PROFESSOR: OK. We can go ahead and get started. So today is going to be the first lecture on an introduction to design optimization. You guys ready? Can you talk about the project? I have office hours right after this, so we can--

So here are three things I'm going to try to cover today. We'll talk about the basics of the design optimization problem, how you take a design problem and set it up as an optimization problem. Well talk about unconstrained optimization methods and how to compute the gradients that we will need for those optimization methods. And then in the next lecture, we'll talk a little more about some of the different methods. We'll maybe talk about some responsiveness modeling, so surrogate models and see what else we have time for.

So this is intended to be a little bit of just kind of a teaser for optimization methods. This is something that I use a lot in my research. So let's just start off with thinking about the basics of writing a design problem as an optimization statement.

So the idea is that a design problem, as I said already, can be written as an optimization problem. And the reason that we're doing this is because then we'll be able to use numerical optimization methods to explore the design space. So what we saw in the last lecture were the design of experiments, which were basically sampling methods.

But what we saw is that if you have a lot of design variables, it get's hard to explore the design space very much. Because you can't [INAUDIBLE] to maybe too many levels. So one thing we can do is try to write the design problem as an optimization problem.

And in doing that, we would define a few mathematical elements. So the first are what are called objectives or objective functions sometimes. And these are what we are trying to achieve, so measures of performance of the design. We'll talk about some examples in a second.

We're going to have constraints. And constraints are going to be things that we can't violate, so requirements on the design are cannot be violated. So these are going to be what we cannot violate.

We will have design variables. And those are the quantities that we can change. Those are the things that as a designer we have to make decisions about. The design variable are what we can change, what we can change and what we can control.

And lastly, we're going to have parameters. And the word parameters is used often in different contexts. Sometimes people use the word parameter when they mean design variable. The way that I'll use it is that a parameter is going to be other quantities that are taking on fixed value.

So they're other quantities that affect the objective and the constraints. But they're going to take on fixed values. So they're not going to be varying as we search the design space.

OK. That's the four elements of the optimization problem, objective functions, constraints, design variables, and parameters. And I'm going to go through each of these. And we'll define it mathematically and also think about what it might be in terms of physical examples.

So let's start off thinking about design variables. So again, the design variables, these are the qualities or the attributes of the design that we have control over as a designer, things that we can change. And we're going to write the design variables in a vector, the design variable vector or the design vector.

We're going to give it the symbol x. I'll put a vector symbol above it just to denote that it's a vector. And it's going to contain n design variables, n dvs, that form the design space. So when we talk about the design space, we're usually talking about the space that's defined by these quantities in the design vector x, so n design variables that form the design space.

And so mathematically, the vector x will be x1, x2, down to x10. OK. And the idea is going to be that as we run the optimization algorithm, it's going to be searching over different values of x. So this is what's going to be changing.

And what you'll see is that different optimization methods use different rules and different information to figure out how to move around in the design space. OK. So what are some examples? So if we think about aircraft conceptual design, say, what would be examples of physical quantities that might be in the vector x?

What? Weight. Weight normally would not be a design variable, because weight is not usually something that you can control directly. It's something that typically would be computed as a function of other things that would be designed variables. So weight is going to often show up as an objective.

AUDIENCE: Wingspan.

PROFESSOR: Wingspan. Yeah. So wingspan would be one. What else? What is it? Number of rotors?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Oh, motors, number of engines? Motors?

AUDIENCE: [INAUDIBLE]

PROFESSOR: OK. Sure, number of motors or engines. Yup. What else?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Payload size? Yeah. Payload weight or payload size or payload number of passengers.

AUDIENCE: Color?

PROFESSOR: Color? Who said color? OK. I'm going to spell it with a U just to have my objection. Yeah. [INAUDIBLE] mark number whatever. OK. So all the things that I think you can directly control and make a decision about. And what you notice is that these quantities can be of different kinds.

So wingspan is going to be a continuous real variable that can take on values between 0 and whatever upper limit you set to it. Whereas, number of motors is a discrete variable, right? It can be 1 or 2 or 3, but it can't be 2.5 or 2.53.

And as we talk about the optimization methods, you'll see that particularly discrete or integer variables cause a lot of problems. It's much harder to optimize a problem that's got something like number of engines or number of motors in it. OK. So will be your design vector x.

Next, let's talk about objectives or objective functions. So the objectives can be a vector as well. And it can be a vector j. And I'll put the vector symbol on there to denote when I'm talking about a vector. And it's going to be j of v system responses or characteristics.

And they're going to be responses or characteristics that we're trying to either minimize or maximize. OK. All right, so these are going to be measures of performance, costs, schedule, anything that we might want to push either as high as possible or as low as possible in making the decisions about our designs. So what might be some examples, again?

For the aircraft conceptual design example, what will be objective functions? Range. Yup. Range might be an objective if you wanted to maximize range. What is it? Speed. Yup, speed sometimes. Fuel consumption, fuel burn, yup.

So weight often shows up as an objective. Max take off weight is often used as a target for costs and fuel burn and everything kind of rolled up. Cost might be another one, operating costs, or it might be entire lifecycle cost. Could be environmental impact, could be noise or different kinds of things.

And so I most-- this is right here. So we would write the vector j as being j1, j2, jz if we had z objectives. Turns out that most optimization algorithms work with a single objective, with a single scalar objective.

There are some ways to do multi-objective optimization if you have more than one objective and you want to look for designs that are at the same time trying to maximize range and minimize costs, say. There are ways to do it. But most optimization methods work with a single scalar objective.

And if that's the case, then the way that you would normally proceed would be by weighting the different objectives. So you would have weight 1 times j1 plus weight 2 times j2 and so on. So you would roll all of the different objectives up, weight them, add them together, and get a single objective function that would then use in your optimization algorithm.

And these weights would represent both some kind of a normalizing factor, because range is going to be measured in meters. And that's going to be in the order of thousands. Whereas, speed would be measured maybe in meters per second or might be in the order of hundreds. Costs could be in dollars.

So these things have different units and different scales. But you could also use these weights to talk about your design of preference. I care more about cost than I do about noise. Or I care more about fuel burn than I do about noise. And you could use the weights in that way.

OK. So if we have multiple objective, either we have to roll them up into a scalar objective and you think about what these weights would be or there are some methods to do what's called multi-objective optimization. I'll add one more to the left. This is Professor [INAUDIBLE] preferred-- does he talk about [INAUDIBLE]?

I think it's payload fuel efficiency-- no, payload fuel energy intensity. It's a measure of fuel burned per person traveled 1 nautical mile or 1kilometer. So it's a measure of how far you fly, how many people you transport, how much fuel you burn, measure of efficiency.

OK. So design variables and objectives, design variables are what we can change. The objectives are going to be the measures of how good the design is that give us some direction for change. Then we're also going to have constraints.

Again, we think constraints are what we cannot violate. So these are going to act as the boundaries of the design space. So here the design space is defined by those design variables.

So a nice visual picture that I like to have in mind is that the design space is like a landscape. So think of your landscape. It's got hills, mountains. It's got valleys. It's got flat regions.

The dimensions of the landscape are the design variables. So you can think about two design variable, x1 and x2. And then the height of the landscape is the objective function. Right? So the mountains, the hills, that's places where the objective is high. And the places where you have valleys, that's where the objective is low.

And so if you try to minimize, say, cost, good designs you'd be looking for them in the valley. And the regions where the space is flat, those are regions where changing the design doesn't really change the cost or doesn't change the weight or whatever the objective is. So then what are the constraints?

The constraints are going to come in and tell you where in the landscape are you allowed to be. So there's going to be boundaries that say you can't go on the side of the boundary. You have to stay within this region. It's going to define regions where designs are allowable, where they satisfy different kinds of constraints. And it's going to tell you regions where your designs are not allowable.

And what kind of constraints can you get? We're going to have inequality constraints, which we'll write as gj is less than or equal to 0, for j for 1 to m1. So we'll have m1 inequality constraint. And the constraint is written that g, gj for the j constraint has to be less than or equal to 0.

So this might be a constraint that's something like [INAUDIBLE] speed. You need a constraint on [INAUDIBLE] speed. And you don't require the speed to be anything in particular. But you've got to make sure that you stay above the constraint. You would bring everything over to the left-hand side and make the constraint less than equal to 0.

Or if you had a constraint on cost, you would say that the cost minus the maximum cost you can incur had to be less than equal 0. So inequality constraints often come when there's some kind of limitation on the resources that you have or there's some kind of physical limitation in terms of the physics.

And then we might also have equality constraints. And we're going to give those the symbol h. So we'll write hk equal to 0. And in general, we'll have m2 equality constraint. So what's an example of an equality constraint in an aircraft design problem?

Lift equals weight, yeah. So that's kind of the classic one that shows up in the aircraft design problem, lift equals weight or left minus weight equals 0. If you had sophisticated-- like, if you have structural models in there and you're using a finite element analysis, then the governing equation of the finite element analysis would also show up here, the laws of whatever the PDE that you discretized. Or if there's a CFD model or a CFD equation, it's the conservation of mass, momentum, energy, and so on.

So often, we can have lots and lots of equality constraints. So you have inequality constraints and we have equality constraints. And then we can also have design variable bounds. And even though these things are kind of like inequality constraints, they're usually separated out, because they're quite often treated differently.

And the design variable bounds for design variable xi-- remember, our design vector was x1 x1 down to xn. We might bound design variable xi by a lower bound and by an upper bound, right? So for example, wingspan was one of the design variables. I mean, the lower bound could be 0. But it might even be bigger than 0.

And the upper bound might be the limit that we know exists, because [INAUDIBLE] in the [INAUDIBLE] constraints, the 80 meter box for a big aircraft. Or it may be that we know we want designs that are going to be between 40 and 50, and so we're going to put those bounds on so we don't waste time searching for designs elsewhere. So we may have lower and upper bounds for some or all of our design variables.

OK. So now, we have all the pieces. I'm going to call the parameters p. What are things that could be parameters? Material properties could be parameters.

Sometimes speed is actually a parameter. Often, aircraft are designed with speed being fixed and not something that you can vary. So often mach number might show up as a parameter-- anything that's going to go in the problem that's going to affect the constraints or affect the objectives, but is not allowed to vary.

So putting all that together, what does the optimization problem look like? Minimize over x from j of x. And x here is a vector. But we're going to, again, consider just a scalar objective function. So if we have lots of objectives, we're going to roll them all up together subject to the constraints, the hk of x.

And actually, let me write explicitly this is a function of x and p. And it's quite common to put the semi-colon here to show that j, the objective, depends on the parameters. But the parameters are kind of different to the design variables. But the things we can change, these are just things that affect [INAUDIBLE], but they do affect j.

So the same thing for the [INAUDIBLE] vector as well. For the constraints, hk is equal to 0. And that's k equal 1, 2. I think we had m2 of those. So gj-- again there, a function of x and p-- are less than or equal to 0. And j goes from 1 to m1.

And then the xi lie between the lower bound-- how did I write [INAUDIBLE] that way-- and the upper bound for the design variables where some of these guys might be minus infinity or infinity if we don't actually have bounds. OK. Yup.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. Yeah. So again, whatever the problem, however the problem shows up, we can always write it in this form. So exactly, if we actually wanted to maximize something, if we wanted to maximize range, we would just minimize negative range. So any objective that you want to maximize, you're going to multiply with a negative sign and try to minimize it.

How about the inequality constraints gj less than equal 0? You can always just rearrange it and turn it around, right? Multiply by minus 1 on this one. And flip the sign if it's a greater than or equal to sign.

So the fact that we wrote this as a minimization and with these guys being less than equal 0, let's go to the standard form that for the nonlinear optimization problem. It's sometimes called NLP for Nonlinear Program. But you can always get your problem into this form.

So I forgot to scroll through this thing. OK. So let's see, any questions? It's clear? Mathematical statement. This is a design problem written as an optimization problem, finding the xs that minimize my objective while satisfying the constraints. And that will be a good design.

All right. So what we're going to talk about now are how we might go about solving this problem. So this is number two. And I'm going to focus mostly on unconstrained optimization problems. I'll talk a little bit about how to handle the constraints maybe in the next lecture-- so unconstrained optimization.

So many or most-- so let's say many optimization algorithms, in fact, almost all optimization algorithms are iterative-- [INAUDIBLE]-- which means that we're going to iterate to try to solve this problem. So what do I mean by that? We're going to have a guess for x. We're going to update that guess in some way. And we're going to keep iterating, keep updating the guess for x, get a new guess for x, new guess for x, new guess for x until we think we've solved this problem.

And the way that we can write the kinds of optimizations algorithms we're going to talk about is that xq is equal to x q minus 1 plus alpha q times xq. And this is going to be true for q equal 1, 2, up to however many iterations we have. OK. So what are all these symbols?

So first of all, q is going to be the iteration number. So the superscript q here denotes our guess for whatever quantity or our value for whatever quantity on iteration q. And specifically, xq is going to be our guess for x on iteration q. OK. So it's our current guess for the design variables.

And what you can see is that the guess on iteration q is going to be given by the guess of the design variables on t minus 1, so what we had before, plus this update turn. And this update turn is about a scalar alpha q. And it's got a vector Sq. So let me explain it, and then we can write it down.

This thing here is called the search direction. And this thing here is going to be out scalar [INAUDIBLE]. So again, what I want you to do is I want you to picture the design space as being like a landscape. Right? So that's the xs.

And again, the height of the landscape is going to be a measure of j. So I'm standing in the landscape. And I'm standing at point x q minus 1. We can [INAUDIBLE] q equal 1. So I'm standing at a point x0. And I want to figure out a way to go.

I'm going to take a step. I'm going to move in the landscape to get my next one, to get my x1. And the way I move is that first of all, I'm to pick a direction to move in. So if q is my search direction, so first of all, I'm going to look around. I'm going to pick a direction to move in.

And maybe you have some intuition that if I'm trying to minimize something, I'm going to try to pick a downhill direction, right? So I'm looking for a bottom of a valley. I'm standing at a point in the landscape I might want to walk downhill. So I'm going to pick a direction Sq.

And then once I've picked my direction, I want to figure out how far to walk in that direction. And that's the alpha q. So the Sq is the direction. And then alpha q is going to be the size of step.

So this is scalar in that direction. So standing in the landscape, pick a direction. Take a step [INAUDIBLE] alpha q. So that gets me to the new point in the landscape. Then I'm going to stop, and I'm going to look around.

I'm going to find a new direction. And I'm going to do the same thing, figure out how far to walk. We're just going to keep walking around the landscape picking directions and figuring out how far to walk in those directions.

OK. And of course, then the trick comes. How do you this? And that's what the different optimization algorithms do, different ways of picking search directions and steps.

OK. So Sq is, again, our vector search direction. So it's got the same dimension as x. Somebody is smoking out the window. Do you guys smell it?

[INAUDIBLE] [AUDIO OUT] is [AUDIO OUT] [INAUDIBLE] q and alpha q is a strong consideration, too. And then the other thing we're also going to need is we're going to need an x0, which is going to be our initial guess, all right? So we're always going to have to initialize from some x0, so that we can start this whole process of walking through the design space.

OK. It's very [INAUDIBLE] wavy. Actually, that's a [AUDIO OUT] satisfy this problem. [AUDIO

OUT] You can [AUDIO OUT] make guarantees about whether the final x that comes out after you iterate satisfies the assumptions [AUDIO OUT]. So there are things that [AUDIO OUT] [INAUDIBLE] has to be true. So [AUDIO OUT] I just want to minimize j of x. Forget about [INAUDIBLE]. It has to be true.

Yeah, what's that mathematical condition? What has to be there? Yeah, [INAUDIBLE]. [AUDIO OUT] [INAUDIBLE] depending on [AUDIO OUT], we can guarantees about [AUDIO OUT] additional [INAUDIBLE] that you need [INAUDIBLE] derivative has to be positive for a minimum.

And we'll talk about what that means, because this is a vector [INAUDIBLE]. So I call these S's. I mean, they're iterates. They're on the way. And again, depending on the what algorithm we use, we will get to at least a solution of that problem.

I want to-- that's strange-- I want to show you this idea. It's a little MATLAB demo. Because I think it will help you think about some of the issues.

And then we'll start talking about how we actually can-- well, we'll talk about some of the mathematical quantities that we're going to need to be able to compute these alphas and S's. And then we'll talk about the different methods and how they do those. So [INAUDIBLE] still simple [INAUDIBLE] script here that does optimization.

It does optimization on the function. This is the peak function in MATLAB. So this is the landscape. There are two design variables.

Here, they're called x and y. So this is x1 and x2. And this is j on the vertical axis. So this is the objective.

And we're going to try to maximize just because it's easier to see what's going on with maximizing. So we've asked [INAUDIBLE] constraint to [INAUDIBLE] an unconstrained problem, we've asked the optimizer to find the design vector x, which in this case is xy, two components, that maximizes the objective j where the height and the color are the measure of j. And this little asterisk sitting here is the initial guess that I gave.

When I called it, I gave it an initial guess of-- what did I give it-- minus 1, 1. OK. So this is x0. And I'm going to start it running in a second. And what you're going to see is a sequence of little asterisk. And each asterisk is a new guess or a new iterate.

So this is x0. It's going to compute x1, x2, x3, through the script all the way until, hopefully, it's gets up to the top here. So we'll set it going. And you can see it's doing what you would want it to do, which is it was asked to maximize.

So it's looking around. It's picking search directions. And then it's deciding how far to step. On each iteration, it's moving in a particular direction and taking a step. And you can see that it gets to the top of the hole.

So this is an optimization method that's called Nelder-Mead simplex. And I'll show you just a little bit about it later on. That's fminsearch in MATLAB. And we'll talk in the next lecture about how to call the MATLAB optimization functions, because they're really powerful. And they can really help you if you're doing design problems. But that one's fminsearch. Do you want to ask a question? Yup.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. We'll do that in one second. Yeah. I just want to show you this-- oops-- still running. Hang on. So it's still going. So fminsearches, you'll see it-- where is [INAUDIBLE]? It's interesting that it's still running.

So fminsearch, this is why you should always put maximum iterations in your code. Because I'm pretty sure it's at the top of the hill, but the [INAUDIBLE] probably fit two [INAUDIBLE]. Yup.

AUDIENCE: [INAUDIBLE]

PROFESSOR: No. It changes each iteration. Yeah. And we'll talk about how to compute it. So I think I'm going to kill that. So fminsearch is actually-- it's called a Nelder-Mead simplex. It's kind of a pattern search. So it's sampling the design space and doing a pattern search.

And we'll talk about that. But I also just want to-- oops, not that one. How does it measure-- well, we'll talk about that as well. Yup, 0.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Nelder-Mead simplex is the method that's implemented in fminsearch. So there's another one-- oops, now I'm [INAUDIBLE] the wrong. Actually, [INAUDIBLE]. There's another one called fminunc, which stands for minimization unconstrained. And this is one that uses gradient information.

And again, we're going to talk about how that works. But let me just run the same, so you can see. [INAUDIBLE] after 49 iterations. I don't know why it was continuing to go. But let's run from the same initial condition.

OK. So it takes a little bit longer per iteration. But did you see how it basically jumped all way to top of hill in the second iteration? And you'll see it when we talk about algorithms. But that's because we actually computed the gradient here rather than just sampling locally, which is what now the Mead simplex does with triangles and flipping them on and taking a long time.

This one used a gradient. And you can see just how quickly we got to the top of the hill, 6 iteration, 24 function evaluations. And again, we'll more talk about the different aspects.

But this is definitely a message that if you can compute gradients-- which we can't always do, in this case we can do it analytically-- you can get really quickly to an optimal solution. And so fminunc and fmincon is the constrained sort of version of fminunc can be really powerful.

So let's see, Kevin wanted to know what happened if we started from a different solution. So what would you like? OK. So do you want to pick a-- so somewhere here? So like minus 1, minus 1? Let's see.

OK. So there's the new asterisk. So what do you think's going to happen? So it goes to the local peak. It's going to convert pretty quickly. So while an advantage of gradient based optimization is that it's super efficient, because you use the gradient information and you go to the [INAUDIBLE] in the design space and [INAUDIBLE] you can [INAUDIBLE].

We'll talk about [INAUDIBLE] condition. The gradient of j equals 0 and [INAUDIBLE]. We can't tell the difference between being here, being here, or being here.

And that's kind of a drawback of a local method. At best, we can say that we are in a local maximum in this case. So we have to be careful about where we initialize. And you probably guess if we start close to this one, we're going end up up there. If we can start somewhere over here, we have a good chance of getting up here. And if we start where we started, we're going to go here. It's one of the drawbacks. [INAUDIBLE], do you have a question?

AUDIENCE: Yeah. I was going to ask how could you know how far to go [INAUDIBLE]?

PROFESSOR: Yeah. We're going to talk about that, yeah. So we're going to talk about how it chooses x, which is going to use gradient information. But it's not just the gradient you'll see. And then how it knows how far to go is that once it's picked the gradient direction, it looks to see the shape of the function in that direction. And we'll talk about how it actually knows how far to go. Yeah. Is there another question?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. This one here? Yeah. It's not quite clear. If it were right on the settle point-- it's not quite clear. And I'll see if I can-- I think it's a similar question.

If I started over here, so let's see, x will be close to minus 3. And y will be close to 3. So let's start it at, like, minus 2.5 and plus 2.5. So I think that's over in a corner of the design space. Do you see any potential problems here?

It's really flat, right? So when we compute the gradient, it's going to be really flat. It's going to know that it's not at a local minimum, because the second road is not going to be positive. But it can be that the algorithm has difficulty figuring out which way to go, which might also be the same thing if you started in that shoulder between the two.

But so it turns out, I mean, it's going to have difficulty. And you can see it's probably having a few issues. It's running pretty slowly, which means it's probably doing a lot more function calls.

But these algorithms and especially MATLAB's optimization toolbox is really good. So they have ways of recovering. So that when things go wrong, if you can't figure out where to go, you do a little bit of sampling and there are many ways that you can recover. And in this case, we managed to recover and actually get up to the top.

We're not there yet. It'll get there, though. It should get there. It is-- yeah, you can see. Even though it managed to get there, I mean, I don't know if you looked at the numbers before.

This is the number of times that it calls the evaluation of the surface. And usually, it's on the order of 3, 6, but you can see it took 27. And that second iteration, that's because it was in this flat region.

But you can see that whatever MATLAB, whatever [INAUDIBLE] built in to try and sort of make the algorithm robust, actually did manage to recover in that [INAUDIBLE]. These things can definitely fail to converge. They can come back without finding a solution.

So there's all kinds of problems you can run into. But conceptually, this is what's going on. I think that a picture of a landscape is a really good one to have in your mind as we talk about the math of the algorithms and the computing of the alpha and the x. OK? Is there any good way to approximate the landscape if the function is too expensive to call?

AUDIENCE: [INAUDIBLE]. We're lucky that we know what it looks like. [INAUDIBLE]

PROFESSOR: Yeah. So we know where it is. And we're visualizing it. But the optimizer didn't know that.

AUDIENCE: Right.

PROFESSOR: So--

AUDIENCE: [INAUDIBLE] are there [INAUDIBLE].

PROFESSOR: So it depends on your problem. If your problem has got particular structure, it may be that you can come up with an approximations, and in particular what's called complex approximations where you can guarantee something about the solution. So that's going to depend very much on the structure of your problem.

That's what a lot of people in operations research spend a lot of time doing is taking difficult problems and then coming up with approximations or relaxations of the problem that they can then say something rigorous about the solution and how it relates to solution [INAUDIBLE]. But I think the reality is that in engineering design the landscape usually looks a real mess. And not only does it have multiple hills and mountains, but often there's cliffs or part of a-- because we're not even looking constraints here.

Part of the design space where there's like a region where there's just no feasible design. And basically, people apply optimization methods to try to improve designs, but don't necessarily worry as much about mathematical optimality. It's more of a tool to help improve design. So think a little bit on what you're trying to do as well. Do you have a question?

AUDIENCE: [INAUDIBLE] what happens like [INAUDIBLE]

PROFESSOR: Yeah.

AUDIENCE: [INAUDIBLE] still [INAUDIBLE].

PROFESSOR: Yeah. So that crinkled up wing was probably like a solution wing. It was horrible. I mean, this is now-- that was a design problem with many more designs, so we can't visualize it. But even just thinking about it here, it was probably a design that was somewhere down here because it was so awful.

But you know, as long as you can get reliable [INAUDIBLE] gradients using [INAUDIBLE] methods, you get yourself out of that awful place, and you get to a good one. So it might take longer. Yup. OK. So Let's see where we are. So I'll turn this off.

OK. So [INAUDIBLE] a couple of mathematical things that we may need to [INAUDIBLE] concepts that we need before we actually talk about what's going on in those optimization algorithms. So we're going to need to think about gradients. So let's just think about what the gradient is in this case.

And so what we're talking about for j of x-- and remember, this is a scalar objective, but this is n dimensional design [INAUDIBLE]. So we're talking about the objective function j of x, a scalar function of n design variables. Then the gradient is what?

First of all, what dimension does the gradient have? n by n? What do you think it is? [INAUDIBLE]? N? n by-- n by 1. It's a vector of link n, n by 1. So the gradient-- but with the gradients, I'm talking about the gradient of j with respect to x.

So you should write that as gradj. You must have seen this in 1802, no? Gradient? Yeah. You're frowning, Jacobi.

AUDIENCE: No, I was nodding.

PROFESSOR: Oh, you were nodding? You looked like you frown when I [INAUDIBLE]. So gradient of j, which is the vector of partial derivatives, right? bj, bx1, bj dx2 down to bj dxn. So the gradient is-- it's an n by 1 vector [INAUDIBLE] grad j, which is just a vector of partial derivative.

And normally, we'll be interested in the gradient evaluated at a particular point, because you know, again, we're going to be evaluating the gradient at the point we're standing in the landscape. And so we would write that maybe as gradj evaluated at the point xk. So [INAUDIBLE] going to use q. You can leave q over there, which would mean taking these partial derivatives and evaluating them at the point x equal to xq.

So you guys did a reading problem that I know was lots of issues, because it broke the MITX platform's ability to take pictures as an answer. But you computed the gradient vector. And then you substituted a particular value of the design vector x. And then the gradients just came out to be numbers, right? Yup.

So that's the gradient. So when we talk about the gradient of the objective, it's an n by 1 vector that contains the partial derivative. And so if we had a gradient [INAUDIBLE] have [INAUDIBLE] is also going to be a vector of dimension n.

OK. So we know that we need gradient of j equals 0 to be at an optimal point. Yup. Minimum, maximum, all at [INAUDIBLE] point. So gradient of j equals 0 means that all these partials have to be equal to 0.

But we also said that if we wanted to be sure that we're at a minimum, we need to look at the second derivative information. So what is the second derivative of j with respect to x? What dimension does it have? n by n. It's an n by n matrix. And it's called the Hessian.

So the Hessian matrix is a matrix of second derivative. So it's an n by n matrix. So we could write it as grad squared j. And so it's just on the diagonal, we're going to be the pure partial. Del squared j del x1 squared del squared j del x2 squared all the way down to del squared j del xn squared.

And then on the up diagonals are they mixed terms, del squared j del x 1 del x1 del x2 and so on. Del squared j del x1, del x10. And what's special about this matrix?

It's symmetric. Because a mixed partial with respect to 2, 1 is the same as the next partial with respect to 1, 2. So n by n matrix, it's symmetric. And so in the scalar case, if you wanted to minimize-- how do I want to put it?

Let me put it here. So in the scalar case if I asked you to find a minimum of f of x where x was a scalar and f was a scalar, you would have two conditions, right? You would say, yes, the x equals 0.

And then second derivative, [INAUDIBLE] positive. And that would be evaluated at the x, the optimum point. So we could have made it evaluate it at x star.

So you've seen all this before, right? This is 1801 or high school calculus. So in our case, we are trying to minimize j which isn't a scalar, but now it's a function of the design vector x.

So the corresponding condition here is this grad j evaluated at x star equals 0. We can put a [INAUDIBLE] symbol on there if you want to remind yourself that this is an n by 1 vector, and

[INAUDIBLE] grad j equals 0 means putting all those entries to 0.

What's the corresponding case here? What's the analogous condition to second derivative being positive? Who's taken 1806? Here it's a scalar. And it's got to be positive. Here, it's a matrix.

What have you ever heard about matrices and being positive. Positive definite, yeah. So the condition is that the [INAUDIBLE] matrix grad squared j, again, evaluated at the thing. And [INAUDIBLE] write it that way.

In symmetric matrix, all eigenvalues have to be positive. They're going to be real and positive. So in another way, for any vector, say y, y transposed times this matrix times y has to be strictly positive, which you can show that it's equivalent to the eigenvalue [INAUDIBLE].

So turns out the eigenvalues of this Hessian matrix tell you an awful lot about the shape of your design space. And maybe intuitively, I find it conceptually easy to think about maximizing. If I'm standing at the top of the hill and I'm trying to maximize and think about the eigenvalues and eigenvectors, there's no direction I can move where things are going to increase, right? So that means all the eigenvector directions are going to have to be associated with decrease.

And so this would be flipping, right? To be the maximum, the Hessian would have to be negative definite. And so it kind of makes sense that the eigenvalues of the Hessian are going to relate to what goes on as we move away from a minimum or a maximum point.

OK. So we've got gradient. We've Hessian. We need one more thing, which is your old friend Taylor series expansion. Yeah, Greg.

AUDIENCE: Can you go over that definition [INAUDIBLE]?

PROFESSOR: OK. This right here? Yeah. So let me write it out here to be-- So let's call it h. So let's write the Hessian-- I'm just going to call it h, so I don't have to keep writing grad squared j.

So the condition is that h is positive definite. And what that means-- we write it this way, h is a matrix. What that means is that v transpose hv has to be positive for av, for any v that's not 0.

So we take any vector v and do the b transpose hv. And that has to be positive. If that's true, then h is positive definite. And then, because h is symmetric matrix, that is the same condition as saying all the eigenvalues of h-- and there are going to the n of them-- have to also be positive. It's a property of the matrix.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah.

AUDIENCE: [INAUDIBLE]

PROFESSOR: So if not all the eigenvalues are positive, if some of them are 0, then it means--

AUDIENCE: [INAUDIBLE]

PROFESSOR: Well, so if some of the eigenvalues are positive and some are negative, it means you're at a settle point. You've seen that in the 1dk--

AUDIENCE: [INAUDIBLE]

PROFESSOR: So I don't know how to act this out. But in a landscape if I a settle point is [INAUDIBLE] point means that the [INAUDIBLE] goes this way in one direction, but this way in the other. So there's still a direction that I can move to [INAUDIBLE].

AUDIENCE: [INAUDIBLE]

PROFESSOR: So maybe [INAUDIBLE] said another way. An optimization algorithm that's trying to minimize or maximize something won't stop at those points. Because it will be up to find a direction of improvement.

AUDIENCE: [INAUDIBLE]

PROFESSOR: No, it'll keep going.

AUDIENCE: [INAUDIBLE]

PROFESSOR: That's right. Think about standing on-- you're standing on the edge and things are dropping either side of you. But if you're looking front, you can keep going up the hill.

So it's just going to constrain the directions in which you move. But the problem with [AUDIO OUT] as we got into the [AUDIO OUT] regions where [AUDIO OUT] you might know [AUDIO OUT] if you're minimal and not unique, then there's actually a ridge. The top of the mountain would be the ridge.

And so there's a direction that you can walk along, and you're not changing an objective. You're staying at constant elevation. And that would correspond to one of the eigenvalues being 0.

But in that case, what you've got is a whole bunch of designs that are equally good. And so that's actually kind of nice to know. Because that would be different design decisions that you could make that were all as good in cost or whatever it is you're trying to do.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. So, yeah.

AUDIENCE: [INAUDIBLE]

PROFESSOR: That's right.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. So you don't have to worry about this stuff. This stuff is taken care of for you when you run something like fminunc and fmincon. But what's important is to understand that there are sort of rigorous mathematical conditions that tell you when you're at optimal solution. And if your problem sort of obeys the given structure, then that's great. And that holds.

In reality, remember when we had our list of design variables we talked about having variables like number of engines? That's something for which the gradient doesn't even exist, right? It's not differentiable.

And so in many cases, our problems don't even satisfy the requirements for the algorithms that we use. And so there are very few guarantees. But they can still help us make progress.

AUDIENCE: [INAUDIBLE]

PROFESSOR: That's the optimal solution?

AUDIENCE: [INAUDIBLE]

PROFESSOR: If there's one of these ridges, if there's a non-unique solution, yeah. Yeah.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. So integers becomes a whole other ball game. And maybe I should have said right off the bat that when you start looking at the gradient and the Hessian analysis, [INAUDIBLE] j of x needs to be at least twice differentiable with respect to x. Or otherwise, what we're writing here doesn't make sense.

When you have integers, the conditions become more complicated. And how you solve the problem becomes more complicated. One approach is to let the integers vary, exactly what you're saying. And then at the end, you round. And that might be effective, but it's certainly not guaranteed to find an optimal solution.

There are specialized optimization solution methods that are tailored for integer programs. And in fact, MATLAB just released a mixed integer program, I think, as part of their latest release of the optimization toolbox. And these methods will handle the integer variables in different ways.

It's something called [INAUDIBLE] bound, which is about searching down different integer combinations. But yeah, integer variables make it really difficult. Yeah, Alex.

AUDIENCE: [INAUDIBLE] j [INAUDIBLE].

PROFESSOR: Yeah, so we're going to talk about-- that's sort of number three. j is usually going to be computer code that we can put in the shape of the aircraft wing or the whole aircraft, and out comes range or cost or whatever it is, that's right. So we can usually only get j of x through simulation not necessarily by analytic.

And so then we're going to need some way to compute gradients. And what's really nice is finite differences, which you guys saw back a few months ago, is what we're going to use to do that. Yeah.

AUDIENCE: Like also, one thing we talked about [INAUDIBLE].

PROFESSOR: So it's all taken care of, because we're not sampling. We're actually measuring the gradient. And I mean, they're going to be-- I mean, we're doing something different here, right? We're moving through the landscape.

We're not sampling, and then trying to say how this variable relates to this one. We're actually just looking for an optimum solution. It's a different thing.

So whatever interactions are there, how are they affected? They're reflected in the shape of the landscape. The shape that the landscape takes is a manifestation of how the design variables relate to each other and how they affect the objective.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. There are. And I'll show you some of those. The problem is actually now to guarantee a global maximum. So there are a whole bunch of methods called heuristic methods. So genetic algorithms-- which Professor [? Devic ?] uses a lot in his research, which I happen to not particularly care for-- are ways to do that. And I'll show you maybe in the next lecture.

They sort of spray points everywhere. And then they use an analogy with natural selection and mutations. And designs have babies. And then the strong babies survive and the other ones don't. So there will be [INAUDIBLE].

And you know, people sort of claim that's a way to do global optimization. The problem is there are no guarantees at all. I'll show you another one, which is a patent search, which sort of has a guarantee of eventually finding a global solution, but only [INAUDIBLE] that you search forever.

Yeah. And it's a really hard problem. If you know something about the structure of your problem, you maybe have to do something. And certain problems, like we were talking about before, have the nice structure where you can be rigorous.

If you have truly just kind of this black box complicated aircraft design problem, there are no guarantees. But again, often what you're trying to do is to find a good design that meets all the constraints that's better than what you could do by hand. So [INAUDIBLE].

Yeah. It depends. Are you an engineer trying to make a good design decision? Or are you a mathematician who wants to guarantee optimality? And where do you fall in that? So that's god.

So you guys have lots of questions. You're trying to avoid getting Taylor series expansions I know. Any other questions?

So Greg, does this sort of answer-- I know I didn't do it very deeply. But it's probably enough.

AUDIENCE: [INAUDIBLE]

PROFESSOR: So just the last mathematical ingredient that we need or that we will need are going to be

Taylor series expansions. And it's just, again, same thing. You've seen them in the scalar case. But let's just make sure it's clear in the gradient case.

And by the way, If you really do have lots of questions and you need to take the graduate class set [INAUDIBLE] and I teach-- which we're just teaching it this semester, and it's offered every other year. But it's on design optimization, a whole class on this stuff, which is fun. Taylor series expansion.

Again, in the scalar case, let's just do what we did over there and do the analogy. So in the scalar case, if I had some f of-- let me use z. I probably should have used z over there. It's OK.

If I have some f of z, what is that? It's f. If I'm expanding about the point b0 plus the first derivative evaluated at the point b0 times z minus z0 plus the second derivative evaluated at the point z0 times z minus z0 squared plus blah, blah, blah. Right? So that's the Taylor series expansion that you've seen.

So how does it look in the vector case? So when we talk about Taylor series expansion, we're talking about expanding j of x, where again x is now this n dimensional vector. And we're going to expand it around the point x0 point in the landscape.

OK. So what does the first term look like? Gradient of j evaluated at x0 multiplied by x minus x0. OK. So we have to think about the dimensions. It always helps me think about the dimensions.

So [INAUDIBLE] n by 1, right? What do we need to do to this thing? Transpose. So it's in a product between the gradient evaluated at x0 and in the delta x, x minus x0.

Yup. Scalar, scalar, 1 by n times n by 1. That's a scalar. OK. How about the second derivative term? What do you think that might look like?

We'll put the Hessian in the middle. And it's the Hessian, again, evaluated at-- let me write it thigs way-- x0.

AUDIENCE: x [INAUDIBLE]

PROFESSOR: OK, good. Yeah. So x minus x0 on that side. And then x minus x0 transpose. And again, this is an n by n and n by 1 [INAUDIBLE] a scalar. And then if you went higher, you would be getting a tensor for the third derivative.

AUDIENCE: [INAUDIBLE]

PROFESSOR: You can. And in fact tensors are very popular, lots of people working on them now. I don't-- don't ask me anything about them. OK. So let me come back to this. Then I can start showing you some of the optimization measures. I'll show you pictures of some of them first. And then we'll look-- yeah.

AUDIENCE: [INAUDIBLE]

PROFESSOR: No. Yeah. So I could've written-- yeah. So that's just the point about which we were doing, the Taylor series expansion. And what you'll probably guess is that it's actually going to be xq. We're going to lock locally.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah, well, whatever. q minus 1, yeah. OK. So we can close this. So actually Professor Johnson, who's in the math department who teaches a really great graduate class on numerical linear algebra, it actually covers lots of things. He has this NLopt package where he's implemented lots of optimization algorithms.

And he has them implemented. And you can access them in MATLAB or Python or a variety of ways. And also on the website, he was a really nice kind of description of the algorithms and what kind of problems they work for and issues with conversions and stuff. It's really nice.

But [INAUDIBLE], you were asking about some of the different kinds of methods. So this is sort of the four categories. There are global optimization methods that sort of strive for this, being able to find the global optimum.

There are local methods. And I'll show you how these work. And in the local methods, there can be ones that are called derivative-free and gradient-based. So these one use gradients, and these ones don't use gradients. And then there are a heuristic methods, things like the genetic algorithms that kind of use a bunch of roles and some randomness to search the design space. Yup.

AUDIENCE: [INAUDIBLE]

PROFESSOR: So non-linear refers to the fact that j of x could be a general non-linear function and that the g of x and h of x-- yup. To be a linear program, j would have to be a linear function of the x's. So just w1 times x1 plus w2. And then the constraints would also have to be linear functions of [INAUDIBLE]. And if you have a linear problem, then there's the [INAUDIBLE] simplex, different from the Nelder-Mead simplex, but the simplex method that's really efficient, can solve really big problems, lots of theoretical guarantees.

All right. So he's the Nelder-Mead simplex. So this was fminsearch. Remember, that was the very first one that our little landscape was working on. So this is a local method that's derivative free. So it's not using any gradient.

And how does it work? So a simplex is a special polytope of N plus 1 vertices in N dimensions. For me it's easiest to think about in 2D [INAUDIBLE] a triangle. So simplex is a triangle in 2D. And here's how this Nelder-Mead simplex method works.

So you take your initial guess that, again, we have to supply. This is the initial point in the landscape. And you form an initial simplex. So with our 2D landscape, we're going to put down a triangle, initial triangle.

And we're going to evaluate. And remember, I'm talking about unconstrained optimization here. So there are no constraints. So you evaluate j, the objective, at each one of the points in the triangle, three. So in two dimensions, we have three points.

So that's what can keep the function value. The function value here is j. [INAUDIBLE] the triangle-- the triangle can be--

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. I mean, yes. In terms of size? You've got your initial point. And you're going to put two other down here. There would be-- they're close by though. It's local. So they're close by. Maybe not super close, but they're close by.

And you'll see that's going to change. So it doesn't actually matter too much what we start with. We're going to order the vertices according to function values and discard the worse one. So if we're trying to minimize, we have the highest value, the lowest value, and the middle one.

If we're trying to minimize, we would throw out the one with the highest value, right? So we're going to throw away, in this case, x high. Because it's got the highest value, and we're trying to minimize.

And we're going to generate a new point by what's called reflection, which means that I'm going to come across this line of the remaining two and reflect the triangle over and generate a new point that's kind of on the opposite side of the simplex. And at the same time, I'm also going to-- I'm going to [INAUDIBLE] a new point over here. This is the xr.

I'm going to run the function there and see whether things got better or not. So we're trying to minimize. We're going downhill. Run the function here and see whether this is actually a better function value.

And then I'm also going to decide, depending on how steeply down the hill I'm going, whether I want to change the size of my simplex. And if things are going really well and I'm generating points and I'm really going downhill quickly, I'm going to make the simplex bigger and bigger, so that I can go down hill faster and faster. But if this guy's not really so good, then I might shrink the simplex so that I look more locally.

So maybe you can kind of visualize that what this optimization algorithm looks like is a bunch of triangles that keep flipping over just by measuring, by ordering the performance of the vertices, throwing out the old one, generating a new one on the other side, walking kind of through the design space. And then the triangles are growing or shrinking depending on how well things are going. So there's no gradient. All it is is just a sampling and an ordering.

There are some convergence issues in this. But actually, it's kind of a simple algorithm. And it's pretty robust. The function doesn't have to be differentiable, right? j of x doesn't have to be smooth as long as a better point has a lower value of j of x than another one. That's OK, right?

So when I ran fminsearch, way back up if I pull it up. So you can see here in the MATLAB output it was [INAUDIBLE] things. So that was telling me what was going on. Every time it was evaluating new points, so two new points each time, that was the simplex expanding, reflecting, contracting, knew what was going on with points.

OK. So that's a derivative-free method. It uses a little bit of sampling, but it is local. And that's fminsearch in MATLAB.

This one is sort of the same-- I don't know it sort of has the same feeling. But it's a global optimization method. It's called DIRECT. And DIRECT stands for Dividing Rectangles. And basically what it does is it divides the domain into these rectangles in 2D. You'll have rectangles in multiple D.

And it basically figures out where to look. So if this one is interesting, then it would divide that more. And then this guy is interesting, so then divide that one more. And that one's interesting and keep dividing and dividing and dividing more.

And it's got all different roles for figuring out which ones to divide and which ones not to divide and how to progress. So this is one that tries to go at global optimization. Problem with this one is that it just really keeps running and running and running and running and running. So we've used this a couple times. But it tends to just take so long to run that it's not particularly useful.

OK. So let's see what time we have left. We have about 10 minutes. So let's at least start talking about the gradient-based method. So again, this is what we've been talking about starting with an initial guess, x0.

Then we've going to compute two things, the search direction, the Sq, and then the alpha-- this should have a q on it-- how far we step in that direction. And with a gradient-based method, we're going to use gradient of j to compute this Sq. OK.

And we're going to check for convergence. We'll talk about what that might mean and just keep going around this slope until we're done. So these are the methods that we'll talk about. I think maybe we'll just talk about steepest descent right now. And then we can talk about the other ones on Monday.

But steepest descent is conjugate gradient of first-order method. Newton method, which I think you've seen maybe in the scalar case. Have you seen the Newton method for root finding in the scalar case?

You've seen Newton [INAUDIBLE] in [INAUDIBLE], a little bit different. We'll see how Newton method looks in the [INAUDIBLE] case. And then there are things called quasi-Newton methods that kind of sit in the middle between the two.

So let's look at steepest descent. It's the simplest thing you could possibly do, possibly think about doing. And it just says fix this search direction on iteration q to be negative the gradient of j evaluated at the current design iterate, Sq minus 1. So why would you pick that search direction?

It's the steepest descent, yeah. So we know that the gradient of j points in the direction of maximum local increase of j. Negative gradient of j points in the direction of steepest descent. So here's the algorithm. But again, just think about the landscape. What are you doing?

You're standing in the landscape. You're looking around. You're finding the direction of steepest descent. And you're going to go in that direction.

And we'll have to talk on Monday about how to choose the alpha. But we find the direction of steepest descent. We choose the alpha.

We take a step. We look around. We find the direction of steepest descent. We can pick a new gradient, find the new alpha, take a step, and keep repeating.

AUDIENCE: [INAUDIBLE]

PROFESSOR: That's right. That's exactly right. And I'll show you how we do this on Monday. That's exactly right. It becomes [INAUDIBLE] the 1D search. Because once you've define the direction, it's now just 1D.

So that's the first gradient-based optimization algorithm. It's really simple. It turns out that's not a great algorithm, that it converges really slowly. But conjugate gradient is actually something that's not too different.

So now what does conjugate gradient do? The first search direction S on the first iteration is the steepest descent direction. Yeah. But then on subsequent iterations, the search direction is the steepest descent direction minus squared j plus this term that's beta q times H q minus 1, so the steepest descent direction modified by the last search direction.

And the beta is the ratio of the gradients on the last two-- the ratio of the norm are the gradients from the last two search directions. So what is going on here? Again, I think you kind of look at the math.

But basically, what is happening is you think about the landscape. So I'm standing at x0. I find the direction the steepest descent. I move in that direction.

Now when I get to the second iteration, I find the direction of steepest descent. But I also look over my shoulder, and I see where did I come from. I find the direction of the steepest descent. And then depending on where I came from, I'm going to modify that direction a little bit, and then move.

And then I'm going to go, and I'm going to find the direction of steepest descent. I'm going to look where I just came from and modify it a little bit. So I'm incorporating information about where I came from.

And if you look at this example, this function is a really famous example that is an analytic example that's often used as optimization algorithms. It's called the Rosenbrock function. It's sometimes called the banana function, because it looks like a banana with really steep walls.

And what you can see here on the left is the steepest descent algorithm starting from the initial point here. So we start here. We compute the direction of steepest descent. These are contours of objectives. So the direction of steepest descent is perpendicular to the contours, right?

Yup. Perpendicular to the contours, what you were saying, Dominic. It looks along here. And there's the point that would minimize.

It gets to the new point, completes the direction of steepest descent, moves along here. It gets here, completes the direction of steepest descent, steepest descent, steepest descent, steepest descent, can you see what's happening to this guy? It's going to take hundreds probably of little tiny steps going zz, zz, zz, zz, as it gets to the optimum solution.

What does conjugate gradient do? So the first one is the same, the first iteration, steepest descent. But now, when we get to this point, we compute the direction of steepest descent, which again would be perpendicular to the contour. So it would be like that.

But now, we also look back over our shoulder, see where we came from, compute that data, add in that beta S1 term and modify it a little bit. So it actually takes us there. Not terribly different, right?

But enough different. Now, we get here. Steepest descent, again, would be perpendicular. It would be here. We modify it a little bit. We skip all the way here and continue on.

So conjugate gradient converted in 1, 2, 3, 4, 5 iterations. Whereas, steepest descent [AUDIO OUT] gradient [AUDIO OUT] much, much faster [INAUDIBLE] to converge. And [AUDIO OUT] and you want to get down, don't use steepest descent. Because you're going to end up going back and forth across yourself. Always look to see where you came from [INAUDIBLE].

OK. So let's finish there. We'll talk about Newton's method and various other things. We'll talk about computing gradients on a Monday as well. But if you have questions, stick around. I'm going to do office hours now if you questions about the lecture or questions about the project.