Inverse Methods in Heat Transfer

Prof. Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology, Madras

Lecture - 29
Tikhonov and Levenberg – Marquardt – Example Code

(Video Starts: 00:19)
Welcome back. We are looking at the code for Tikhonov regularization as well as Levenberg Marquardt regularization or Levenberg Marquardt for Gauss Newton in this video as we had discussed in the class. The basic idea of both Tikhonov as well as Levenberg Marquardt is to modify the basic Gauss Newton algorithm so, as to make it a little bit more stable. Now, the example that I have chosen here.

Because we did this example in Gauss Newton is optimized not for showing the good properties of Tikhonov and Levenberg Marquardt even though I will show a little bit of it. Some of it should be visible but it is optimized to us to enable you to do this by hand I am of course showing this by code because showing this my hand in a video recorded lecture actually looks quite boring. You can do that by hand also.

And cross verify with the code which will be uploaded for people taking this course for credit. We are going to repeat the same unsteady inverse problem that we did the last time. Remember we had a first order system which had some heat generation within it. And of course, it was losing heat via convection and we had collected temperatures at various time points like 10 seconds the temperature is 3.1 or the temperature deficit is 3.1. $(T - T_\infty)$ and so on and so forth.

And we already knew that the forward model had the form $\theta = a(1 - e^{-bt})$ as I had mentioned in the overfitting lectures this week. We actually need a lot of parameters to suit, to require the need for regularization. Either I need to add a lot of noise, I have not chosen to do that. But if we look at this problem the primary purpose of showing you this course is how smaller modification you need to make to the Gauss Newton algorithm in order to get Tikhonov regularization as well as Levenberg Marquardt.

So, we are going to continue the same problem. I will show you what happens when you do not add damping in the form of Tikhonov or Levenberg Marquardt. And then what happens when you add those and how the iterations proceed. So, remember I have just done the same thing as the last time encoded the data or basically store the input data. This is just the table of data which was their t and theta I changed the names of the variables just for convenience I called X as t.

you can see the same as the time variable and Y is the theta variable again just for notational convenience, M of course is the number of data points which in this case is 6. So, last time we had started with an initial guess of 35 and 0.004. I just want to show the same initial guess I will change the code. We will ensure that, we are getting the same results for lambda = 0 and then we will start changing lambda.

The model remains the same, because the forward model has not changed so Y hat still remains 1 time $1 - e^{-bx}$ in this case X is basically just time. Then dydt or Dyda is the same as last name $1 - e^{-bx}$ and Dydb is also the same as first type. Now here is where we are actually going to do the iterations of our Tikhonov. So, we are going to do iterations of Tikhonov. Here again same as before you first find out what the forward model is we can in fact write that after finding the forward prediction.

So, remember we first make the forward prediction, take a guess for A and B which we have done here. Here are the guesses take a guess for A and B for that A and B for every X in your data set find out the Y. So, we did that. So, once you do that you can also find out the gradients so the Jacobians, we calculate same as Gauss Newton which you should remember from last week Dyda or del y hat del a and del y hat del B which we have already stored here.

Then we create this matrix Z, which is what lets us solve the Newton system the derivation was there, last time. I also showed you not really a derivation but a semi derivation for Tikhonov and Levenberg Marquardt this week. So, here is the Newton system Z transpose Z times delta I = Z transpose B where B is the error or the difference between what your prediction is and what the actual value of y is at those points.

So, for example prediction will be theta actual value is whatever is there is known this column that is that matrix D here. For Tikhonov it almost stays the same but you can see that particular you have a small variation you have this additional term lambda I on the left-hand side. What it does? As I told you in the class was it makes the equations are the linear system of equations. A little bit more diagonally heavy which stabilizes the system.

So, it is like adding a small term in the denominator if you have some a by B term instead of that if you do, A by B + lambda that makes it a little bit more stable. So, the code here is simple before I added this lambda term it just looked like dA is LHS by RHS where LHS was Z transpose z. Now I have added this term, I here simply is the identity matrix I will show it to you when I run it. So, this is simply the identity matrix.

So, lambda times I lambda we have given I the older value was 0.01. But I have given a value of 0 now when you run it you will see what happens and then of course RHS the same thing. You are basically doing inverse of our LHS multiply RHS using Gauss elimination. This is the MATLAB notation for that and then your A update your B really nothing much has changed from Gauss Newton as you can see. It is a trivial sort of small improvement over Gauss Newton.

But it helps stabilize things which is why it is very popular, well not really just Tikhonov. Now as I will show you even more Marquardt what is a little bit more popular. So, we can go ahead with the same parameters as before and see if we obtain the same results if we set lambda = 0. And you can see please remember these final values 48.8239 and 0.0070 so if you wish we can just add these final expected values for Gauss Newton are a = 48.8239 and b = 0.0070.

So, these are the ideal values that you get and we got here if you notice 1, 2, 3, 4, 5 about in 5 iterations. So, if you wish we can add the iteration number also. So, that it is a little bit clear I will run it once more. so, you can see at around the 5th iteration, we attain the value 48.8239 and 0.0070. Now what we are going to do is play with a few values of lambda as I said really speak you cannot expect too much of an improvement here.

And in fact, as I told you in the theory class it can damp things when they are already working well. So, you are sort of trying to remove off, too much noise from the convergence process. So, it can actually do a little bit worse. So, let us see we will have to play with it instead of taking lambda = 0 let us say we take lambda = 10 power -4. Or let us say 10 power -7 which is really small and we can run it.

So, you can see that it has still converged in about 5 iterations. So, the convergence process is slightly different but lambda = 10 power -7 which is very small close to 0 still gets you about 5 iterations. If I make lambda very large, let us say I make lambda 1. You can see it has not yet converged it has actually damped the things. It is pulled them down and it is still converging it does not converge. So, somewhere in between, you can try lambda equal to 10 power -2.

For example, for 10 power -2 you have the 6th iteration so we can try something like that power -3. So, here you see maybe once again the 5th type version. Now the important thing here is to see what all these terms are the left-hand side is a 2 by 2 matrix. So, notice as I had told you before Z transpose Z is a square matrix. You should remember this from your Gauss Newton algorithm class also. But this is you can notice LHS is a 2 by 2 matrix.

The identity matrix as you can see is just a 2 cross 2 matrix of size 1. The new LHS with which you solve will be just that plus lambda times I. And the RHS is as it has to be itself to cross 1 matrix so that is what happens here as we go forward. So, we can try 1 more lambda value but more than that, I want to change the initial guesses and let us see what the effects are. So, suppose I change the initial guess and once again just to set a baseline I put a lambda of 0.

Now you can see that when you put a lambda of 0, this does not converge at all. Actually, let me give a little bit gentler as an initial condition. So, when you put a further away initial condition and you try to converge it, gets to around 7 iterations or actually if you come to a 0.829 at about 9 iterations. So, 9 iterations are, what it takes for the new initial condition. So, this is just to remind ourselves that, it is 9 iterations for the new initial condition.

And now we can try with various values of lambda. So, let us say I take lambda as 10 power -4 you take around 8. So, as I said you cannot expect much improvement with this kind of Tikhonov but just to satisfy ourselves, we can try a few other values. So, now this gets in 6 so you see get a slight advantage with Tikhonov with a particular lambda. So, the lambda as I told you in class is called a hyper parameter much like alpha the learning rate in the gradient descent.

These are not parameters we are solving for A and B are the parameters we are solving for. But you get some advantage for slightly differing lambdas. Some smoothing can be expected in this case again if we increase lambda too much it took too much time. I mean it is not converging exactly to that value within 10 iterations it could probably take more. Of course, all these iterations are a little bit painful to do when you do them by hand.

Nonetheless again people doing it for credit I would recommend that they do that. So, somewhere between 6 and 8 iterations is what Tikhonov takes. So, we saw one case with some slight advantage this was lambda = 10 power -2. But otherwise, at least for the original initial conditions there was practically no advantage. You really have to look at a combination of when the number of parameters is large you will actually find that Tikhonov works better than pure Gauss Newton for several cases.

Especially when your initial guesses which typically will be the case will be far off from the actual values. So, what I want to do is now just switch from Tikhonov to Levenberg Marquardt which I will show you in the next code. So, let us see that. So, here you see the Levenberg Marquardt code, again practically identical to the Tikhonov code which we saw earlier. You can see I have practically made no change here except there will be a small change below.

So, but just to walk you through this the same initial model, the same data and the same initial guesses. So, I have you know commented out the new guesses that we made the last time and again I have set lambda = 0. This is generally a good programming practice just to ensure that you are getting the right values at least for the same old cases. And same model, same gradient and same initial thing of calculating the model and then proceeding further.

But the usual Gauss Newton step was as we saw Z transpose Z delta a at equal to Z transpose D and the Tikhonov was that plus a lambda I. And the only change we make in Levenberg Marquardt as I told you in the last was have effectively a variable lambda for each variable. Another way of doing it is lambda times diagonal of Z transpose Z. So, this is the term that you have to be careful about.

So, here notice Z transpose Z, this is z transpose Z plus lambda times this matrix DL which is the diagonal of LHS. I will show that to you shortly as we run through the program and then the rest of it is exactly the same as before. So, programming wise, it is extremely straightforward. In fact, even calculation wise, it is not particularly harder than Tikhonov or original Gauss Newton. It is the Gauss Newton which is a hard part in these problems.

So, what we will do is, let us just start running the program. So, when we come here, please notice what Z is? Z is actually you can see 2 columns are there the 1st column is del y-hat del a. It tells you how much your prediction would change if you change a. And the 2nd column is del y-hat del b tells you how much the prediction would change if you change b? Obviously, our job is to change a and b and figure out what is the optimal sort of place where we can move this y hat too or a and b too.

So, once we have that, we 1st can calculate this Z transpose Z term. So, if we calculate Z transpose Z let me just do that. If you calculate Z transpose Z it comes to some term but then you calculate the diagonal. So, the diagonal is exactly the diagonal terms but I have written it as a full matrix. Just so that you can add these 2 matrices here multiply that by lambda Z transpose Z plus lambda times this diagonal term becomes the new LHS and then we solve the Z transpose D which is again an RHS term and we do that repeatedly. So, let me just run this and once again let me actually put an iteration number also here. So, the same as what we saw last time in about 5 iterations, we converge to 48.82399 and 0.0070. So, of course this is the case where we have not added any lambda at all.

Now we can try adding some lambda to it. Let us add a really small lambda 10 power - 7 and you can see that this still converges within 5 iterations. Levenberg Marquardt is better behaved. That is, it damps lesser than Tikhonov actually as you get closer and closer to the answer you get better and better. There is a way of actually changing lambda scheduling lambda as I told you briefly within our class you can actually change the lambda value as you go through the iterations.

But we are not going to do all such fancy things. We are just going to stick with the basic algorithm just so that you get an idea after this is a methods course rather than a full-scale application to any practical problem. So, this is just primary and intentions to for you to understand methods. so, now in the 5th iteration you see this 48.8239 again we can play with lambda I can make it a little bit larger. So, it just makes it a little bit slower.

So, you have damped more it makes it slower if I make it really large like lambda = 1. It does not converge or at least it takes longer much like Tikhonov. Now we can try the case with the further away lambda or further away initial condition. You will remember that it took around 9 iterations to get there and now we can try it with damping. So, we can try it with the Levenberg Marquardt algorithm where I change the lambda to 1 e to the power – 7 you should put 1 dot n bar - 7.

So, we are still getting 9 iterations here too. If I make it 10 to the power - 4, you see 7 iterations which is a little bit faster than what we had earlier. We had 9 iterations. So, Levenberg Marquardt actually offered us an acceleration over simple Gauss Newton with this initial condition I can try a little bit larger lambda. And here you see again 7 iterations maybe it is converging a little bit faster.

You can also see that the value of B has very quickly converged to 0.0070 which it did not for Tikhonov or Gauss Newton. So, Levenberg Marquardt seems to have certain advantages for this example. Again, I would want you from making a generalization from here. Generally, it is true that Levenberg Marquardt is better than Gauss Newton. And Tikhonov is also better than pure Gauss Newton.

You have to play lesser with lambda and Levenberg Marquardt in order to get good performance. And you will have to play a little bit more with the Tikhonov lambda in order to get good performance. Overall, in general in all practical problems it is best to use Levenberg Marquardt. I will show you later on when we come to the machine learning portions that when we teach when we use Levenberg Marquardt.

In fact, MATLAB uses Levenberg Marquardt by default and of course it has a sophisticated internal scheduling routine that makes convergence really fast on fairly complex problems. So, we will see that later on in the course so that is it for this video we saw a couple of examples of Tikhonov as well as Levenberg Marquardt applied to the standard problem that we used in the last week, the theory you had seen earlier. So, that is it for this video, thank you.
(Video Ends: 20:13)