STOCHASTIC APPROXIMATION: THEORY AND APPLICATIONS

Dr. Gugan Thope

Department of Computer Science and Engineering

Indian Institute of Science, Bangalore

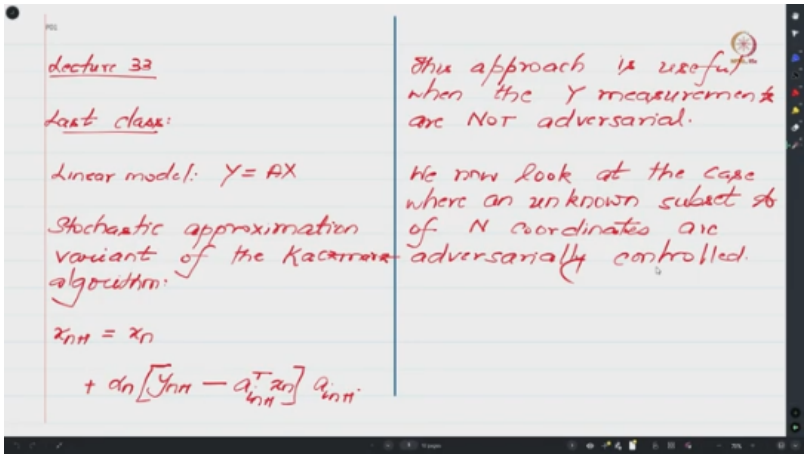
Week 9

Lecture 33

Robust Distributed Learning under Adversaries

Hello and Namaste everyone. Welcome to Lecture 33 of this NPTEL course on Stochastic Approximation. So, recall that during this week and the next week, we will discuss two variants of the standard stochastic approximation algorithm that we have discussed so far. One is the two-time-scale stochastic approximation, and the other is stochastic recursive inclusions. Right, and instead of directly describing what they are and so on, my goal in the previous class, as well as this class, is to set the stage for why we may even want to consider such variants, right.

So, if you recall, in the previous class, we looked at what we called the stochastic approximation variant of the Kakschmark's algorithm for solving an undetermined system of equations, right? And at the end of the previous class, I asked, what if some of the measurements of this random variable Y were adversarial in nature? Then what would one do? In this class, we will try to answer that question and reach a position where the utility of two-time-scale stochastic approximation algorithms becomes natural. With that in place, let us begin our formal discussion. So, let us do a quick recap. In the previous class, we considered this linear model Y equals AX, where X and Y are random vectors, and A is an a priori known matrix. We asked, given access to samples of Y, how can we estimate the expected value of X? Towards that, we proposed what we referred to as the stochastic approximation variant of the Kakschmark's algorithm.



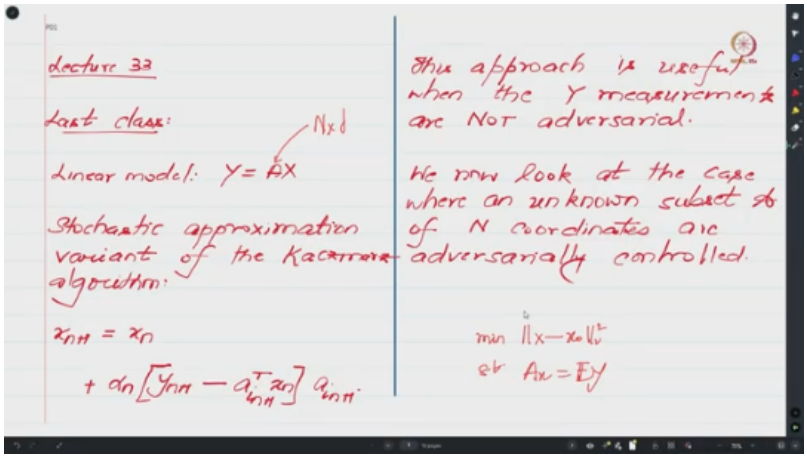
Its update rule was as follows: at time step n, given xn, xn plus 1 was computed using the Forling formula. So, alpha n is your step size, calligraphic yn plus 1 is an independent sample of the i n plus 1th coordinate of this vector y. Right, and a i transpose i is the ith row of A, and a i n plus 1 transpose similarly is the i n plus 1th row of A, right? And this is the transpose of that vector, right? And this i n plus 1, we presume, is sampled uniformly from the set 1 to capital N, where N denotes the number of columns of A. In other words, we had presumed that this matrix A had dimension N cross D.

$$Y = AX$$

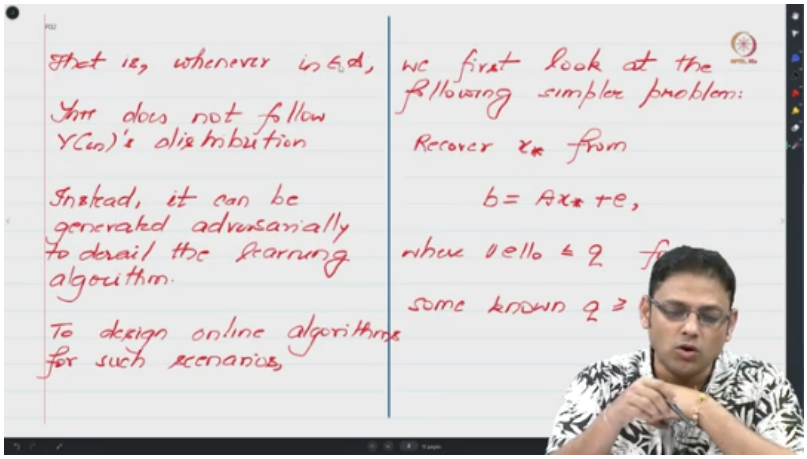
$$x_{n+1} = x_n + \alpha_n \left[y_{n+1} - a_{i_{n+1}}^T x_n \right] a_{i_{n+1}}$$

And we said that such an update rule, one can show, first of all, converges to X star, which is the solution of this problem: minimize X minus X0 the whole square such that AX equals the expected value of Y.

So, the niceness about this algorithm is that you know this is a noisy algorithm, but nevertheless it will converge to the solution of this problem which is deterministic in nature. So, that was nice about this algorithm. However, this approach is useful when these measurements of Y that is these calligraphic Yn plus 1 are generated by honest sensors or appropriately working sensors, but often in practice You know, some of these measurements come from faulty sensors or you know, they are generated by adversaries, right? So, in that case, one can ask, can we continue using such an algorithm, right?



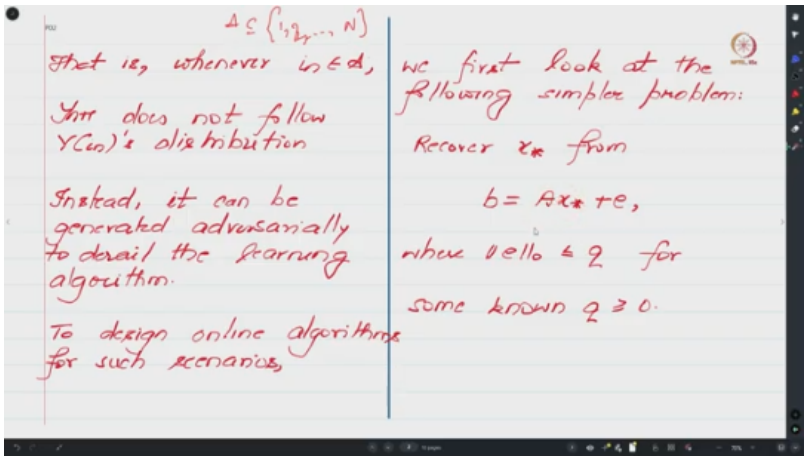
In particular, we will, you know, in this class and also the next couple of classes, we will focus our attention when an unknown subset of n coordinates of y, right, are adversarially controlled. Okay, that is whenever I n belongs to this set of adversarial indices, right? So, A here is presumed to be a subset of 1, 2, dot, dot, dot, capital N, right? So, we will presume that, you know, this is the index set of adversaries. So, whenever I n belongs to calligraphic A, right?



This calligraphic Y n plus 1, instead of following, you know, the i nth coordinate of y is distribution okay we can presume that y n plus 1 is generated adversarially right and it may be generated to derail your algorithm right the purpose of this algorithm is to estimate expected value of x and you know these measurements could be generated by adversary to derail your learning algorithm. So, one can ask how do we make our algorithm robust to such adversarial measurements. So, to design online stochastic

approximation algorithms for such scenario, what we will do now is to look at a simpler problem, take insights from it and then use those insights to design this desired stochastic approximation algorithm. So, what is this simpler problem?

Suppose the problem is as follows: there is some vector X star, and we would like to recover it. However, we do not have access to X star. Instead, we have some linear measurement of X star, which is A X star. So, if this E was not there, the question is basically: given B, can we recover X star using B? So, this is like solving a linear system of equations, and there are several algorithms for solving such a linear system of equations. Now, suppose instead of being given A X star, let us say we are given A X star plus E.



And we will presume that this E has the property that its L0 norm is at most Q for some known Q. So, the L0 norm is basically the number of non-zero coordinates. So, the L0 norm being at most Q implies that at most Q, Q many coordinates of E are non-zero.

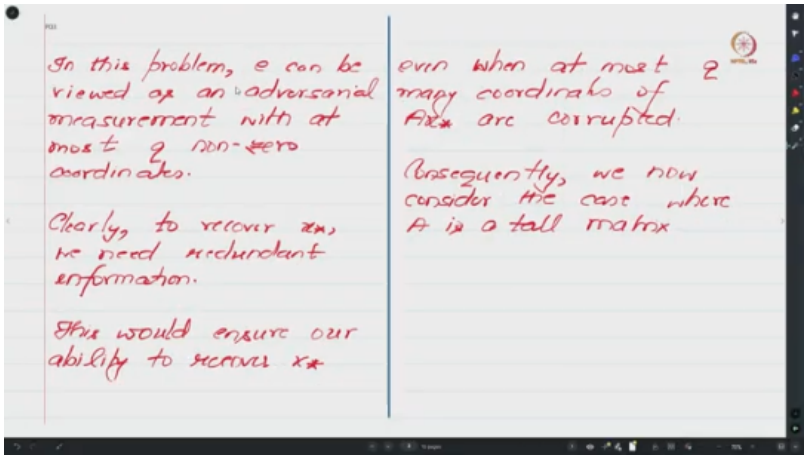
$$b = A_{x_*} + e$$

$$||e||_0 \leq q$$

$$q \geq 0$$

So, in that sense, this problem can be interpreted as follows. Instead of being given A X star, what you do is you take A X star, and the adversary corrupts

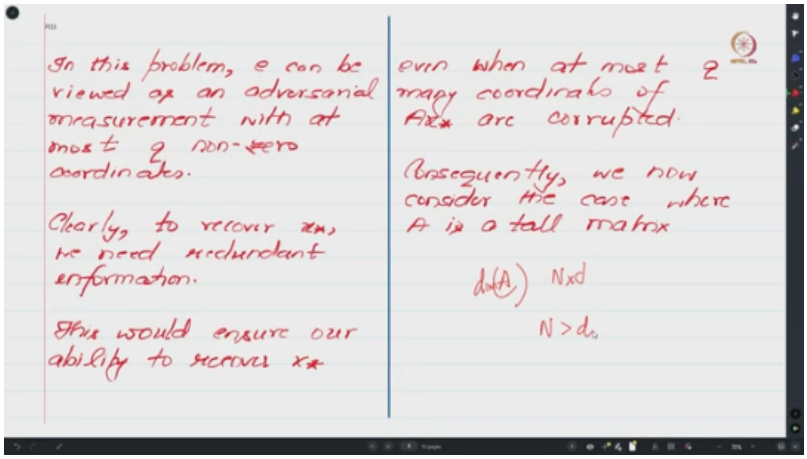
at most Q many coordinates of A X star to construct this B, and this B is revealed to us, right? And now the goal is: can I recover back X star from this adversarially corrupted measurement, right? So, you can imagine this vector E to be an adversarial measurement with at most non-zero coordinates, right? And given this vector B, right? The question is: can we recover X star?



Of course, if you know which coordinates of A x star have been corrupted, then you know, in some sense, it is an easier problem because whatever the corrupted coordinates are, you throw those coordinates away, right? And from the remaining, you try to recover x star. So, that is a relatively simpler problem. Here, we are considering the situation where we do not know which coordinates of E are corrupted. All we know is that at most Q many coordinates are corrupted. So, given just this information, the question that we are interested in answering is whether we can recover X star.

So, it is natural to see that in order to recover X star, we have to ensure that the information in A X star is redundant. Redundant means that the same information, in some sense, appears multiple times so that even if you know a few coordinates of A X star are corrupted, because of this redundancy in information, from the remaining measurements, we can recover back X star. So, the key idea that we often exploit to handle the adversarial scenario is this concept of redundancy. So, recall that when we were discussing solving this expected value of X from this measurement Y equals X in the previous class, we had presumed that A is a wide matrix, meaning the number of columns is larger than the number of rows.

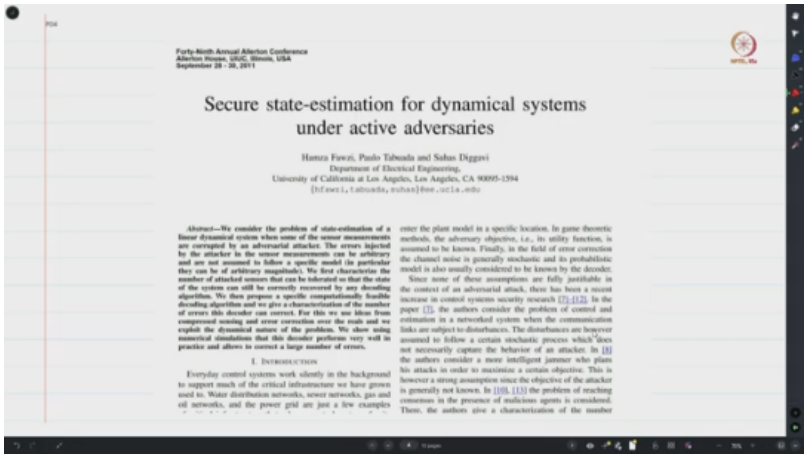
However, in this scenario—that is, the scenario where some of the coordinates can be adversarially corrupted—we want redundancy, and in order to ensure this redundancy, what we will do is we will henceforth presume that A is a tall matrix, meaning the number of rows is larger than the number of columns. Which means that, you know, if I mean, so in our case, this capital N is the dimension of the number of rows. So, your A is of dimension N cross D, right? And what we are presuming is that N is larger than D. So, this means that for a D-dimensional vector, we are providing additional information about, you know, x star in the form of A x star. And from this redundant information that is present in A x star, the hope is that even if a small subset of these coordinates of A x star are corrupted, right, we may still be able to recover x star back, right.

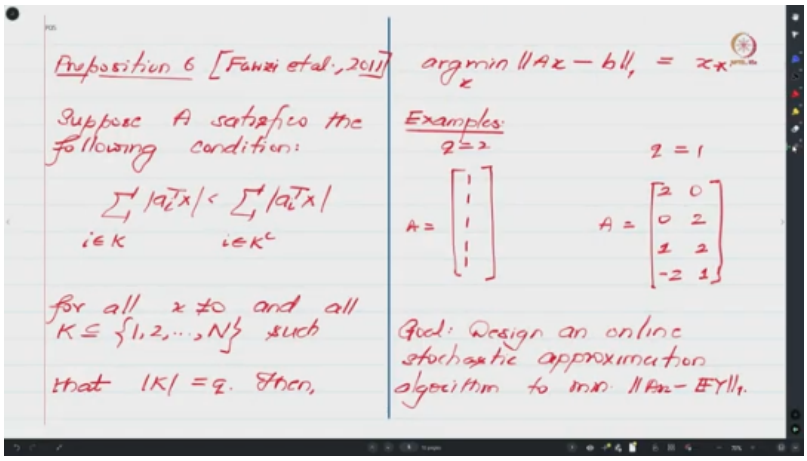


So towards this there is a very fascinating paper which I have put up over here that came in the year 2011 which discusses a sufficient condition on A that enables recovery of X star. More specifically, if you look at proposition 6 of this paper, the one that I showed you before, it says that suppose your matrix A satisfies some condition like this, then the If you solve the L1 minimization problem AX minus B and if you find the X star or the solution to this problem, then the solution to this problem will indeed be X star.

$$\sum_{i \in k} |a_i^T X| < \sum_{i \in k^L} |a_i^T X|$$

So, let us digest this one more time. This result provides a sufficient condition so that from an adversarially corrupted measurement of X star which is AX star plus V if you solve this L1 minimization problem notice that there is 1 here instead of 2.





So if you solve this L1 minimization problem then the solution to this L1 minimization problem will indeed be X star. So, even if there are adversaries and the number of adversaries is at most Q and you know this sufficient condition is satisfied, then we can recover X star. So, let us try to digest what this sufficient condition is. The sufficient condition requires that this quantity be less than this quantity.

So, what is there on the left hand side? You require that the absolute value of AI transpose X, right, where recall AI transpose is the ith row of A. So, absolute value of AI transpose X for every I in this set K, this should be less than A similar sum where I goes over K complement. Now, what is K?

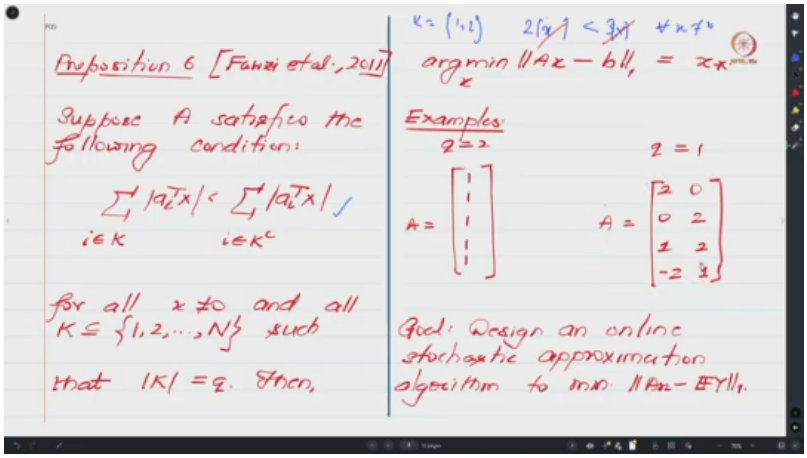
K is any subset of the index set 1 to N such that the cardinality of K is Q. So, in other words, you consider every possible subset of this index set of size Q, right? Take this sum, and we require that this sum be strictly less than this sum for all x not equal to 0,

right? And if such a condition holds, then this result concludes that by solving the L1 minimization problem, we can actually recover x star, right? So here are a few examples of a matrix A, you know, which satisfies a condition like this. For example, if you want to tackle at most two adversaries, right, and your x star is of dimension 1, which means that x star is a real number, right? Then you can work with a matrix of this form, right? So here, one can check that such a condition indeed holds. For example, if K is, you know, {1, 2}, right? So you can see that its cardinality is actually 2, so the left-hand side in this case will be

x star because A1 transpose here is just 1, as indicated over here. So, x star, and the cardinality of K is 2. So, you will need to add x star 2 times, right? And we require that this be less than, right? The right-hand side, which again, since all the AIs are the same, right?

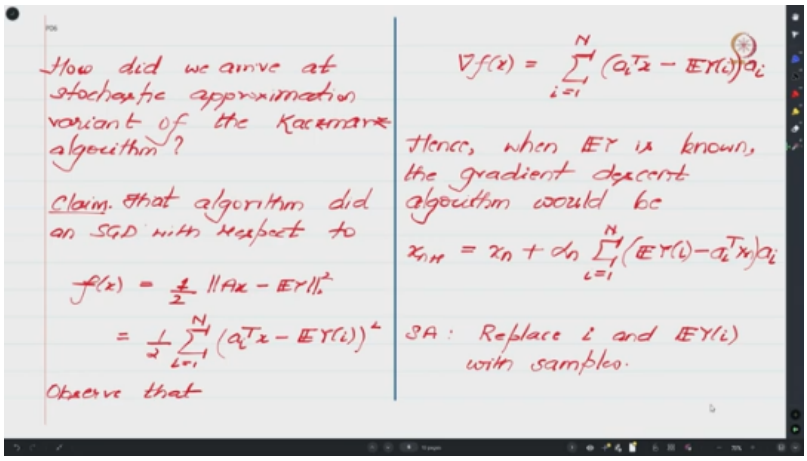
The left-hand side will be 3. Sorry, I should not put x star here; rather, it should be for x, right? So, we require that 2 times the absolute value of x be less than 3 times the absolute value of x for all x not equal to 0. And since the absolute value of x is a positive quantity—strictly positive—we can cancel this off, and indeed one can conclude that since 2 is less than 3, this inequality indeed holds when A is chosen in this fashion. One can similarly check for other choices of K. And in the same way, for Q equals 1, and if your x star is of dimension 2, such a matrix can handle at most one adversary.

And again, one can check that this matrix A satisfies such a condition for all X not equal to 0 and index at K of 1 to N, where the cardinality of K is 1. So, this measurement matrix can, you know, handle one adversary; this measurement matrix can handle two adversaries, and so on and so forth, right. So, now, you know, this proposition tells us, right, when you have a deterministic system of equations where the measurement is, you know, adversely corrupted in at most Q many coordinates, how to recover back X star, right. So, now the question or the goal that we would like to pursue is: can we use this idea—that is, the idea presented in this Proposition 6—to solve this problem? So, notice that I have replaced B with the expected value of Y, and I would like to presume that maybe some coordinates of the expected value of Y are corrupted.



Can you somehow you know, solve this minimization problem? So, of course, if the expected value of Y was known, then you know you could just use a standard, you know, L1 solver, which is actually a linear program. So you could use that and find this X star. However, the interesting case would be when we do not have the expected value of Y, or we do not know this; instead, we have access to samples of this random vector Y.

So that is our goal, and now, as I told you, I mean, I want to sort of set the stage that motivates the use of two-time-scale stochastic approximation. So, towards that, let us recall how we arrived at the stochastic approximation variant of the Kaksch-Marx algorithm. This was the algorithm that we arrived at or, you know, considered or studied in the previous class. So, let us first imagine or try to come up with a recipe that helps us identify that the Cox-Marx algorithm is one of the potential algorithms. So, there, in the problem setup that we discussed in the previous class, we had no adversaries.



Right, and hence, you know, one can ask: in order to solve this AX equals expected value of Y, okay, can we construct an objective function in this fashion where notice that the norm is with respect to That means this is the L2 or the standard Euclidean norm, and we ask: can we somehow minimize this function where we do not have access to the expected value of Y, instead we have access to samples of this random vector Y? So one can see that if I write this objective function in this way and have used this half over here, just to simplify our lives. So let us say f of x is this function, and this one can immediately see equals this expression that we have given over here. This is the sum from i equals 1 to capital N, and this expression is ai transpose x minus the ith coordinate of the expected value of y.

Right? And one can easily see that if I take the derivative of this expression, this will give me an expression like this. Right? And you can see that this half helped in cancelling off the 2 that will arise when I take the derivative of this expression. Right?

$$f(x) = \frac{1}{2}||Ax - Ey||_2^2$$

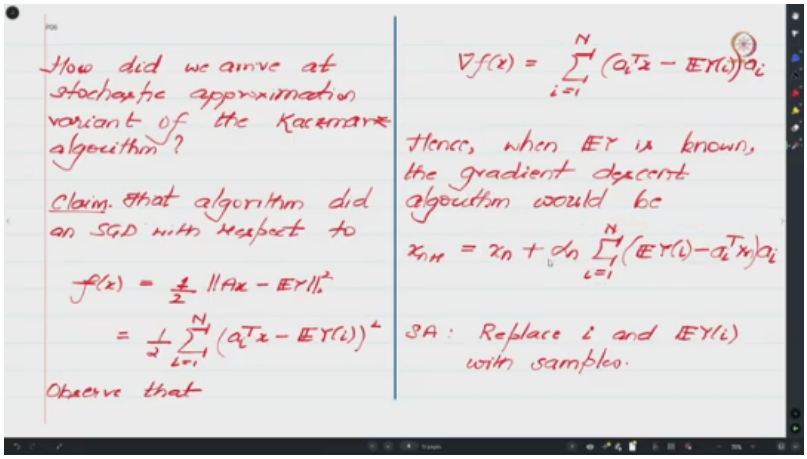
$$= \frac{1}{2}\sum_{L=1}^{N}\left(a_i^T x - Ey(i)\right)^2$$

$$\nabla f(x) = \sum_{i=1}^{N}\left(a_i^T x - Ey(i)\right)a_i$$

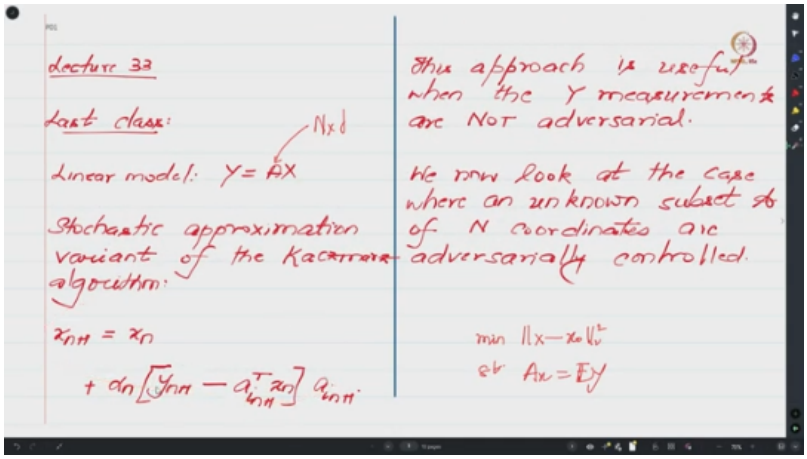
So, this is the gradient of this objective function. Hence, when the expected value of y is known—that is, this vector is known—right, and you sort of write down the gradient descent, not stochastic gradient descent yet, because we are presuming that we know this vector, the expected value of y. So if you know the expected value of y and try to write the gradient descent algorithm, its update rule will be what is given over here. That is, we would have xn plus 1 equals xn plus some step size, and I want to write the gradient of this objective function with a negative sign. Right?

So the gradient is over here. I put negative sign over here and one can see that I would end up with an expression like this. That is Xn plus the negative of the gradient times some step size is used to construct Xn plus 1. So this will be our gradient descent

algorithm. Now, the problem that we were interested in solving was the case where expected value of Y was unknown.



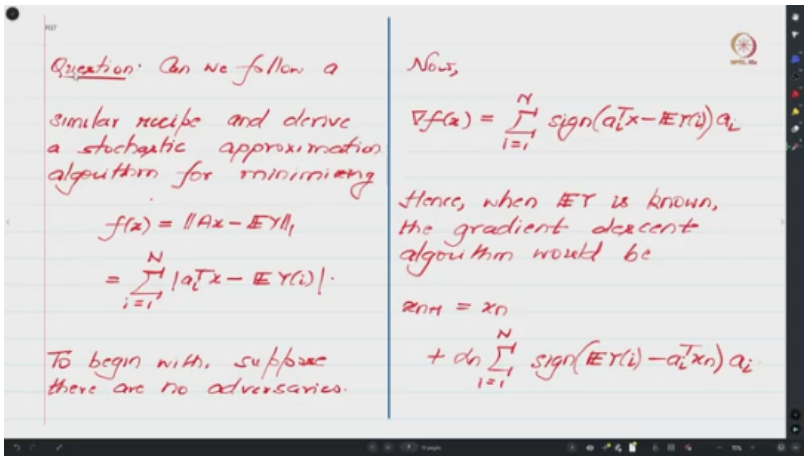
Instead, what was available to us were these IID samples of Y. And if you think of the stochastic approximation variant of the Karks-Marx algorithm, what we did was wherever this expected value of Y appeared, we replaced it with the sample And wherever we had i, we again replaced it with a sample that is chosen uniformly randomly from the index set. And that led us to the stochastic approximation variant of the Karks-Marx algorithm. So let me just quickly take you back to that update rule. So you see, wherever you had expected value of i, I replaced it by the sample and wherever I had i, I replaced it with this randomly chosen index.



So, one alternative way of looking at this stochastic approximation variant of the Cox-Marx algorithm is to think of it as the stochastic gradient descent algorithm with

respect to this objective function over here. Now, this looks interesting because you know this in some sense tells us a recipe to construct stochastic approximation algorithms. In particular, it tells us a recipe to construct stochastic gradient descent algorithms. So, one can ask this works well when there are no adversaries and we are working with the L2 norm in the objective function. Recall that in the presence of adversaries we would like to you know work with the L1 norm of this objective function right and one can ask can I you know construct the stochastic gradient descent version I mean the stochastic gradient descent to minimize that algorithm that objective function itself.

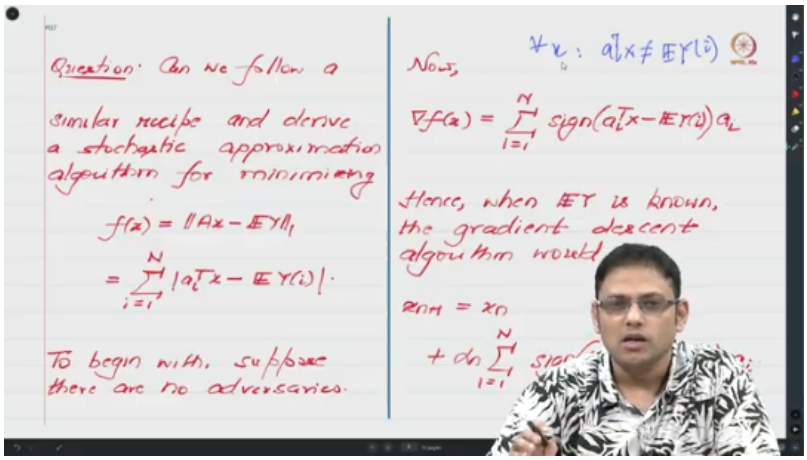
So that is what I have written here. So the question that one can ask is: Can we follow a similar recipe to what we followed in the previous couple of slides and derive a stochastic approximation, or more specifically, a stochastic gradient descent algorithm for minimizing this objective function? Notice that here we have 1 instead of 2. So, as we did in the previous slide, this objective function can be expanded in this fashion—that is, we can write this objective function as the sum going from i equals 1 to capital N, and you have the absolute value over here. Notice that unlike the case where we had the L2 norm, here there is no square. So, that is the difference between an L1 norm and an L2 norm.



So again, one can ask: Let us first look at the gradient descent of the algorithm, presuming knowledge of the expected value of y. Then, what we will do is, in order to derive a stochastic gradient descent, wherever the expected value of y appears, maybe we will just try to replace it with its samples. So that could be one approach to take. So, towards that, if you try to look at the gradient of this function, we will end up with

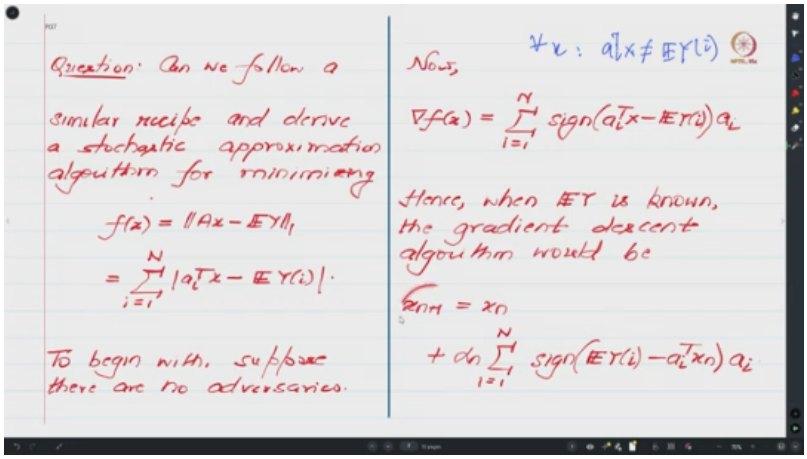
something like this because you have an absolute value over here. Of course, this function is not differentiable—I mean, the absolute value function is not differentiable where the argument is 0. So, except for that scenario—that is, when the argument is not 0—this function is differentiable, and the value of the derivative is either plus 1 or minus 1, depending on whether the argument is positive or negative. So, with that concept in place, one can see that the derivative of this expression equals this.

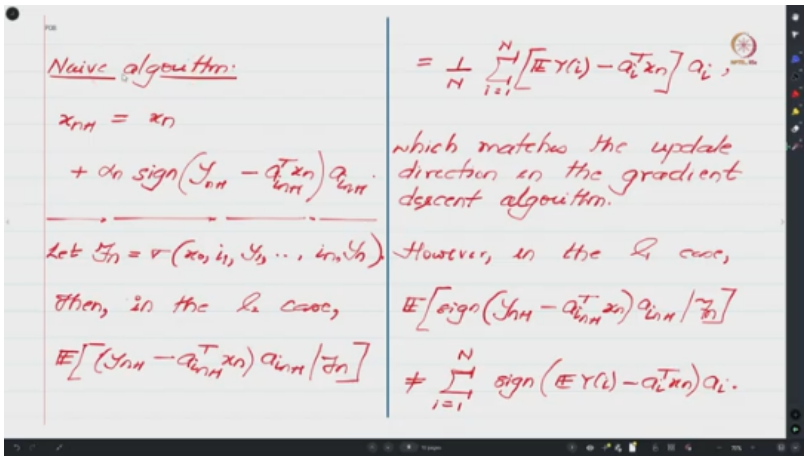
x such that, you know, Ai transpose x is not equal to the expected value of y of i, right? So, let us, you know, try to work with such a scenario, and then we will worry about what to do when Ai transpose x equals the expected value of y. So, let us presume that for every x, where these arguments are not 0, right? This gradient is well-defined, and let us see if we can use this expression for the gradient to come up first with a gradient descent algorithm and then with a stochastic gradient descent algorithm. So, with regards to coming up with a gradient descent algorithm, let us first presume that we have knowledge of this expected value of the y vector.



So, if this vector is known, then the gradient descent algorithm is given over here, which is basically xn plus 1. equals Xn plus some step size times the negative of this gradient. So, since we want the negative what I have done is whatever is this expression notice that I have flipped it that is I have taken this expression multiplied with minus 1 to get this expression and this then can be viewed as a gradient descent algorithm for minimizing this objective function. So, with this update rule in place, one can ask can I come up with a stochastic gradient descent version of this algorithm to tackle the case where we do not

know expected value of y. Instead, we have access to samples of this vector y. So the naive approach to come up with a stochastic gradient descent algorithm is given over here.



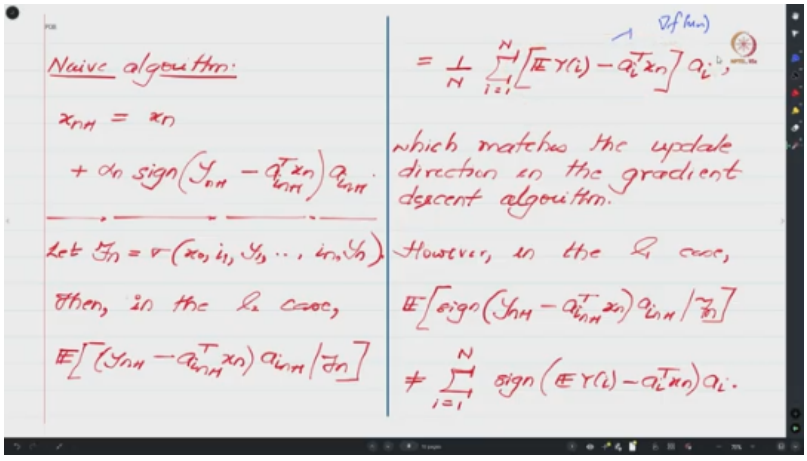


So this algorithm has the update rule xn plus 1 equals xn plus something. But observe that here instead of putting in expected value of y or some appropriate coordinate of expected value of y, I have replaced it with the sample of that vector. So, one can say this looks like a nice algorithm, can I work with this algorithm as in the Cox-Marx algorithm case. That is in the L2 case, we just took the gradient descent algorithm wherever expected value of y was there, we replaced it with a sample and using your standard stochastic approximation theory, one can show that that algorithm converges to the desired place. So one can ask in the L1 norm case as well would such an algorithm behave or help us you know estimate the desired quantity which is or you know minimize

the L1 norm of AX minus expected value of Y. Unfortunately that is not going to happen and the reason for that is if you define FN in this fashion that is the information that we have at time N then in the L2 case

You know, whatever your update was, if you take its conditional expectation with respect to the information that you have at time n, then this expression equals this and this quantity over here matches the true gradient at x. So, this quantity over here matches the true gradient at Xn. Sorry, I said X, I should have said Xn. So, this quantity matches grad F of Xn where your F is defined using the L2 norm. So, one can show that this matches I mean we may have to play with these constants a little bit but other than the constant the expression for grad f of xn you know is given by this expression over here.

And because of this reason, one can think of as the Cox-Marx algorithm as trying to do a noisy gradient descent. That is exactly what is stochastic gradient descent. But the conditional expectation of the update is aligned with the true gradient. That is what you know in some sense makes your algorithm work because our stochastic approximation theory somehow ensures that the cumulative effect of the noise is asymptotically cancelled out and hence the update direction is eventually detected, eventually directed along the true gradient, negative of the true gradient direction. That is why in some sense one can say that the stochastic approximation variant of the Kaksmark's algorithm works.

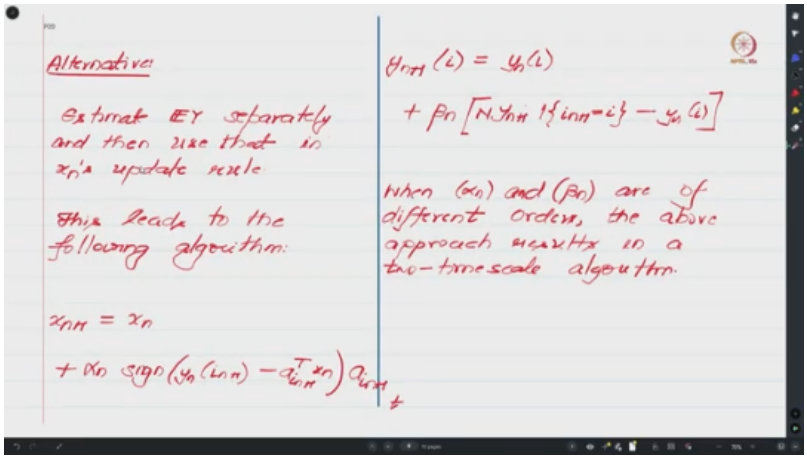


However, in the L1 case, if you take the conditional expectation as we did here, this will not equal the gradient of your L1 objective function at xn. The reason being that you know the sine function is not a linear operator and because it's not a linear operator

expected value of sine is not equal to sine of expectation that is you can't interchange expectation of sine. If we could do that then of course you would end up something like this because notice that this expectation if it were allowed to go inside then the expectation of this quantity will be expected value of y of i where you know i n plus 1 is sampled uniformly randomly.

So, maybe I should add 1 over n here right. So, this will be 1 over n right. So, unfortunately this conditional expectation will not equal this. And which implies that right you know this algorithm that we have over here this naive algorithm in an asymptotic sense is not behaving like the gradient descent algorithm right. If we had such a thing then indeed you know this algorithms behavior would be at least in an asymptotic sense.

mimic that of your stochastic gradient descent algorithm sorry not stochastic rather the gradient descent algorithm with respect to the L1 objective however since this conditional expectation does not equal this right even asymptotically this expression will not mimic the behavior of your gradient descent algorithm with respect to the L1 objective so this is the reason why this naive algorithm right is not principally constructed right So, how do we handle this? And that is where we come up with two time scale stochastic approximation algorithm. So, the idea simply is that we estimate expected value of y separately and use the estimate in Xn's update. Basically, what you do is you write down the same update for Xn except that wherever you had calligraphic Yn plus 1, which was like one sample of Y of In.



So Y of In, one sample of that. So instead of writing capital Y of In or calligraphic Yn, which is basically a sample of this, what I now do is I have little Yn. And here I have, maybe I should be careful, I should have placed i n plus 1 here. Let me write this properly. So this is i n plus 1.

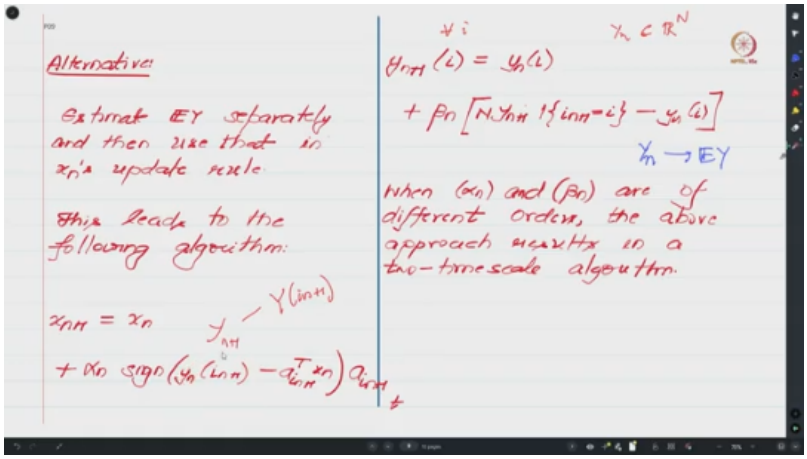
So y n plus 1 was basically a sample of this thing and in my previous algorithm I had directly placed this quantity over here. But here instead what I am doing is instead of plugging in calligraphic yn plus 1, I am plugging in the in plus 1th coordinate of little yn. where this little yn is updated separately, right? In particular, this quantity tries to estimate expected value of y. In particular, you know, I should be careful here. So, there are some typos.

This is just yn. Sorry, I should say this was correct. Okay, so I should say for all i, you update your yn of i in this fashion, right? So, what is happening over here is that your yn is, okay, is a vector whose dimension equals, you know, the dimension of your expected value of y, and you update the i-th coordinate of yn in this fashion, right?

So, we are updating it in this fashion, and you know, in order to construct this, I am at this point presuming that there are no adversaries. I will first, you know, come up with a stochastic gradient descent-type algorithm for minimizing the L1 objective, and then in the next class, we will see how to handle adversaries. So here, this little yn in some sense is trying to estimate the expected value of y. I will soon tell you why it estimates that. And whatever is the yn-th estimate at time instance n, I will look at the i n plus 1-th coordinate of that and use it here. So, if you remember, if I had calligraphic Yn and I took the conditional expectation of this expression, the expectation and this sign could not be interchanged, and that sort of prohibited me from concluding that you are somehow making use of an estimate of the expected value of Y over here.

In contrast, observe that here, since I have replaced this calligraphic Yn with little yn, and if I am doing this algorithm appropriately, one can hope that this yn actually converges to the expected value of y, right? And this part will ensure that at least as n goes to infinity here, I am actually working with the expected value of y. And since I am asymptotically working with the expected value of y, this algorithm, that is the Xn update, can be

presumed, at least in an asymptotic sense, mimicking the behavior of a stochastic gradient descent algorithm with respect to your L1 norm. Now, let me quickly tell you why this algorithm is one of the natural ways to estimate the expected value of Y. So, again, let us look at the conditional expectation of this update rule, and since we have presumed that your I n plus 1, you know, is sampled uniformly randomly.

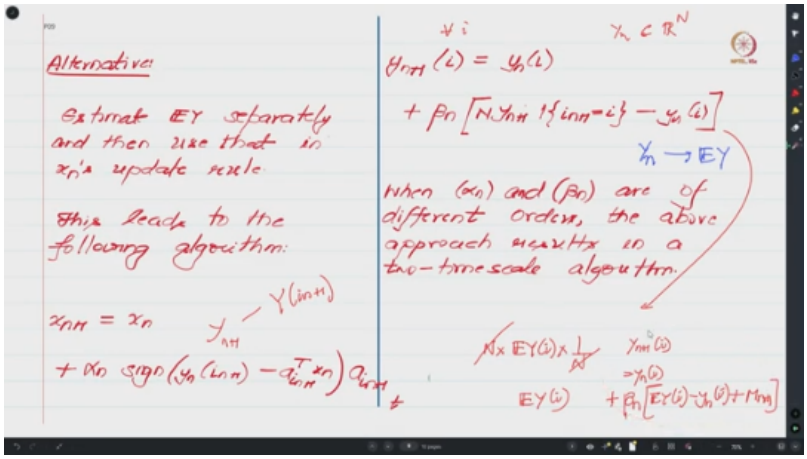


So, I_N plus 1 will equal I with probability 1 over N, right? And this is an indicator that at time instance N, I_N plus 1 equaled I, right? So, if you take the conditional expectation of this with respect to the information that you have at time instance N, one can see that the conditional expectation will actually be summation i equals 1 to n, right? And n times, okay, now your I_N plus 1 could be one of the many indices, right?

And since we are focusing on, so maybe I will write it as j equals 1 to n. So, one can say that this expression is actually the expected value of Y of j and you know the probability that I_N plus 1 equals i is basically 1 over n, and hence this expression will actually be times 1 over n, right? And this n and this n will cancel off. Let me just make sure I am doing it correctly. Sorry, you know, let me do this properly. So, for a fixed i, this expectation will basically equal n times the expected value of Y of i times 1 over n. One can check that the conditional expectation of this with respect to the information that you have at time n and using all your independence properties. One can show that the conditional expectation of this expression is actually this, which would imply that these n's cancel off and we have the expected value of Y of i over here. And in this sense, this

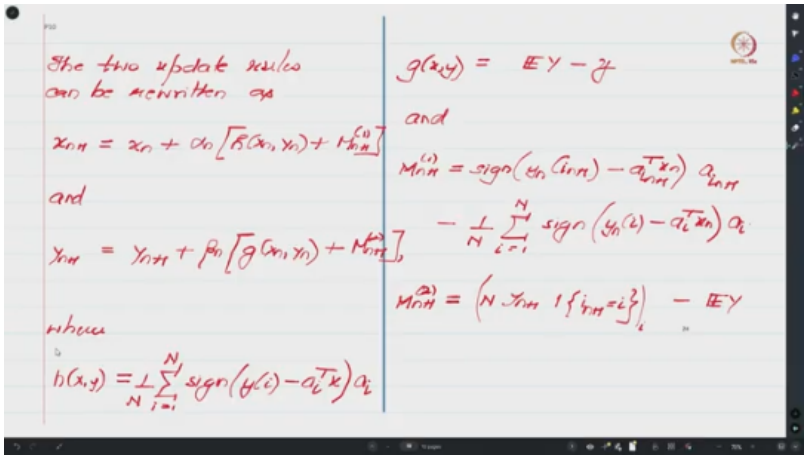
update rule can be rewritten as Y_{n+1}i equals Y_n of i plus beta_n times the expected value of Y of i minus Y_n of i plus some noise sequence.

Is this okay? And this expression, if you had just this expression, one can see that this quantity in that spirit tries to compute the expected value of Y, the ith coordinate of the expected value of Y. In particular, if you sort of apply the stochastic approximation analysis, in particular the ODE method that we have studied so far for this update rule, one can see that this update rule indeed tries to estimate the expected value of Y of i. So, in other words, the ith coordinate of Y_n tries to estimate the ith coordinate of the expected value of Y in an asymptotic sense. And by making use of this quantity within the sine function, we are ensuring that as n becomes larger and larger, we are indeed working with an estimate of the true gradient. That is what ensures that this update rule in an asymptotic sense behaves like a stochastic gradient descent with respect to the L1 norm.



So, what one can do is we can you know take these two update rules for Xn and Yn and rewrite it in this fashion right. This is some exercise that you can do where one can see that H of X, Y has an expression like this and the expression of G of X, Y has an expression like this and your noise terms have expressions like this. So, this is something one can check and you can see that now your Xn plus 1 and Yn's they are updated using two different step size sequences and because you have two different step size sequences. One can conclude that you know this algorithm that we have has a two time scale nature.

In particular, whenever your alpha n's and beta n's are updated using two different step size sequences, we will say that we have a two time scale algorithm.



So, this brings me to the end of this class. Let me quickly summarize what we have done in today's class. We looked at minimizing the L1 norm of Ax minus expected value of y and we saw that if you try to come up with a naive gradient descent algorithm or a naive stochastic gradient descent algorithm, the expectation and the sine function do not interplay with each other in that the expectation and sign cannot be interchanged and because it cannot be interchanged this stochastic algorithm does not behave like the stochastic gradient descent for minimizing the L1 norm. To counter this issue, what we do is we estimate these expected value of y separately, right?

And whatever is that estimate, we directly plug in into the sign expression, right? And by ensuring that this alpha n and beta n are step sizes of different orders, we end up with a two time scale algorithm. In the next class, we will see what role do these step sizes play and how can such an algorithm be analyzed. I would also like to highlight that in the later half of today's discussion, we sort of did not consider adversaries. That was just to keep things simple.

In the next class, we will ask what if we had adversaries and how this algorithm will tackle the adversarial measurements, and so on and so forth. So that, in some sense, it will be taken care of by what is known as stochastic recursive inclusions, and that will be discussed in subsequent classes. Until then, goodbye, thank you, and namaste.