happy? We can look at this, we fit the best line to this data point, we computed k, are we done? Well I'm kind of concerned, because when I look at my picture it is the best line I can fit to this, but wow it's not a very good fit in some sense, right. I look at that line, the points are pretty far away from it. And if it's not a good fit, then I have to be suspicious about my value of k, which is derived from having the model I get by doing this fit.

Well, all right, let's try something else. Let's look at FitData1, which in addition to doing a linear fit, I'm going to fit a cubic -- partly to show you how to do it. Here I'm going to say abcd equals pyLab.polyfit of xVals, yVals and 3 instead of 1. So it's a more complex function. Let's see what that gives us. First let me comment that out. So we're going to now compare visually what we get when we get a line fit versus we get a cubic fit to the same data.

Well it looks to me like a cubic is a much better description of the data, a much better model of the data, than a line. Pretty good. Well, should I be happy with this? Well, let's ask ourselves in one question, why are we building the model? We're building the model so that we can better understand the spring. One of the things we often do with models is use them to predict values that we have not been able to run in our experiments. So, for example, if you're building a model of a nuclear reactor you might want to know what happens when the power is turned off for some period of time. In fact, if you read today's paper you noticed they've just done a simulation model of a nuclear reactor, in, I think, Tennessee, and discovered that if it lost power for more than two days, it would start to look like the nuclear reactors in Japan. Not a very good thing. But of course, that's not an experiment anyone

wants to run. No one wants to blow up this nuclear reactor just to see what happens. So they do use a simulation model to predict what would happen in an experiment you can't run.

So let's use our model here to do some predictions. So here I've taken the same program, I've called it FitData2, but what I've done is I've added a point. So instead of just looking at the x values, I'm looking at something I'm calling extended x, where I've added a weight of 1 and a 1/2 kilos to the spring just to see what would happen, what the model would predict. And other than that, everything is the same.

Oops, what's happened here? Probably shouldn't be computing k here with a non-linear model. All right, why is it not? Come on, there it is. And now we have to un-comment this out, un-comment this. Well it fit the existing data pretty darn well, but it has a very strange prediction here. If you think about our experiment, it's predicting not only that the spring stopped stretching, but that it goes to above where it started. Highly unlikely in a physical world. So what we see here is that while I can easily fit a curve to the data, it fits it beautifully, it turns out to have very bad predictive value.

What's going on here? Well, I started this whole endeavor under an assumption that there was some theory about springs, Hooke's law, and that it should be a linear model. Just because my data maybe didn't fit that theory, doesn't mean I should just fit an arbitrary curve and see what happens. It is the case that if you're willing to get a high enough degree polynomial, you can get a pretty good fit to almost any data. But that doesn't prove anything. It's not useful. It's one of the reasons why when I read papers I always like to see the raw data. I hate it when I read a technical paper and it just shows me the curve that they fit to the data, rather than the data, because it's easy to get to the wrong place here.

So let's for the moment ignore the curves and look at the raw data. What do we see here about the raw data? Well, it looks like at the end it's flattening out. Well, that violates Hooke's law, which says I should have a linear relationship. Suddenly it stopped being linear. Have we violated Hooke's law? Have I done something so

strange that maybe I should just give up on this experiment? What's the deal here? So, does this data contradict Hooke's law? Let me ask that question. Yes or no? Who says no?

AUDIENCE: Hooke's law applies only for small displacements.

PROFESSOR: Well, not necessarily small. But only up to an elastic limit.

AUDIENCE: Which is in the scheme of inifinitely small.

PROFESSOR: Compared to infinity [INAUDIBLE].

AUDIENCE: Yes, sorry, up to the limit where the linearity breaks down.

PROFESSOR: Exactly right. Oh, I overthrew my hand here.

AUDIENCE: I'll get it.

PROFESSOR: Pick it up on your way out. Exactly, it doesn't. It just says, probably I exceeded the elastic limit of my spring in this experiment.

Well now, let's go back and let's go back to our original code and see what happens if I discard the last six points, where it's flattened out. The points that seem to be where I've exceeded the limit. So I can easily do that. Do this little coding hack. It's so much easier to do experiments with code than with physical objects. Now let's see what we get. Well, we get something that's visually a much better fit. And we get a very different value of k. So we're a lot happier here. And if I fit cubic to this you would find that the cubic and the line actually look a lot alike.

So this is a good thing, I guess. On the other hand, how do we know which line is a better representation of physical reality, a better model. After all, I could delete all the points except any two and then I would get a line that was a perfect fit, R-squared -- you know the mean squared error -- would be 0, right? Because you can fit a line to any two points. So again, we're seeing that we have a question here that can't be answered by statistics. It's not just a question of how good my fit is. I have to go back to the theory. And what my theory tells me is that it should be linear, and

I have a theoretical justification of discarding those last six points. It's plausible that I exceeded the limit.

I don't have a theoretical justification of deleting six arbitrary points somewhere in the middle that I didn't happen to like because they didn't fit the data. So again, the theme that I'm getting to is this interplay between physical reality,-- in this case the experiment-- the theoretical model,-- in this case Hooke's law-- and my computational model, -- the line I fit to the experimental data.

OK, let's continue down this path and I want to look at another experiment, also with a spring but this is a different spring. Maybe I'll bring in that spring in the next lecture and show it to you. This spring is a bow and arrow. Actually the bow is the spring. Anyone here ever shot a bow and arrow? Well what you know is the bow has the limbs in it. And when you pull back the string, you are putting force in the limbs, which are essentially a spring. And when you release the spring goes back to the place it wants to be and fires the projectile on some trajectory.

I now am interested in looking at the trajectory followed by such a projectile. This, by the way, is where a lot of this math came from. People were looking at projectiles, not typically of bows, but of artillery shells, where the force there was the force of some chemical reaction.

OK, so once again I've got some data. In a file, similar kind of format. And I'm going to read that data in and plot it. So let's do that. So I'm going to get my trajectory data. The way I did this, by the way, is I actually did this experiment. I fired four arrows from different distances and measured the mean height of the four. So I'm getting at heights 1, 2, 3, and 4. Again, don't worry about this. And then I'm going to try some fits. And let's see what we get here.

So I got my data inches from launch point, and inches above launch point. And then I fit a line to it. And you can see there's a little point way down here in the corner. The launch point and the target were at actually the same height for this experiment. And not surprisingly, the bow was angled up, I guess, the arrow went up, and then it came down, and ended up in the target. I fit a line to it. That's the

best line I can fit to these points. Well, it's not real good.

So let's pretend I didn't know anything about projectiles. I can now use computation to try and understand the theory. Assume I didn't know the theory. And what the theory tells me here, or what the computation tells me, the theory that the arrow travels in a straight line is not a very good one. All right, this does not actually conform at all to the data, I probably should reject this theory that says the arrow goes straight.

If you looked at the arrows, by the way, in a short distance it would kind of look to your eyes like it was actually going straight. But in fact, physics tells us it can't and the model tells us it didn't.

All right let's try a different one. Let's compare the linear fit to a quadratic fit. So now I'm using polyfit with a degree of 2. See what we get here. Well our eyes tell us it's not a perfect fit, but it's a lot better fit, right. So this is suggesting that maybe the arrow is traveling in a parabola, rather than a straight line.

The next question is, our eyes tell us it's better. How much better? How do we go about measuring which fit is better? Recall that we started by saying what polyfit is doing is minimizing the mean square error. So one way to compare two fits would be to say what's the mean square error of the line? What's the mean square error of the parabola? Well, pretty clear it's going to be smaller for the parabola. So that would tell us OK it is a better fit. And in fact computing the mean square error is a good way to compare the fit of two different curves.

On the other hand, it's not particularly useful for telling us the goodness of the fit in absolute terms. So I can tell you that the parabola is better than the line, but in some sense mean square error can't be used to tell me how good it is in an absolute sense. Why is that so? It's because mean square error -- there's a lower bound 0, but there's no upper bound. It can go arbitrarily high. And that is not so good for something where we're trying to measure things.

So instead, what we typically use is something called the coefficient of

determination. Usually written, for reasons you'll see shortly, as r squared. So the coefficient of determination, R-squared, is equal to 1 minus the estimated error EE over MV, which is the variance in the measured data. So we're comparing the ratio of the estimated error, our best estimate of the error, and a measurement of how variable the data is to start with.

As we'll see, this value is always less than 1, less than or equal to 1, and therefore R-squared is always going to be between 0 and 1. Which gives us a nice way of thinking about it in an absolute sense. All right, so where are these values? How do we compute them? Well, I'm going to explain it the easiest way I know, which is by showing you the code. So I have the measured values and the estimated values. The estimated error is going to be-- I take estimated value, the value given me by the model, subtract the measured value, and square it and then I just sum them.

All right, this is like what we looked at for the mean square error, but I'm not computing the mean, right. I'm getting the total of the estimated errors. I can then get the measured mean, which is the measured sum, divided by the length of the measurement. That gives me the mean of the measured data. And then my measured variance is going to be the mean of the measured data minus each point of the measured data squared, and then summing that.

So just as we looked at before when we looked at the coefficient of variation, and standard deviation, by comparing how far things stray from the mean, that tells us how much variance there is in the data. And then I'll return 1 minus that. OK, Tuesday we'll go look at this in more detail. Thank you.