Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan

Department of Mathematics

Institute of IIT Kanpur

Lecture: 54

Hello everyone, this is lecture 4 in week 11. Recall that in the previous lecture, we were discussing in detail about Karmarkar's algorithm. If you recall the structure, we first convert the given problem into canonical form, take an initial value, and use a projective transformation to convert it into (1/n, 1/n, ..., 1/n).

The transformed problem actually becomes a non-linear program, but since the solution is 0, we can treat it as a linear program. Then we choose the direction and the step size. The direction is chosen by projecting the steepest descent direction to the feasible polytope, and the step size α was not discussed in detail, but it comes from Karmarkar's analysis. We then use the reverse projective transformation to convert it back to the x domain. We repeat this until the objective function value is close enough to 0.

All of this works very well if we know that the optimal value of the objective function of the canonical form is 0. Even if we know the optimal value of the standard form, we can use $-v^*$ and get away with it. But what if we do not? In that case, we started with this problem. We have two issues. The first issue is regarding the transformation. I discussed that when you choose the little a vector as all zeros and one, we are somehow getting rid of this issue. But such an issue, we cannot get around because now this will actually become

$|c^T X_k - v^*| >$ tolerance and v^* is not something that we know.

No problem. We will just use the technique that we used in the previous affine scaling method, the duality technique.

If you recall, the problem that we have at hand is the canonical form of this form: minimize $c^T x$ subject to $A x = 0$, $a^T x = 1$, and $x \geq 0$.

This is the problem that we have. Let us write down the Lagrangian.

We will have $c^T x + \mu^T A x$.

I would like to write it as $b - A x$.

Here I will write it as $-\mu^T A x$. If this is an $m \times n$ matrix, this will have another μ. I will write that as μ_{m+1}. So that is $+ \mu_{m+1} (1 - a^T x)$ or $-\lambda^T x$; that works better.

If we had written these two together maybe we would have been better off. Since we have here m equality constraints and one equality constraint, in total we have m+1 equality constraints. We will write down a matrix $\tilde{A}$ which is A and a^T, and the b here turns out to be $\tilde{b}$ which is m zeros and a 1. So $\tilde{A}$ is now an $(m+1)\times n$ matrix and this is an $(m+1)\times 1$ matrix. I can write this as minimize $c^T x$ subject to $\tilde{A}x = \tilde{b}$ and $x \geq 0$. Note that there is no change in x; I have just changed the representation of these m+1 constraints in this form.

If you actually write down the Lagrangian,

$L(x, \lambda, \mu) = c^T x + \mu^T(\tilde{b} - \tilde{A}x) - \lambda^T x$.

When you differentiate this with respect to x, you have $\tilde{A}^T\mu + \lambda = c$.

You will get the transpose of all this. You will get $c^T - \mu^T\tilde{A} - \lambda^T = 0$.

$\partial L/\partial x = 0$ implies this.

I am going a little fast because this is something that we have derived earlier in the affine scaling method. You have $\tilde{A}^T\mu + \lambda = c$.

That is fine, not a problem, and the complementary slackness condition is that $\lambda_i x_i = 0$ for all $i = 1$ to n.

This tells us that if $x_i > 0$ then $\lambda_i = 0$ and thus $(\tilde{A}^T\mu)_i = c_i$, and if $x_i = 0$ is one other possibility. What you know is that this tells us that $x_i (\tilde{A}^T\mu - c)_i = 0$, because if $x_i > 0$, this is 0.

If $x_i = 0$, anyway the product is 0.

This is true for all $i = 1$ to n. If you want to write this in the form of a matrix, it is

$X (\tilde{A}^T\mu - c) = 0$.

This is exactly the condition that we are actually going to use.

In fact, we will use this.

When you have the optimal value of x and the optimal value of μ, you know that this condition is satisfied where $x_i (\tilde{A}^T\mu - c)_i = 0$.

If this is true, the summation is also true. So that is $\Sigma\, x_i (\tilde{A}^T\mu - c)_i = 0$.

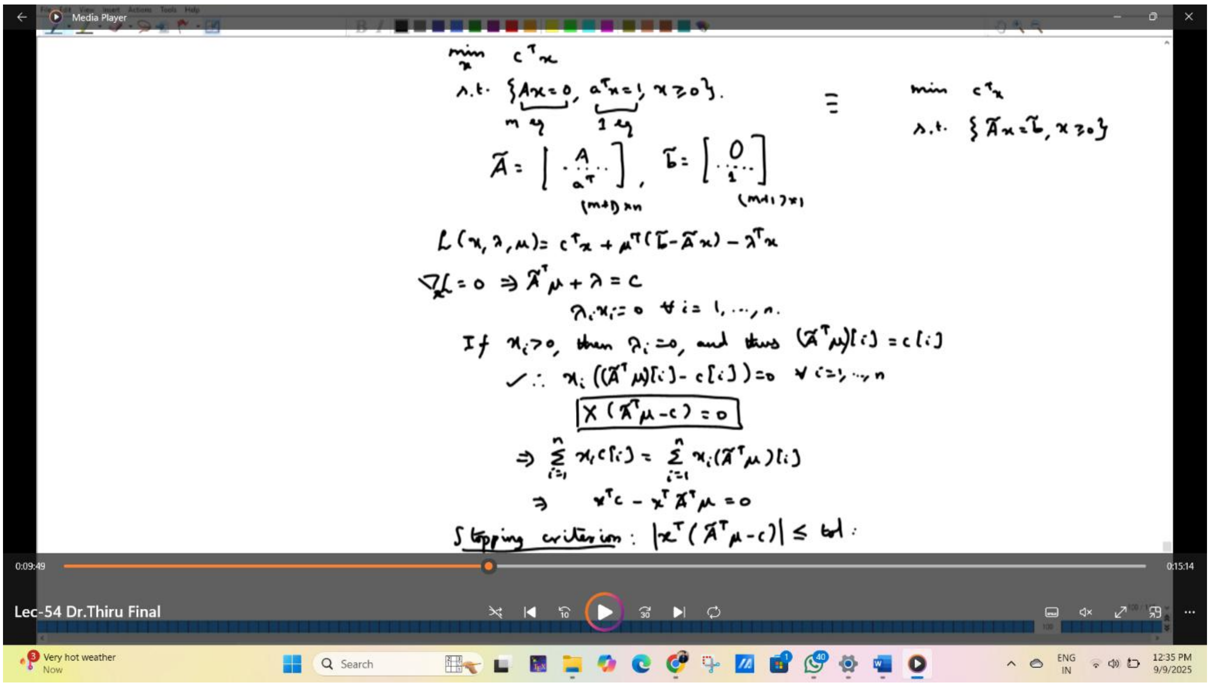
If this is true, of course the sums are equal, and what this means is that $c^T x - x^T\tilde{A}^T\mu = 0$.

That is exactly what I am going to check.

The stopping criterion will actually be $|x^T (c - \tilde{A}^T\mu)| < \varepsilon$.

If I modify the constraint here as what I just mentioned now, in that case we will actually have the algorithm ready for Karmarkar's method.

(Refer Slide Time 9:49)



Going through it once more: you are first converting the given problem to canonical form. This step is mathematical. I need not have written this right. Here I will actually replace this with initializations. You first initialize x_0, of course, $k = 0$.

Converting into canonical form actually means you are given A, little a, and c.

If you recall what the canonical form is, you have c, capital A, and little a; you do not need anything else. And this is x_0, $k = 0$, and I can write it here, capital $X_k = \text{diag}(x_k)$.

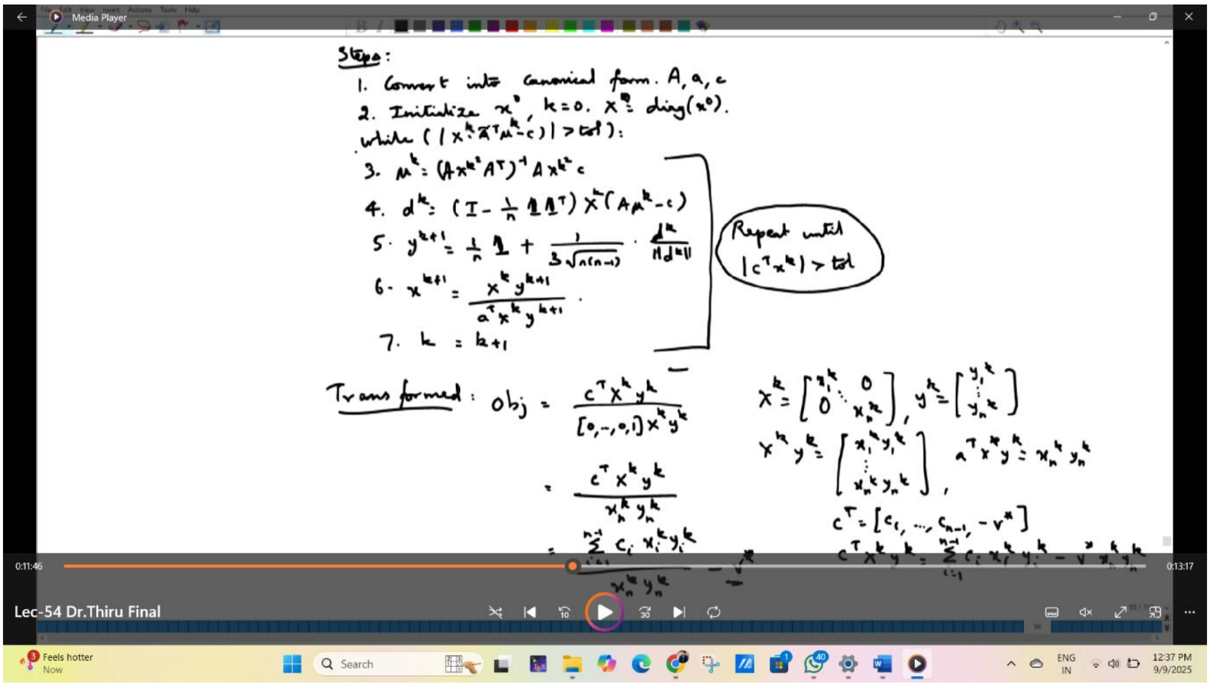
Since I am not using a bracket here, I will just write it this way.

This gets repeated. The condition is slightly different. You are going to do while

$|x^T (\tilde{A}^T\mu - c)| > \text{tolerance}$.

That is changing.

(Refer Slide Time 11:46)



This is no more the condition. The condition is this, and you need to do an indent, but I think you can understand all of that from whatever I have written. Let us actually work out one example, and we will start with our favorite example, which is

maximize x_2

subject to

$x_1 \geq x_2$, $x_1 + x_2 \leq 1$, and $x_2 \geq 0$.

If you recall the standard form of this is just to note what is going on, you are actually maximizing x_2 when the feasible polytope is a triangle with vertices (0,0), (1,0), and (0.5,0.5).

The standard form is

minimize $-x_2$

subject to the condition that A is the following: it is [-1, 1, 1, 1, 0], [1, -1, 1, 0, 1] and this was x_1^+, x_1^-, x_2, x_3, and x_4, and this was [0, 1], and of course all of this has to be ≥ 0, all five components have to be ≥ 0.

This is the standard form.

In canonical form, what we have to do is first introduce one more variable x_5 which is equal to 1 always. This has to be brought in here.

So this equation becomes $-x_1^+ + x_1^- + x_2 + x_3 = 0$.

That is equal to 0, and the second equation becomes $x_1^+ - x_1^- + x_2 + x_4 - x_5 = 0$,

and you add another constraint where $x_5 = 1$.

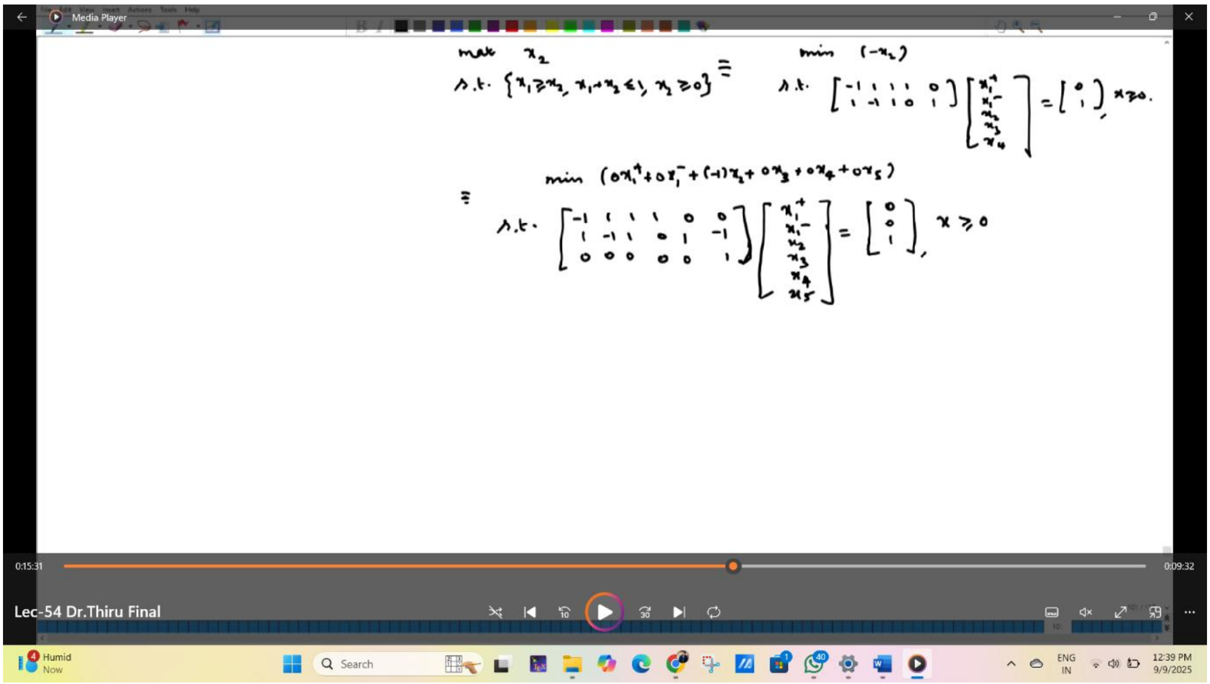
The canonical form turns out to be: you will have 3 rows. The first is [-1, 1, 1, 1, 0, 0], the next is [1, -1, 1, 0, 1, 0], this is 0 and -1, so x_5 needs to be added, so this is 0 and this becomes -1, and here you just have all 0s and 1.

This is x_1^+, x_1^-, x_2, x_3, x_4, and x_5. And here it is actually [0, 0, 1].

And this is $x \geq 0$, nothing to change here. Since we are looking for c as well, I will change that as well. This is $0\ x_1^+ + 0\ x_1^- + (-1)\ x_2 + 0\ x_3 + 0\ x_4 + 0\ x_5$.

I will assume that we do not know v*. I will just put it as 0 anyway; it does not matter; we are convinced that the answer will arrive.

(Refer Slide Time 15:31)



Let us write down the code for this particular example.

This is for Karmarkar's method. We start by importing NumPy, and A is the 3×6 matrix which I just wrote down: [-1, 1, 1, 1, 0, 0], [1, -1, 1, 0, 1, 0], and the last is [0, 0, 0, 0, 0, 1].

This is A, and a is just the last row.

I will just write it as A[-1], the last row.

And c is two 0s, -1, and three 0s. Now we need to choose a starting point.

We will choose (0.1, 0.05), which we chose last time. You have to fill the other ones as well, and this we will write as 1. So $x_5 = 1$.

But these, so when I write $x_1 - x_2 \geq 0$, the remaining is 0.05.

That is the reason you need basically you need $A\ x = 0$ for the first two rows.

You can see that it is -0.1 + 0.05 + 0.05 that is 0.

Now here you have 0.1 + 0.5 is 0.15; I am sorry this should be -1.

So now it is 0.1, 0.05, 0.85, and -1. So that's 0 as well.

All right, so now capital X is the diagonal of x.

I can write k = 0 and $\mu = (A X^2 A^T)^{-1} (A X^2 c)$.

All of them are defined before. I will print the required values k, x, and the stopping criterion. The stopping criterion, if you recall, is np.abs(x @ ($A^T\mu$ - c)). That is $A^T\mu - c$.

Again, that is going to be our constraint; the absolute value of x @ ($A^T\mu$ - c) should be when it is greater than the tolerance you continue.

The first step is choosing d. d, if you recall, is $(I - (1/n)\, 1\, 1^T) (X (A^T\mu - c))$.

I will show that to you: $(I - (1/n)\, 1\, 1^T) (X (A^T\mu - c))$.

This is np.eye(n) - (1/n) * np.ones((n, n)). So 1 into 1^T: 1 is a column vector of 1s, 1 is a row vector of 1s.

When you do a matrix multiplication, you have a matrix of all 1s.

So $1\, 1^T$ is np.ones((n, n)). I have to use another bracket, and I divided by n.

So that is $I - (1/n)\, 1\, 1^T$. You multiply that with x and you multiply that with $A^T\mu - c$.

That is d. Now, y is np.ones(n)/n + (1/(3√(n(n-1)))) * (d / np.linalg.norm(d)).

I have just written step number 5: $y_{k+1} = (1/n)*1 + (1/(3\sqrt{n(n-1)})) * (d_k/\|d_k\|)$.

That is exactly what I have written here. We have written down the expression for y. Now we have to do the reverse projective transformation, which is $(X y) / (a^T X y)$.

Again, it is just (X @ y) / (a @ (X @ y)), nothing else.

Now, you have to write k = k + 1, and of course, the μ, capital X has to be the new diagonal matrix and μ needs to be replicated.

Again we will print the usual values k, x, and the absolute value of x @ ($A^T\mu$ - c).

We have replicated all the steps. Let us see if it works.

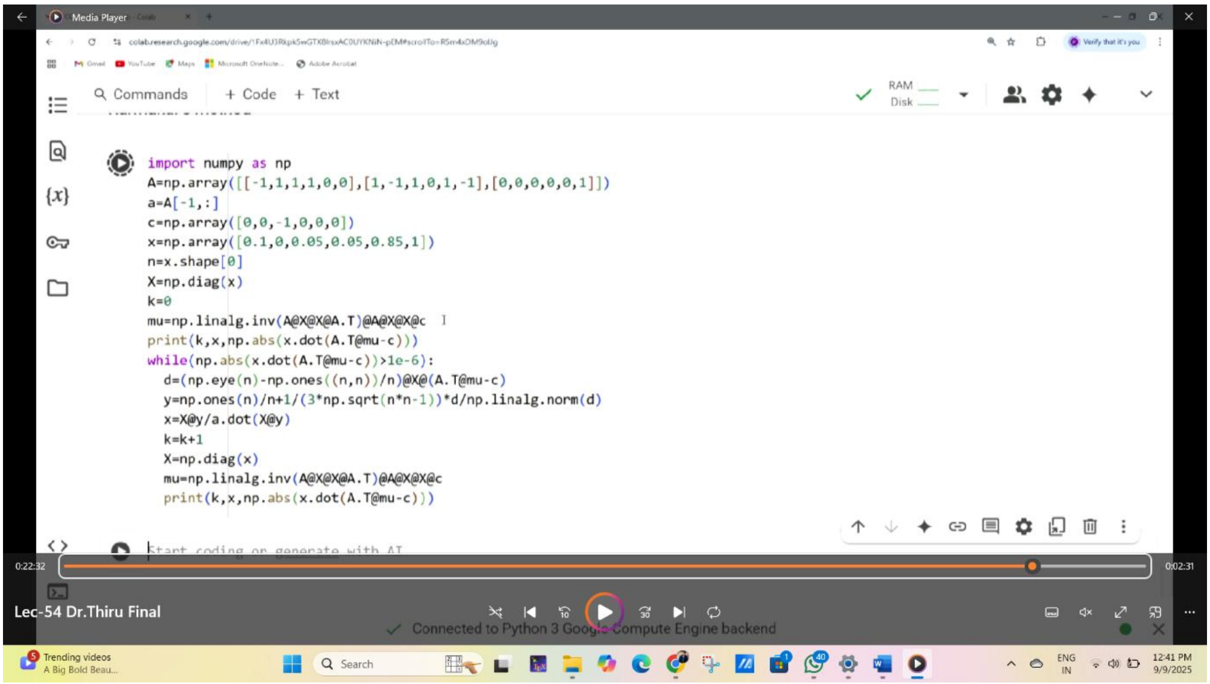
You can see the answer here. It is (0.5, 0, 0.5, 0, 0, 1).

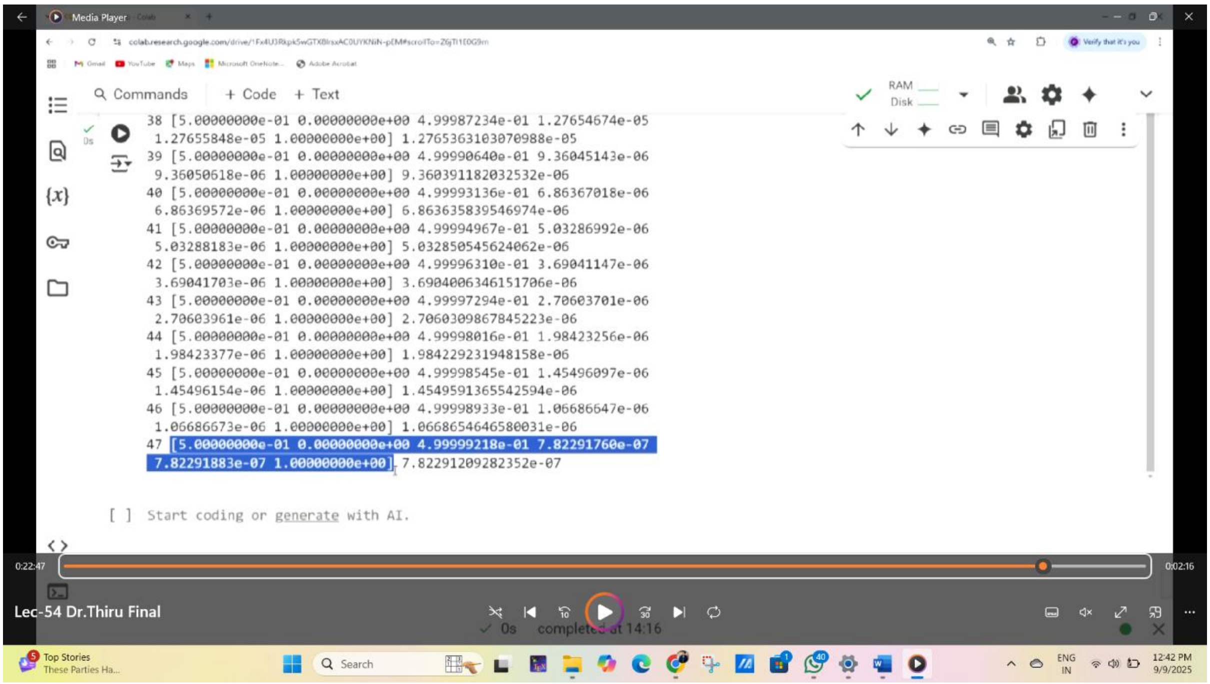
The last one is supposed to be 1 anyway.

So we just have 0.5 and 0.5 as the right answer.

We got the answer in forty-seven steps. You can see that the algorithm is very similar to what you had for the affine scaling method.

(Refer Slide Time 22-23)





It is just that instead of using a linear transformation, we are using a projective transformation, and instead of the standard form, we are using the canonical form.

Because of this, you also have a change in the stopping criterion. The stopping criterion turns out to be the absolute value of x @ ($A^T\mu$ - c) should not equal the tolerance.

One of the key things that works here is the value of α. The value of α is key in having the polynomial algorithm.

The number of steps should not go to 2^n like what we had for the simplex method. Here we have a guarantee that such things will not happen. In the next lecture, we will work out a few more examples, and I will then let you know about the polynomial order that Karmarkar's algorithm is boasting of. With that, we will end our discussion on Karmarkar's algorithm. Thank you.