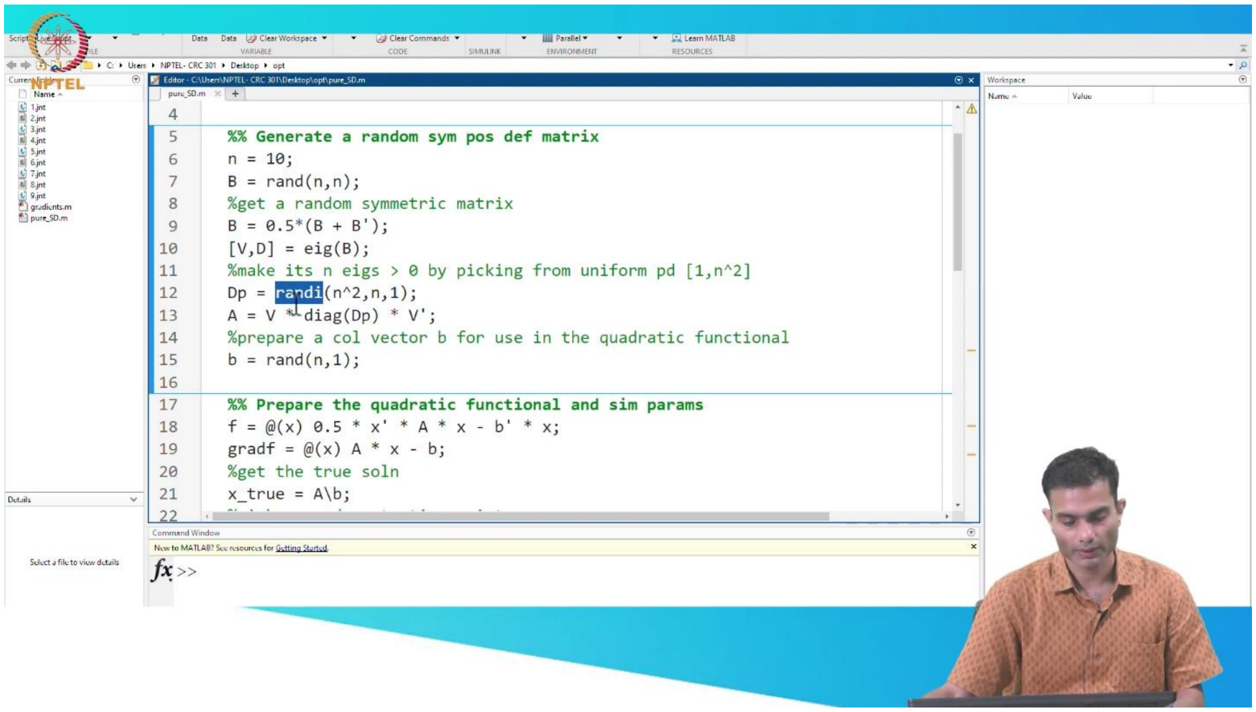
Course Name: Optimization Theory and Algorithms
Professor Name: Dr. Uday K. Khankhoje
Department Name: Electrical Engineering
Institute Name: Indian Institute of Technology Madras
Week - 05
Lecture - 32

Implementation of an optimization algorithm in MATLAB

Okay. Any questions? Something not clear over here? If not, what we'll do is let's have some numerical fun. Let's look at the implementation of so far what we have done, right? So this is your first optimization algorithm that you've learned, which is gradient descent. Now you are convinced also that it should work in theory not just numerically it should work in theory. Let us look at how we would code a simple implementation of it and you know there might be some surprising things which you would not expect. So let us fire up MATLAB over here.



```
%% Generate a random sym pos def matrix
n = 10;
B = rand(n,n);
%get a random symmetric matrix
B = 0.5*(B + B');
[V,D] = eig(B);
%make its n eigs > 0 by picking from uniform pd [1,n^2]
Dp = randi(n^2,n,1);
A = V * diag(Dp) * V';
%prepare a col vector b for use in the quadratic functional
b = rand(n,1);

%% Prepare the quadratic functional and sim params
f = @(x) 0.5 * x' * A * x - b' * x;
gradf = @(x) A * x - b;
%get the true soln
x_true = A\b;
```

Now, in the same line as the proof that we did for the rate of convergence, I want to construct an artificial example first. That artificial example is going to be the same,

$$\frac{1}{2}x^\top Qx - b^\top x$$

But where will I get my Q and b from? Let me just generate it randomly. Now, if I am going to generate it randomly, what do I have to be careful of? If I generate an $n \times n$ matrix randomly, will it work? It won't work.

Why? It need not be positive definite. So I need to do a little bit of tricks to make it positive definite so that I could work with it, right. So this line over here rng is to initialize the random number generator, yes question? Even bigger, okay. Is it better now? Okay. So line number two is essentially telling you to initialize a random number generator with the same seed.

Why would I want to do that? So that I can reproduce the results. Every time I run this code, I should be able to reproduce. If I want to do debugging and analysis, this is very useful because I don't want to every time get a new random matrix. Once I've got confidence on how it works, I can knock this line off. But this is a good way of reproducing your results.

Now I want to generate a random symmetric positive definite matrix. So this trick is very simple. If I just take, let's say I'm working in 10 dimensions. So if I do this, rand(n, n) is going to give me a random $n \times n$ matrix. If I want to make it symmetric, this is all I need to do.

$$\frac{1}{2}(B + B^\top)$$

is going to make it symmetric. No problem. Then what I do is, to make it positive definite, what is it that I need to play around with? I have to make sure eigenvalues are positive. So I take my B matrix and this I is basically going to calculate what? Eigenvectors, eigenvalues. Eigenvectors will fall into V and eigenvalues will fall into D, right? It'll make it a diagonal matrix, right? Now, what do I need to change in order to make it positive definite? I need to make sure that this D has positive entries in it.

So all I do is I use this randI function which is random integers, give me n random integers, give me some range is given. What do I do with that? I construct a new matrix A which has the eigenvectors from before but I have replaced the eigenvalues to be these random integers, random positive integers, greater than 1, therefore it is always going to be positive. This is just one way of doing it. You could just manually put in the numbers. You could do whatever you want because we are constructing a synthetic example, right? So this is, so my A matrix, therefore, by I've constructed its eigenvalue, I've used its eigenvalue decomposition to ensure that it is positive definite, okay? Then what do I have? I needed a random b vector.

Is there, this is the simplest way, just n random numbers, right? Now here is a shorthand way of writing a function in MATLAB. For those of you, who is not familiar with this way of writing, just seeing it for the first time? Okay, very good. So let's just look at what is happening. So I am saying f, okay. At the rate x simply says that what I am going to define after here is a function of x.

x is a variable, right? And after this whatever follows is the body of the function. You could also write this as a standalone m file as a function. It's the same thing. This is the much shorter way of doing it.

So whatever follows after this at the rate x is the definition of the function. Could be $\sin x$, $\cos x$. In this case I have

$$\frac{1}{2}x^\top Ax - b^\top x$$

That's the example that we had looked at. So all of this hard work was required to do what? Just construct the cost function.

I can also explicitly write the gradient $Ax - b$. Is it a function of x? Yeah. Alright. So I have

$$\nabla f$$

Now I also know the true solution, right? The true solution is simply what? The true solution is the point at which the gradient of f goes to 0.

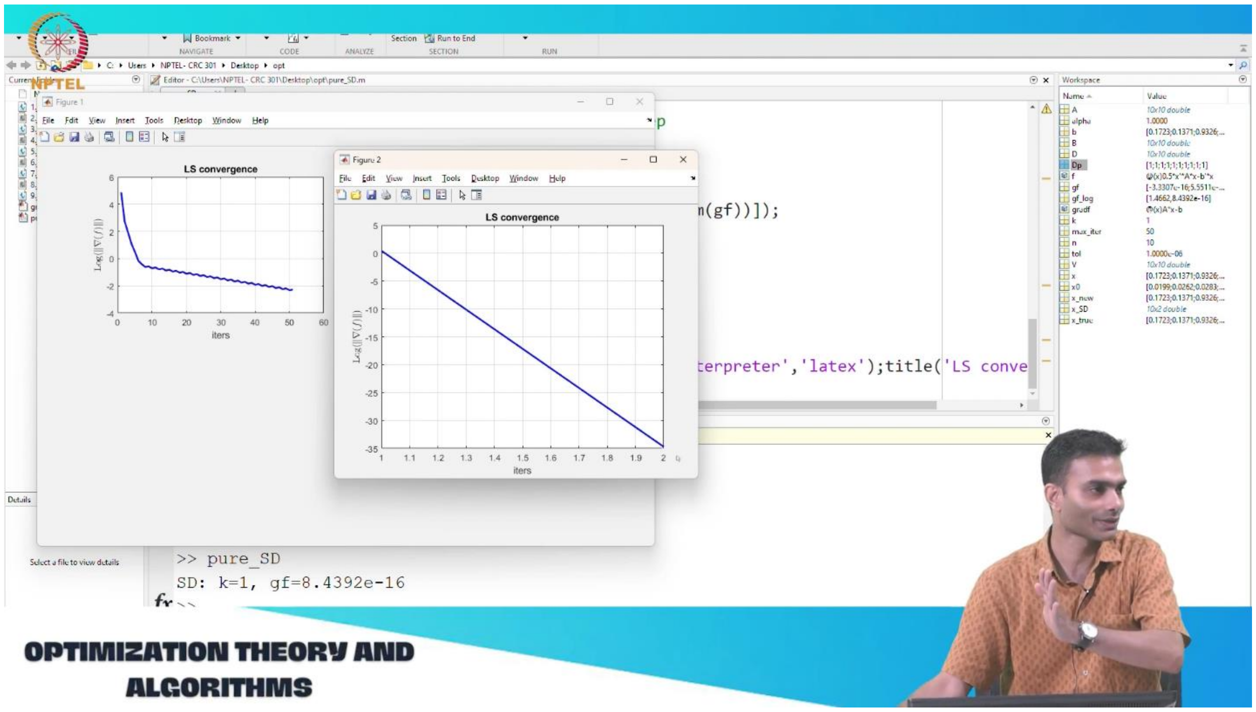
$$\nabla f = 0 \Longrightarrow Ax = b$$

Therefore, the solution is $x = A^{\top} b$. I am calling it x_{true} just to make sure that I know what it is, right? Now, I pick a random starting point. x_0 is my random starting point.

What is it? n random numbers. So, start from anywhere in n-dimensional space. Let us start from there. Now, getting into a little bit more practical territory, I need to define how many iterations at max are permissible. So, that is a variable which I call, let us say, max_iteration.

For example, I have set it to 50, just for illustration, it could be whatever you want. And a tolerance is a threshold below which I am going to declare success. So, when the norm of ∇f goes below this threshold, I will say, okay, this is good enough for me. I may not get 0 on machine precision. If I get a small number, it is okay.

I put 10^{-6}. If your application needs 10^{-12}, you define 10^{-12}. Okay. Yes. Why? Because I know it analytically. I could have used the gradient function to get it, I could have done it.



```
n(gf))]);
terpreter','latex');title('LS conve
```

```
>> pure_SD
SD: k=1, gf=8.4392e-16
```

Name	Value
A	10x10 double
alpha	1.0000
b	[0.1723;0.1371;0.9326;...
B	10x10 double
D	10x10 double
Dp	[1;1;1;1;1;1;1;1;1;1]
f	@(x)0.5*x'*A*x-b'*x
gf	[-3.3307e-16;5.5511e-...
gf_log	[1.4662,8.4392e-16]
grudf	@(x)A*x-b
k	1
max_iter	50
n	10
tol	1.0000e-06
V	10x10 double
x	[0.1723;0.1371;0.9326;...
x0	[0.0199;0.0262;0.0283;...
x_new	[0.1723;0.1371;0.9326;...
x_SD	10x2 double
x_true	[0.1723;0.1371;0.9326;...

In this case because this is more like an illustrative example, I am writing it. But I could have used the gradient, Hessian, all of those things. Now we want to solve this using gradient descent and as we just now used, we will use exact step size. The update equation is:

$$x_{\text{new}} = x_{\text{old}} - \alpha \cdot \nabla f$$

where the expression for α is what we just wrote out recently, which you will also work out in the tutorial.

So how do I start? I need to use x as the variable that is updating, that is my iterate.

So I am initializing it with x_0. This x_{sd} is just a way for me to keep history, that at each point I don't want to forget what I had done. So I am just keeping track of it. So I am initializing it in this way. Similarly, the gradient is being written with this gf, and I am keeping a log of all the gradient values so that at the end of the process, I can analyze how the gradients looked like.

So k is my iteration counter. Now this is how you would write it. You would say that while the norm of the gradient is higher than my threshold and the number of iterations is less than the maximum number of iterations. A usual beginner's mistake would be to omit this line. If I omit this line and if my tolerance threshold is very strict, the algorithm could take, let us say, much, much longer than you expect because it's trying to get it low. But if it's entered a very flat region, it makes very, very small steps.

And you think, you know, my computer is hung or something. So this max iteration is just to prevent it from taking forever. And here is a step length analytical expression. I update my x, so

$$x_{\text{new}} = x_{\text{old}} + \text{step\_length} \cdot p_k$$

where $p_k = -\nabla f$, and ∇f was initialized over here, okay. Then, given the new x_{new}, I can calculate ∇f at the new x_{new} because as we have seen this is just a function of x, right? If you give me x, I will give you the gradient.

So I have calculated the new gradient and I am just updating my log variable so that I add the new norm over here and so on, right. So that is all there is in this. I am updating my x, I am updating my gradient, and I am keeping a log. That is all I am doing over here. And it will quit when the AND condition is met, so it is going to quit accordingly, okay.

Either I run out of iterations, right, and I'm going to plot it over here. So let's see, does it look like something you would expect? So what would you expect? I am plotting for example, the history of the norm of ∇f, right? What would you expect? So it's gonna start from a high value because I started with a random number. And this function is a quadratic function. It's positive definite.

So it's a bowl like this. So I know a solution exists, right? So am I gonna go, how do you think the trajectory would be? And if I ask you, what is the history of norm ∇f? One student says it should just go linearly down. Any other guesses? Like a hyperbola? Okay, let us see what it looks like. Let us run this.

It also spits out something over here. This is what the convergence looks like. So I am plotting $\log(\| \nabla f \|)$ instead of $\| \nabla f \|$. What do you expect? Log scale means something, right. So it is starting with something very high, and within the first 8 iterations, it has covered maximum ground, right? In log scale, it has gone from 5 to -0.5. That means, you convert it into linear scale, the maximum progress has happened in the first few iterations.

After the first few iterations, what is happening? For the remaining 40 iterations, it is covering only about 2 units in log scale. So, this is something very typical of gradient descent. You make a lot of progress in the beginning, and then you kind of enter into a slow gradient zone. So if I had wanted 10^{-12} and I didn't have a cap on the number of iterations, it would go on for a very, very long period of time. So this is one of the, if you want to call it a drawback or a feature of gradient descent, it is this.

So many times it's advantageous to just run gradient descent for a few iterations, get some improvement, and then if you have a facility to switch to a faster method, then switch into that. So here I was just plotting what is the norm of ∇f, so you notice in the first iteration we started with a very high value (15), and as we went down you can see the numbers are kind of just not really progressing, it seems to get stuck near 0.1. Very slow progress.

Okay. Yeah. But what is our expectation based on? Correct. Yeah, so that is why I said that you should expect some surprises over here. Right. Correct. So, I mean, this is one of the first surprises that you feel when you look at this trajectory. If you look at it in the log scale, actually, it is clear.

It is like an L shape. The first L is very sharp and then after the kink it is slower. So, if I look at it in linear scale, it will actually look like a hyperbola.

Question: Why is it wavy right now? Why is it wavy?

I do not have a good answer for you right now. This kind of small oscillatory behavior is known and seen in many numerical methods. Remember, this is not a function value. The function value will always decrease. This is a plot of the norm of the gradient. This is exact line search because I put α analytically equal to the exact line search angle, right? So this is as kosher as gradient descent can get.

For a circular contour, it was a single step for a circular contour. That would be when all the eigenvalues are the same. So actually, we can look at what the eigenvalues are over here, which are sitting here in dp.

So, these are my eigenvalues. Remember $\lambda_n - \lambda_1$, that expression is going to be some ugly number over here. Yeah, in fact, 2 is the smallest, 100 is the biggest. So the condition number is not very bad.

So those of you seeing this kind of code for the first time, anything unclear? This is a very standard way of defining functions in shorthand rather than having to separately define M files for the function and looping it in this way.

So, what is the condition number? Let us see. So what we could do is we could type in, for example, norm of, we started at x_0 and we went to x_{true}, right? So, it is 1.48 when I started, and when I ended, I think x should still contain the true solution, I mean the final solution. So if I do $x - x_{\text{true}}$, that gives the norm. So if I run it for more iterations, I can get even faster.

Pure quadratic convex? No. No, I do not think you can. Yeah. So, in fact, because it is a pure convex quadratic, you could just do Newton's method starting from x_0 and you will reach the solution in one shot. But anyway, the purpose of this is to illustrate how you code a real-life algorithm. I would not call it real-life because I am writing it exactly as the method is.

So, if you were writing this for a real-life problem, what would you change? α, I don't know how to calculate in a real-life problem.

So what would I replace α with? Backtracking line search is one way. I don't know α, so I will do backtracking line search until I get a good enough α, which satisfies, for example, Wolfe conditions. Actually, on a related note, you had asked the question yesterday. When we proved the convergence, we assumed a few things, right? The function is bounded from below, we assumed that α obeyed the Wolfe conditions. And so there was a good question about there being one relation for sufficient decrease and the second being a condition for curvature. Why should it be that there is an α that satisfies both?

It's not clear, I didn't prove it. It turns out that if C_1 is less than C_2, you can prove that there is an α which satisfies both. I'll post a proof for that on the class website. But that's all you need to make sure that an α exists that satisfies both conditions.

Any other questions? Convergence in MATLAB. Okay, let's do that. So, let's replace it here. Okay? So I'm just putting one everywhere. I don't need this subplot. Let's remove this subplot. Let's run it.

Notice something very different over here. Let's see if the other graph is still open. Is it open still? Yeah.

So, this was the first case, and this is the second case. Notice one big difference. What is the x-axis here in the new graph? Iterations. I mean, it was still iterations earlier. But what is it saying? 1 and 2. That means I started, basically in one step, I reached. And look at the log scale. I started with something that is not in my hand, right? But where am I ending? -35 in log scale. It's basically error is 0. When the eigenvalues are the same, and that came to us from the theorem, which is why we should not be surprised.

And this is also the power of knowing some theory behind what you are doing. Otherwise, you would never know, right? So, you know, you can try to engineer ways by which your condition number is better, and there are a lot of numerical tricks where before you start working with your problem, you try to do some shaping of the condition number to make it better, and this is the reason why. Instead of 50 iterations ending up at log scale -2, I end up in basically one iteration or two iterations at log scale -35.

Can we know the condition number? Yes, very much. So, let us do it, to know the condition number, it is quite simple. I just have to run this once again. Let us comment this and here it comes. So MATLAB simply has this.

Oh sorry, it is A. So for the tutorial problems also that you have, you can just use this MATLAB command to get the condition number.

To get some starting point, yeah. To get the orthogonal eigenvectors. And then I just change the eigenvalues. I mean, there are various ways in which you can do this. If you wanted to, with this code, you could just modify it to plot.

Actually, you cannot visualize in 10 dimensions. Here we took $n = 10$. If you took $n = 2$, you could actually plot a trajectory of x_0, x_1, x_2, x_3 with non-dots, and you will be able to see that 90-degree thing happening over there, zigzag. And as you vary your eigenvalues, you will see how stretched or whatever it is and you can see the zigzag there or it go away, right? So you can use this code as a starting point to really understand many things. And just for fun, instead of making $n = 10$, make $n = 10^6$ and see how much time it takes.

If you are doing gradient descent, yes, I do not know how to prove it. So, for that every time you cross a contour, you should be orthogonal to it in some sense. Then you will keep going along that. Correct. Correct. Yeah, if you are doing exact line search and your starting vector, starting negative gradient was pointing straight at the solution, right? Which is what happened here, when I made all the eigenvalues the same.

Correct. We will have to think about it, like are there some really funny counter cases where it does not happen. It is not very clear immediately, what you are saying. Correct. You can start at the four corners of it, and you will reach in one shot because your trajectory is always orthogonal to the contours. The gradient and contour are always perpendicular, so you just fall straight into the solution.