PROFESSOR: All right, I think we're gathered. So let's get going. We were going to DAEs today, but we decided to postpone for one day and do a little catch up on different things that you should know. And I thought I'd first back up a minute and recall what you've learned so far. So we started out talking a lot about linear algebra and, particularly, the problems of this type where you're given a matrix and you're given a vector, and you want to figure out what the x is that makes this equation true.

And you've become extremely familiar with the convenient Matlab backslash function. You've probably used this a lot of times. And if you remember, this is one of the only problems where we often have a unique solution. We know a solution exists. And we can do a finite number of steps in order to get the solution. And actually, the method we have is really good. So it almost always gives us the solution and to machine accuracy. So this is a brilliant method. And so everything is based on this in our whole field, actually.

So we learned this first. We learned it takes n cubed operations in order to get the solution, where n is the size of the vectors. And when we do the solution, the x we get for backslash, invariably, it either is a solution, solves this equation to machine accuracy, in the sense that you multiply m times x, you get b to as many digits as you can read.

However, we talked about how m could be poorly conditioned so that there might be multiple x's that would solve this equation to machine accuracy. And so that's the problem with ill conditioning. So it's not that it doesn't solve the equation. It's that we're not really sure that's the right x. You could get a bunch of different x's that, basically, solve it as well as we can tell. All right, so do you guys remember this?

All right, and so that also implies that if you make a very small change in b, you might get a gigantic change in x. Because even for one b, you have a quite a range of x's that could work, all right. Then we try to solve problems like this. Right, systems of equations, not just linear ones, but nonlinear ones as well. And what we decided to do as one of our best methods is

Newton-Raphson, where we keep on doing j backslash f [INAUDIBLE] negative f.

Right, so all we did in this method, a lot of times, is we just kept on calling this method over and over again. All right, so this took order of n cubes. This thing is going to take order of n, I don't know what you call it, steps or something. Maybe if you guys have a word for it. What do you call the iterations? Iterations. So it takes the number of iterations times n cubed.

Now, if you really can make Newton-Raphson work, and you have a really great initial guess, the number of iterations is really tiny. Because Newton-Raphson is such a fantastically great convergence. However, we're usually not such good guessers. And so it might take a lot of iterations. And in fact, we use the dogleg method to solve this as we've discussed about where you mix up different methods. And you have a trust region and all this baloney. And so it might actually be quite a lot of iterations to get there.

And then we went and tried to solve this problem. And what we decided to do for both of these was almost do the same thing. So I said, well, we can really solve this one by instead solving this. Right, that's really what we ended up doing.

That's what happens inside fsolve is it converts the problem of finding f of x equals 0 into a minimization problem to try to make the norm of f as small as possible. Because the answer will be when the norm is zero. And if f of x equals 0 has a solution, then the global minimum here is a solution, as long as your initial guess is good enough that you're in the valley that leads to the true minimum. Then this will work. And that's actually what fsolve does.

So these two guys get kind of coupled together, the minimization problem and the root-finding problem, or the problem of solving systems of non-linear equations. And in both of them, we're going to call the first method a million times. OK, so that's the way we're working so far. And actually, when you do fmincon, which was the next level, then you actually end up looking for saddle points, which, again, we look for as minimizing the norm of a gradient squared, which gets it back into this kind of form. And so they're all the same thing.

And you can, again, work out estimates of how much effort it takes. And it's something times n cubed, Typically, if most your work is this. Now, I should make a comment that, in practice, that's not always true. So in practice, sometimes, although the highest order term is n cubed, the linear algebra programs that do this are so great that the prefactor is tiny.

So in order to get the limit where the n cubed term was actually dominate the CPU time, you

had to have a gigantic n. And in fact, the limit, it's often the effort to construct the matrix j. And so it really goes as n squared. So although this nominally is like n cubed, in practice, a lot of times, it's n squared. And I say it for these other methods.

And then a lot of them get into how good the initial guesses are. Because the number of iterations can get crazy. And in fact, maybe, it never converges if your initial guess is bad. So that can be a really critical thing. [INAUDIBLE] is better. Because you can't change n I mean, how many unknowns you've got is how many unknowns you've got. But you can change the number iterations by, say, more clever initial guessing. And that was the whole concept about continuation and homotopy and all this kind of stuff was trying to help reduce this number.

Last time, we talked about implicitly solving ODEs. And so we had an update formula. So we were solving an ODE. The initial value problem, y of-- Right, and we said that, first, we went through the explicit methods. And they were perfectly straightforward to do. And you don't have to solve any systems of nonlinear equations to do them. And so that was great.

But then we pointed out that, a lot of times, they're numerically unstable. And so then you have to instead use implicit methods. But the implicit methods generally work this way. They have your new value of y. What you're trying to compute is your current value of y plus delta t sums g. It depends on the size of delta t and y new. Maybe y old also.

And these g's are things like the Crank-Nicolson, which you're going to work on in the homework, and we show it in class too. And so there's different update formulas of people suggesting how to do this. The trick of this is that the y new is inside the function. So therefore, this is really another problem like we had before. You can really rewrite this that way by just putting this on the one side. Put them all on one side of zero. And now, I'm trying to figure out what y new is. And I'm going to solve that using this method. And I'm going to solve that by using this method.

All right, so now, how is the scaling going to go? At each time step, I have to solve one of these guys. How much effort did that take? We did that, right. Here it is, n [? after ?] times n squared. And then how many times do I have to do it?

AUDIENCE: It's h times 7 times [INAUDIBLE].

PROFESSOR: Right it depends on how many times steps I have [INAUDIBLE] my delta t. So it's something like t final minus t 0 over delta t. That's the number of times steps. But I have constant delta t.

Times the number of iterations that it takes to do the nonlinear solve times n squared, which is a cost of forming the Jacobian matrix.

Now, this can be pretty large. So as we talked about, this might be as big 10 to the eighth in a bad case. Certainly be less than 10 to the eighth because otherwise, you'll have numerical problems. Maybe, it's really going to be 10 to the sixth for a real problem. So you have a million time steps. At each times step, you have to solve the number of iter-- the solve costs this much. Number of iterations might be what? How many iterations do you guys take to solve nonlinear equations? You solve a lot of them now. How many iterations does it take?

AUDIENCE: 10.

PROFESSOR: 10, does that 10 sound right?

AUDIENCE: Yeah, [INAUDIBLE].

PROFESSOR: You've got an initial guess 10. If you have a medium initial guess, maybe 100. It it's more than 100, you're not going to converge, right?

AUDIENCE: Yeah.

PROFESSOR: Yeah, OK. So this is order of 10 maybe. So the million 10 and then n squared just depends on how big your problem is. So in my group, the problems we typically do, n is approximately 200. So this is like 200 squared. And then you can see that the number of iterations, and this is maybe a little bit overestimated. Maybe I can get away with 10 to the fifth, something like that. But you can see it starts to get pretty big.

But for writing it, as you guys are writing software, it's not that big a deal. So I can assign it as part 1 of a three-part homework problem. It's the right ODE solver. It solves implicit equations. And you can write it. And the reason you can is because you've already written methods that solve this. Actually, Matlab did it for you. But you write it yourself. I assigned to the [? 10:10 ?] students. So you write the [? backsub ?] solution to solve the equations.

And we we've already wrestled this. And I think you guys wrote one of these yourself. And also, we can use fsolve, which is just another little bit better implementation of the same thing. And then now, you write your next level. So you can see you're doing a composite process, where you're doing something at the top level. It's calling something at a lower level. That's calling something at a lower level.

And then what you really have to be concerned about if you're worried about CPU time or whether the thing is going to solve before the homework is due, is how big does this get. How many operations are you really asking the computer to do? Now, remember the computer is fast. Right, so it can do a gigahertz or something. So you can do like 10 to the eighth somethings per second.

And what should I say about this? I have not included, but I should that it's-- well, I'm assuming here that the function evaluation is sort of order n of how many variables they have. OK, sometimes, function evaluations are much more difficult than that, primarily because, sometimes, they're actually including further nested stuff inside. So your f is actually coming from some really complicated calculation itself. And then you have another big giant factor of how much it costs to do f. But on this right here, I'm just assuming it goes as n.

So actually, let's just multiply this out. So this is what, 10 to the fourth, 10 to the 10th. So this is 10 to the 10th. 10 to the 10th kind of flops. But your computer does 4 times 10 to the ninth per second. Is that right? Is that [? what they ?] told us, four gigahertz? Yeah, so is this really not bad with just a few seconds on your computer. So that's why you can do these homework problems, and you still get them done. And if you have even [? porous ?] limitations, it takes 1,000 times longer. Still, 1,000 [INAUDIBLE] take 20 minutes on your computer.

So you're OK, but you can see we're starting to get up there. And it's not that hard to really make it a big problem if you do something wrong. And just a notation thing, a lot of these things we're doing are writing software that takes as input not just numbers or vectors. So those are things that we call functions. Right, things that take a number of vectors and input and give you, say, a function, a number of vectors and output.

But oftentimes, we actually want to take the names of functions as inputs. So for example, the root-finding problem, it takes as input what is f. Right, you have to tell it the name of a function for fsolve to be able to call it. So fsolve is an object that takes as input the name of a function. And things that do that are called functionals

So for example, this one is x is equal to this root-finding functional, fsolve. And I'm going to write with square brackets. It depends on f. That's it, right? [INAUDIBLE] f. So you give it the function f. And if fsolve's good, it should be able to tell you what the x is. Now, in reality, we help it out. So we give it x 0 [? 2 ?] just give it a better chance.

I write the square brackets just indicate that one of the things inside is a function, not just numbers. We do this a lot. So you guys have done this since you were kids. So derivatives Yeah, ddx of f. So we have, I don't know, for example, grad of f. That's a functional, the grad. The symbol means the operator, the functional, it operates on functions. You could write a general gradient function that takes any scalar valued function and computes the gradients with respect to the inputs.

This week, we give you one that computes a Jacobian. Give it any f of x. It gives you the Jacobian of f with respect to x. So that's a functional. So anyway, we do a lot with functionals. You've been using them all the time. I'm just trying get you to think about abstractly. And then there's a whole math about functionals. And there's a whole thing called calculus of variations. It's all about, if the objects you're worrying about are functionals, not functions, then you have another whole bunch of theorems and stuff you can do with that, which you'll probably end up doing before you get out of this place. But I'm not going to do that right now.

I would comment that one kind of functional that's particularly important to a lot of people in your research is the fact that the energy of any molecule is a functional of the electron density. And this is the basis of what's called density functional theory.

And so this is why a lot of people, you and many of you included, are going to end up using computer programs that are trying to find what the electron density shape is, things like molecular orbitals and stuff. You've probably heard these before. And then there's the functional that gives the energy of the molecule. And what you care about, mostly, is energy, because that connects to all your thermo. You're doing [? 1040, ?] it's all about the energies. And once you know all the energies, you can actually compute the entropies and all kinds of stuff too.

And so this is one application that's really important. It was a little surprising this was true and able to be proven to be true. And so the guy who did that got the Nobel Prize-- his name was Cohen-- a few years ago.

Actually, in liquid phase stuff, you work with Professor Blankschtein. They also have a version of this for properties of solution. And then where it's not the electron density, but it's actually the density of the material in the solution phase. Maybe Professor [? Swan ?] too. You guys do that? Yeah, so if you work with Professor [? Swan, ?] you might learn about that too.

So that's one where the actual functional is like a big focus of the entire project. And the key is

that this is a function. The density is a function of the position. And then the functional converts that into a number. That's all page one here.

Now, I'm going to sort of change topics, but it's related. It's about the connections between numerical integration, interpolation, and basis functions. These things are all intimately tied up together. So I'll try to show you. So we try to solve a numerical integral.

And then let's look at what we're really doing. When we're doing this, we typically only do a few function evaluations. So we have a few points where we know the numbers f of tn. We've picked a few t's. We evaluated f. And from those few points, that small number of function evaluations, we want to get a good estimate of this whole integral.

Now, it's a little goofy, actually, if you think about what we're trying to do. So suppose you're trying to do an integral from negative 1 to 1. We have some function that has a point here, a point here, a point here. And just for concreteness, let's say this is 1 over 26. And this is 1. And this is 1 over 26. And I have a particular function in mind that has those three points.

Now, the first thing to keep in mind is there's an infinite number of functions that go through those three points. Now, what we've been doing is we've been fitting some function to these points and then integrating that function that we fitted. So when you're in high school, you did the rectangle rule and what the function was used was this and this. And you knew the integral of this function and the integral of this function. And you added them up. And you decided from the rectangle rule that the integral was equal to 1 and 1/26.

But then some other people in high school are a little smarter. And they said it's not so smart [INAUDIBLE]. This thing was symmetrical to begin with. And now, I'm fitting it with a function that's wildly unsymmetrical. So let's instead not evaluate those points. Let's instead evaluate the midpoints. And maybe we find out that the midpoint values are here and here. And so I would assume that the function is just a flat line. That looks not so good either, because it doesn't get through these points.

But anyway, for the two points I knew, if I only used those two points, that would be the midpoint method. And I would get the I midpoint method. And I'd get something. And I did it for this particular function I have in mind. And it was 8 over 29.

And then some other people say, well, let's use the trapezoid rule. So let's approximate the function is this over the interval, which looks kind of sensible. And if you do that, it turns out

you get the same numbers as you do with the rectangle rule. So I trapezoid.

Now, let me just as a warning, if you just do the rectangle rule and then the trapezoid rule, and you got the same number both times, you might be tempted to conclude you have converged. And you have the true solution, because you did two methods of really different orders, and you got exactly the same number. And you're super happy, and you're ready to publish your paper.

But of course, the function might not look like this. So the particular function that I always had in mind was f of t is equal to 1 over 1 plus 25 t squared. OK, that was just the function I had in mind. And the way that function looks like is like that. And the real integral of that function, which you guys if you remember how do it from high school, you can actually evaluate that integral. It's nothing to do with either 8/29 or 1 in 126. It's something completely different.

Now, those of you who went to really advanced high schools, you also learned Simpson's rule too. And Simpson's rule says let's fit it to a parabola. And so that would give you something like this. And if you do Simpson's rule, I ended up getting this if I did the arithmetic right. 1 and 1/3 plus 2/78. Who knew? OK, which also has little relation to the true value of the integral.

Now, if you're smart, and you did trapezoid and then I it to Simpson's, then you say, oh my goodness, these are not really that similar. And so I'm not converged, and I'd better do something else. So I just want you to be aware when we do these rules, what we're actually doing is we're fitting our points to some continuous function that we know how to integrate. And then we do the analytical integral of that function, which is a fine thing to do.

But just be aware that it's like extrapolation. It's like we're imagining what the function is doing between these points, when we actually haven't tested it. Now, we could. We could just call more points. We could calculate. But that would cost us more function evals. And we're trying to save our computer time. So we want to finish off before the TG so we can go buy some beer. And so we have to take some choices about what to do. And we can't do an infinite number of points for sure.

All right, so in all these methods, what we're doing is we're approximating our true f with some approximate f, which is equal to a sum of some basis functions like this. So we pick some basis functions. We know how to integrate. We fit, somehow determine what these c's are. And then we decide that the integral is equal to the integral of f tilde, not the actual original f.

And then that's actually going to be the sum of ck times the integral. And we carefully chose basis functions that we don't how to do this integral; analytically. So we just plug that in, and we just have to figure out that the c's are to get this to work. OK, so that's what you really did when you're doing this trapezoid rule and Simpson's rule, and all that kind of stuff. And it's just different choices of what the basis functions were that you used.

Now, there's a lot of choices about how to do that kind of approximation to make f tildes that are sort of like f's with some limited amount of information. And it generally has to do with how many k's we have. So if k is less than n, then it's really an approximation. Because there's no way that we would expect that with just a few parameters, we could get a lot f of tn. So we would expect that f tilde of tn is not equal to f of tn, at least not at all the points. Because if n is-- if we have a lot of points and we only have a few parameters, we don't expect to be able to fit it perfectly.

So we know this is what we expect. It might happen if, by some luck, we chose a basis function which was actually the true function f, then it would actually fit perfectly. But most of the time, not. By Murphy's Law, in fact, it's going to be completely wrong. But this is not such a bad idea. We may come back to this later.

Another possibility is to choose k equals to n. And then usually, we can force f tilde of tn to equal f of tn. So at the points that we have, if we have enough parameters equal to the number of points, then often, we can get an exact fit. We can force the function to go through all the points. And when we do this, this is called interpolation.

Now, when should you do the one or the other? When should I make k as big as n? Or when should I use k less than n? A really crucial thing is whether you really believe your points. If you really believe you know f of tn to a very, very high number of significant figures, then it make sense. You know that's true, so you should force your fitting function to actually match that.

However, if you get the f of tn, for example, from an experiment or from a stochastic simulation that has some noise in it, then you might not really want to do this. Because you'll be fitting the noise in the experiment as well. And then you might want to use a smaller number of parameters to make the fits. OK, so you can do either one. You're in charge. Right, you get the problem. If somebody tells you, please give me this integral, they don't care how you do it. They just want you to come back and give them the true value of the integral to some

significant figures. So it's up to you to decide what to do. So you can decide what to do.

What a lot of people do in the math world is this one, interpolation, where you're assuming that the f of tn's are known exactly. And so your function evaluating function is great. And in that case, you can write down this nice linear equation. So we're trying to make f of tn with a true function to be equal to ck phi n of t. Sorry, phi k of t. Which n in [INAUDIBLE] k is equal to n.

And I can rewrite this this way. This is like a set of numbers. So I can write as a vector. I'll call it y. And this thing has two indices on it, so it looks like a matrix. So I'll call that m. And c is my vector of unknown parameters. And wow, I know how to solve this one. So c is equal to m backslash y.

Now, for this to work and have a unique solution, I need m not to be singular. And what that means is that the columns of m have to be linearly independent. So what are the columns of m? They are phi k of t1, phi k of t2, phi k of tn. And I want that this column is not a sum of the other columns.

So this is kind of requirement for j not equal [? to k ?] This is kind of a requirement if I'm going to do this kind of procedure is I can't choose a set of basis functions that's going to have one of them be a linear combination of the other ones. Because then I'm going to get a singular matrix. My whole process is going to have a big problem right from the beginning. So I have to ensure that this is not true. Yeah, that this is true. These are not equal to some of them.

Another way to write that is to choose the the columns that phi to be orthogonal to each other. And so I can write it this way. OK, so I want them to be orthogonal to each other.

And then you can see how this kind of generalizes. If I had a lot of t's, this would sort of look like the integral of phi k of t, phi j of t is equal to 0. And so this gives the idea of orthogonal functions. So this is orthogonal functions, and my discrete point's tn. And this is the general concept of orthogonal functions.

And then when I define it this way, I need to actually decide my own a's and b's that I want to use. And if you use different ones, you actually have different rules about what's orthogonal to each other. And also, sometimes, people are fishy, and they [? spit ?] a little w of t in here just to do, because they have some special problem where this w keeps showing up or something, and they want to do that. But that's up to them. All right, we won't specify that.

But this is the general idea of making basis functions orthogonal to each other. And it's very

analogous to making vectors orthogonal to each other, which is what we did here. These are actually vectors. Right, it's just as the vector gets longer and longer, it gets more and more like the continuous function, and we have more t points. And eventually, it looks like that integral as you go to infinity.

So anyway, so a lot of times, people will choose functions which are designed to be orthogonal. And one way to do it is just to go with a continuous one to start. Now, we can choose any basis functions we want. Over here, we already chose a whole bunch. You guys have done this already. I don't know if you knew you were doing it. But you chose a lot of different basis functions completely different from each other. You can choose the different ones if you want.

And one popular choice is polynomials. But it's not the only choice. And in fact, a little bit later, we're going to, in the homework problem six, I think we're going to have a problem where you have to use a different kind of basis functions. You can use any basis functions you want. And there's kind of ones that match naturally to the problem.

However, people know a lot about polynomials. And so they've decided to use polynomials a lot. And so that's the most common choice for numerical integration is polynomials. One thing is polynomials are super easy to integrate, because that was the first thing you learned, right, how to do the intervals of polynomials. And also, they're easy to evaluate. It only takes n operations to evaluate a polynomial of nth order. And there's a ton of theorems, and things are known about polynomials. We have all kinds of special properties. And we have all kinds of convenient tools for handling them. So let's go with polynomials for now.

Now, I want to warn you, though, that how you order polynomials are really not very good fitting functions or interpolating functions for a lot of problems. And in particular, for this one I showed here where this is the function. Here, that function, you can try to fit that with higher and higher order polynomials. So I can have my function. The function value at 0 is 1. So the function looks kind of reasonable. Like that, symmetrical, better than I drew it.

And you can put some number of points in here. And then you can do an interpolating polynomial that goes through all the points. And if you do it for this one with five points, you'll get something that looks like this. It goes negative, comes up like this, goes negative again. So that's one you get if you use five points across this interval. If you use 10 points, then you get one that looks like this where the peak value here is two where as the max of the original

function's just one. And the original function's always positive, but they both go negative.

So even within the range of the interpolation, the polynomials can have wild swings. And so this first one was for a fifth order polynomial, or maybe fourth order, five points. And this one is for the next 10th order or 9th order polynomial. I'm not sure which one. Yes.

AUDIENCE: What points are these polynomials trying to fit? Or does it seem like [INAUDIBLE]?

PROFESSOR: So I didn't draw very well. When you guys go home, do this. Right, on Matlab it will take you a minute to solve this equation where you put in the function values here. And put in the polynomial values that [INAUDIBLE] here for the monomials, t to the first power, to the second power, to the third power, like that. And just do it, and you can get the [? pot ?] yourself.

And just to convince yourself about the problem, and this is very important to keep in mind. Because we're so used to using polynomials that think they're the solution to everything. And Excel does them great. But it's not this solution to everything. Nonetheless, we're going to plow on and keep using them for a few minutes.

So one possibility when I'm choosing this basis set is I could choose what are called monomials, which is phi k is equal to x to the k minus 1. And this one is super simple. And so a lot of times it satisfies this orthogonality thing, which I showed you before. But it's a really bad choice usually.

And what it is is that the matrix you have to solve for this interpolation, if you have evenly-spaced points and you have x to the k, it turns out that this matrix usually has a condition number that grows exponentially with the number of terms in your basis, the number of how many monomials you include. So condition numbers that grow exponentially is really bad idea. So this is not what people use. So you might be tempted to do this, and maybe you did it once in your life. I won't blame you if you did. But it's not a really smart idea.

So instead, a lot of the mathematicians of the 1700s and 1800s spent a lot of time worrying about this. And they'd end up deciding to use other polynomials that have people's names on them. So a guy named Lagrange, the same guy who did the multiplier. He suggested some polynomials. They're kind of complicated. They're given in the book.

But what they really are is I'm taking the monomial basis set, and I'm making a new basis set. So I want phi l to be equal to the sum dlk of phi k, where this phi k it the monomials. So really, you can choose any d you want. You're just taking linear combinations of the original

monomial basis set to make up polynomials. And if you cleverly choose the d, you can choose it to have special properties.

So Lagrange gave a certain choice of the d that is brilliant in the sense that, when you solve the problem m times c is equal to y to try to do your interpolation, it turns out that the c's you want, ck, is equal to f of tk. So you don't have to do any work to solve what the c's are. It's just the f values directly. No chance for condition numbers, no problems, that's it. So that's the Lagrange polynomials.

Now, Newton had a different idea. So this is for Lagrange polynomials. If you choose a Lagrangian [? polynomial ?] basis, then this is it. And you do a square system. That's it. If you choose the Newton, even he got into this business too. And he made some polynomials. And he made ones that have a special property that if you solve it-- so he chose a different d. You still have to do some solve. You get the solution.

And then you decide that I don't like that interpretation. I want to put some more points in. I'm going to do a few more function evaluations. I'm going to add some more. So from the first solve, I have the values of c1, c2, c3, up to c the number of points I had initially.

And now, I want to add one more data point, or add one more f of tn, one more function evaluation. I want to figure out with c of n plus 1 is, what the next one is. It turns out that it's his polynomials. The values these c's don't change when I make this matrix a little bit bigger by adding one more y. And I add one more parameter. And I make one more row in the matrix, one more column in the matrix.

When I do that, it's a unique property that none of these other c's change. The solution is going to be the same, except for the n plus one [? level. ?] So then I can write a neat, cute little equation just for the n plus 1 one. I don't have to mess with the previous ones. And so I'm just adding a basis function that sort of polishes up my original fit. So that's a nice property too.

Whereas with the Lagrange ones, you have to recompute everything. And all the c's will change. Actually, the c's won't change. I take that back. You add points. The c's won't change, but the actual polynomials will change. The d will change. Because the d depends on the choice of the time points in a complicated way in Lagrange.

So now, you start to think, oh my goodness, I have all these choices, I can use Lagrange. I could use Newton. I could use monomials. Maybe, I should go back to bed. And you think,

well, could someone just tell me what the best one is? And I'll just do that one. I don't need to learn about all this historical stuff about development of polynomial mathematics. I'd just like you to just tell me what the answer is.

So Gauss, typically, figured out what the best on is. And he said, you know, if we're going to do integrals, we'd really like to just write our integrals this way. So our approximate value to be equal to some weight functions times the function evaluations at some times. Right, that's easy to evaluate. That would be good. If I pre-computed the weight functions [? and ?] the times, then I put any function f in here, I could just do this like zip. It's one dot product of the f of tn's I'll have the solution. So I want to do that.

So what's the best possible choices of the w's and the t's to make this really be close? Well, the problem is that there's an infinite number of functions that somebody might want to integrate. I can't do them all. So he said, let's just find the one that works for as many monomials as possible to the highest order, as far out as I can go. And it turns out I have nw's. I have nt's. So I have two n parameters that I can fiddle around with. And so I should get something like two n's in my polynomials, maybe 2n minus 1.

So what I want is that I want to choose this so that for the monomial 1, and the monomial t, and the monomial t squared, that this i is actually going to be exact. So what I want is for the monomial 1, I want the summation of wn. My f for the monomial 1 is always 1. So it's times 1. I want this to equal the integral from a to b of 1dt, which is b minus a. OK, so that gives me one equation for the w's.

Then for the next one, I want the summation wn times tn to equal the integral of a to b of t. dt, which is equal to b squared over a number 2 minus a squared over 2 if I did my calculus correctly. And so there's another question. I have another question for these guys. And I keep going until I get enough equations that determine all the w's and c's. All right, so I do it for t squared. So I can wn tn squared is equal to [INAUDIBLE] b t squared dt, and so on, OK.

And so if you do this process, you get the weights and the points to use that will give you what's called the Gaussian quadrature rule. Now, the brilliant thing about this is that by this setup, it's going to be exact for integrating any polynomial up to order 2n minus 1. So if I choose three, here I'll be able to determine-- if I choose three points, I'll have six parameters. So I'll be able determine up to the 11-- 11th order polynomial would integrate exactly.

No, that's wrong. n's only 3. I have six parameters. I can't do this. 2 times 3 minus 1, what's

that? 5, there we go. To the fifth order polynomial integrate exactly, I'll have an [? error ?] for sixth order polynomial.

So this is a rule. So before, if I gave you three points like here, most of you probably would use Simpson's rule. Fit this to a parabola. It has three parameters, a, ax squared plus bx plus c, right, for a second order polynomial. I would fit it. Then I'd integrate that. That would be the normal thing that a lot of people would do. I don't know if you would do that. But a lot of people would do that. So that would be that what you would get.

And what [? error ?] would you get? You'd have error that would be the scale of the width of the intervals to the fourth power. Right, because Simpson's rule exact up to the cubic. You guys remember this from high school? I don't know if you remember delta t to the fourth. Yeah, OK.

So Simpson's rule has an order of delta t to the fourth. But Gauss's method has order of delta t to the sixth. So this is for three points. In Gauss's method, you can extend it to as many points as you want. All right, so that's kind of nice. You only do three points, you get [? error ?] of the sixth order. So that seems pretty efficient. So people kept going.

And so the very, very popular one use, actually, seven points for Gauss. So you'd be able to get to the n minus 1. So you'd be exact for 13. So if you have delta t to the 14th power as your error, that's pretty good. And then what people often do is they'll add eight more points.

And they make another method that uses all these points. It's called [? Conrad's ?] method. He made another method that gives you the best possible solution if you had the 15 points, given that 7 of them are already fixed by the Gauss procedure. And so you can get the integral using Gauss's method and using [? Conrad's ?] method.

So you can compute the i using Gauss's method, the seven points, the i with the [? Conrad ?] method. I don't know how to spell his name, Actually I think it's like this. 15 points. And then you can take the difference between these two. And if these two are really similar, then you might think your integral's pretty good. Because they're pretty different methods, and if they get the same number, you're good. And people have done a lot of numerical studies in this particular case, because it's used a lot in actual practice.

And it turns out that the [? error ?] is almost always less than this quantity to the 3 Gauss power. So you have an error estimate. So you do 15 function evaluations. You get a pretty

accurate value for your integral, maybe, if you can be fit well with whatever order polynomial, 13th order polynomial. 13th order polynomial, yeah.

And so any function that can be fit pretty well with the 13th order polynomial, you should get pretty good value in the integral. And you can check by comparing to the [? Conrad ?] estimate, which uses a lot more points. And if two of these guys give the same number or pretty close, then you're pretty confident that you're good. And there's even a math proof about this particular one. So this is like really popular, and it's a lot better than Simpson's rule, but it's kind of complicated.

And part of it you have to recompute these tn's and wn's for the kind of problem you have. Yeah.

AUDIENCE: If you're using 15 points already, why not just split your step size smaller? And if you did that, how would the error [INAUDIBLE] the simplest? Use a trapezoid.

PROFESSOR: Trapezoid or Simpson's rule of something like that. Yes.

AUDIENCE: How would the error with those 15 steps using trapezoid compare to [? Conrad ?] and Gauss?

PROFESSOR: So we should probably do the math. So say we did Simpson's rule, and we have delta t, and we divide it by 15 or something. Then we can figure out what this is, but it's still going to have the scaling to the fourth power. So the prefactor will be really tiny, because it will be 1/15 to the fourth power.

But the scaling is bad. So that if I decide that my integral's not good enough, and I need to do more points as I double the number of points to get better accuracy, my thing won't improve that much. Because it's still only going to use the fourth power, whereas the Gauss' one say it's going use the sixth power if I only use the three points. Where we use 15 points, is going to get scale to some 16th power or 15th power or some really high power.

So for doing sequences, the Gauss' one is a lot better. However, what people do is they usually pre-solve the system of equations. Because this system of equation does not depend. When you do it with the monomials, it doesn't depend what f is. You can figure out the optimal t's and w's ahead of time. And so they [INAUDIBLE] tables of them depending on where you want to go. People recomputed them, and then you can get them that way.

Time is running out. I just want to jump up. Now, for single integrals, I didn't have to tell you

and of this. Because you could have just solved it with your ODE solver. So you can just go to ode15s or ode45, and you'll get good enough. It won't be as efficient as Gaussian. But who cares. You still get a good answer. You can report to your boss. It'll take you three minutes to call it e45 to integrate the one-dimensional integral.

The real challenge comes with the multidimensional integrals. So in a lot of problems, we have more than one variable we want to integrate over. And so if you have an integral, say, of a to b from the lower bound of x to upper bound of x. So this is dx, dy, f of xy. This is the kind of integral that you might need to do sometimes. OK, because you might have two variables that you want to integrate over.

And so how we do this, the best way to think about it, I think, is that f of x is equal to the integral of l of x u of x dy f of xy. And then, if I knew what that was, I would evaluate that say with a Gaussian quadrature or something. So I'd say that this i 2d is approximately equal to the sum of some weights-- I'll label them x just to remind you where they came from-- times f of xn. And I would use, say, Gaussian quadrature to figure out what the best value of the x's and the y's are to use.

And then I would actually evaluate this guy with a quadrature two. So I'd say f of xk or xn is equal to the integral from l of xn to u of xn dy f of xn y. And so this itself, I would give it another quadrature. This is going to be equal some other weights. nj f of xn yj. Is that all right?

And so this is nested inside this. So now, I have a double loop. So if I took me, I don't know, 15 points, 15 function evaluations to get a good number for my integral in one dimension, then it might take me 15 squared function evaluations to get a pretty good integral for two. 15 might be a little optimistic, so maybe 100 is more like if you really want a good number. So I need 101 dimension, 100 in the second dimension squared. Because I subdivided the integral into six equal pieces, say. And then now, so this is OK. So you can do this. It's only 10,000 function evaluations. No problem, you've got a good gigahertz computer.

Now, in reality, in a lot of them, we do have a lot more dimensions than that. So in my life, I do a lot of electronic structure integrals. We have integrals that are dx 1d y1 d z1 for the first electron interacting with a second. And then something that I'm trying to integrate. It depends on, say, the distance between electron one and electron two. OK, so this is actually six integrals.

And so instead of being a 2 in the exponent, this is going to be like a 6 in the exponent. So

now, this is 10 to the 12th. That means that just evaluating one integral this way might take me 100 seconds if I can evaluate the functions like that. But in fact, the functions I want to evaluate here, these guys will take multiple operations to evaluate the function. And so this turns out I can't do. I can't even do one integral this way with its six integrals deep.

And so this is the general problem of multidimensional integration. It's called the curse of dimensionality. Each time you add one more integral, you're nested there. All of a sudden, the effort goes up tremendously and more, a couple of orders of magnitude more. And so we'll come back later in the class about how to deal with cases like this when you have high dimensionality cases to try to get better scaling. Instead of having it being exponential, it was really 100 to the n right now, 100 to the n dimensions. [? Can we ?] figure out some better way to do it. All right, thank you.