

Computational Systems Biology
Karthik Raman
Department of Biotechnology
Indian Institute of Technology - Madras

Lecture – 43
Methods for Parameter Estimation

(Refer Slide Time: 00:11)

Computational Systems Biology
Methods for Parameter Estimation

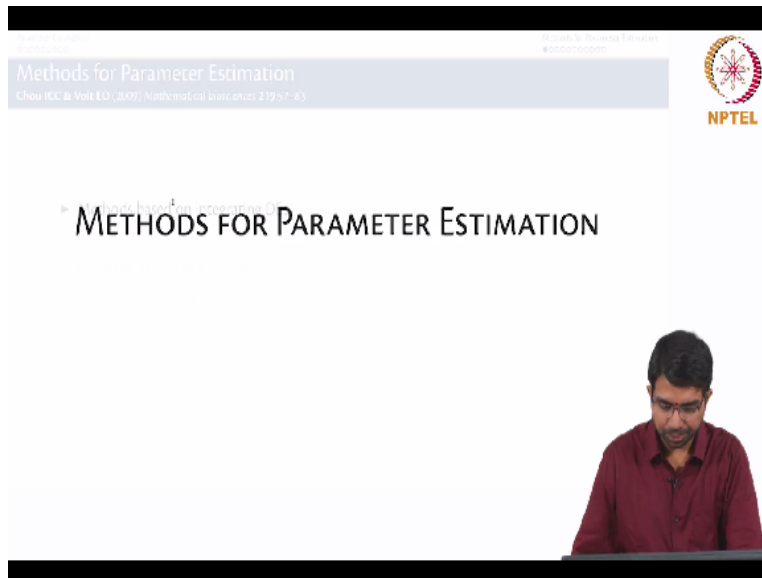
- ▶ Introduction
- ▶ Generic Recipe for Direct Search Algorithms

Karthik Raman
Department of Biotechnology, Bhuvan & Jyoti Aheha, School of Biosciences
Initiative for Biological Systems Engineering (IBSE)
Robert Bosch Centre for Data Science and Artificial Intelligence (RBC DSAI)
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

From this video onwards, we will start a series of videos where we study methods for parameter estimation. So these are essentially methods to tackle high dimensional optimization problem. They could be multi-objective. In most cases, we do not use a multi-objective optimization function but in this lecture, I will introduce you to the methods of parameter estimation and convince you that gradient-based methods and other methods.

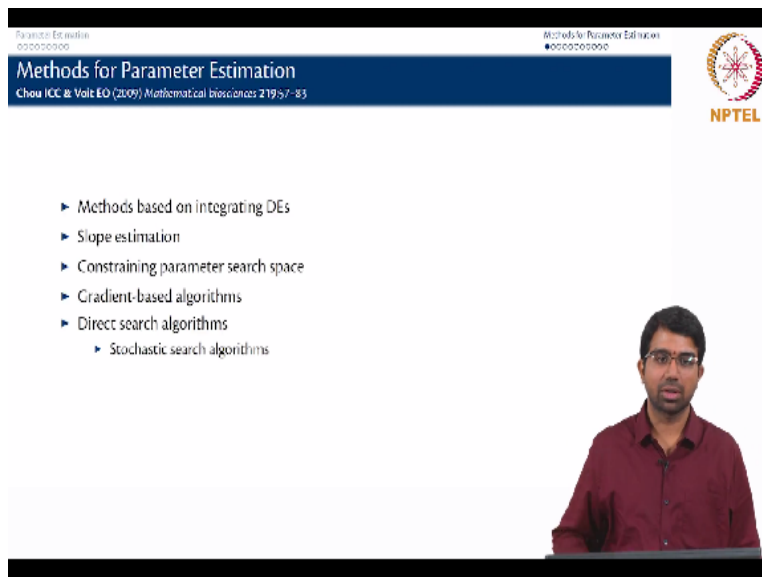
Other standard methods are not entirely useful for parameter estimation for biological networks and move over to direct search algorithms which are the most commonly used family of algorithms for estimating parameters.

(Refer Slide Time: 00:51)



So let us now look at the methods for parameter estimation. So there are many methods of parameter estimation and we are obviously more interested in the non-linear, you know, optimization algorithms, right because we have a quite a non-linear system.

(Refer Slide Time: 01:08)



So there are methods based on integrated differential equations. There are methods that involve slope estimation. You want to typically constrain the parameter search space in your analysis and there are also methods that involve gradient-based, that involve the gradient computations and so on but if you see most of these algorithms are not very applicable to biological datasets, okay. Everything involves integrating of differential equations but you really cannot use.

And we will constrain parameter space but basically gradient-based algorithms are not very easily usable in biological datasets because the, it is very expensive to compute the gradient, right. So at this point just step back and think of what do we mean by computing the gradient? What is it that you are going to compute the gradient for and what does it entail. True. So that is the gradient. What is the direction in which there is a maximum delta in your function? What is the function?

“Professor - student conversation starts” Sir, the Cost function. The cost function. **“Professor - student conversation ends.”**

(Refer Slide Time: 02:20)

So remember the cost function involves, the cost function involves your, the cost function we said was non-linear and the recap, let us call it some error e of θ , the θ is a vector. This is basically x measured, \hat{x} predicted, i of θ ... something of this sort, right. So now how do you compute this prediction?

“Professor - student conversation starts” You have a model, form of the model. What is the form of the model. The differential ($\dot{()}$) (03:19). So it is going to be dx/dt is f of x,t or rather... Is it right? So this involves? θ . θ , a parameter subset. Yes, but, okay. **“Professor - student conversation ends.”** Right but look at the problem here. How do you know, you want x of t . So from this to get x of t , you need to integrate numerically.

“Professor - student conversation starts” (()) (04:04) **“Professor - student conversation ends.”** Which is expensive. So let us look at this. You do not have these time courses, right. What you actually have are some points. **“Professor - student conversation starts”** Sir I know the initial condition. Let us say you know the initial condition as well, right but you need to basically integrate this first of all. (()) (05:05) You know, so you will basically guess a theta and integrate it.

So for any given theta, you perform an integration and then you compute, well, you will compute the curve point, the prediction. Yes? So if theta is constant, if parameters are constant, parameters, why do you say they are constant for some value. Okay. In that case, you will integrate once. Yes. Having theta same, after integrating, I will just substitute theta. You do not have to take it again and again.

You will just integrate with adding theta some value. No because you are doing (()) (05:52) you have to actually do that because, it is a good point, which is like you know what you corrected me here is quite important, right. I just said f of x, t is actually f of x, t, θ . When you write the ODEs, you will be passing, it is not that you will have a function that has the integrated symbolic version of the ODEs where you can just plug in the thetas.

So you have to basically stick in this theta into the ODE function and then pass it to the integrator, like ODE 23S or ODE 15S or something like that. So that does not happen. So that is, that is what actually makes it more expensive. One of the things that make it more expensive. Do you understand? Not convinced. So, maybe we should look at the examples, right. That is where the, but if you go back to what we discussed in the previous class, so we said there will be an, and, and you will basically inside of this function, you will run ODE23S of at and you know, x, d initial value and all that.

So in fact, theta will also become an additional input here. But theta is just a value right. It is not going to judge the time or it is not going to change with any... Yes, but it is what the estimator keeps changing. Okay. So the parameter estimator will, first take a guess of theta 0, it will

compute e , right. It will check how good e is. Then it will go to θ_1 , hopefully by gradient but let us see.

We were trying to say how difficult gradient is because you, you basically have to do $\Delta e / \Delta \theta$. You have to explore all possible... It is very difficult. So, you can explore it but you can imagine how expansive this is, right. So computing the cost function is expansive. It involves how many integrations? (()) (08:26) Many, yes. **“Professor - student conversation ends.”**

It depends upon the number of states and so on, right. So it becomes very painful. So if you have 2-3 curves, you have to make 2-3 integrations in the first place and it also depends upon the kind of data that you have, right. Your data material is something like, let me give you a different example. But you are familiar with this kind of dataset. We will be looking at it since the previous class.

One difference is, if I say I did something here. Let us say, I added a drug. So then I have to stop the integration until this point, change an initial condition and integrate again. So I have to essentially do multiple integrations to basically compute, I want to compute the predicted value at this point, predicted value at this point, predicted value at this point and this point. For just computing 4 points, I might need 2 integrations and extend this to the number of species I have and so on.

So it can rapidly become very, very expansive, right. So this is a practical side of things that you need to understand. So it is a difficult proposition. Which is why you do not want to ever compute the gradient. If you cannot compute the gradient, the next best option you have is, guess work. Random search or better search, right and this is what is captured under direct search algorithms or, you know, many of these are also stochastic search algorithms.

They are stochastic in that you do not search the same bay each time. The model in which you explore the parameter space varies in each iteration. **“Professor - student conversation starts”** Given the search algorithms, you are going to (()) (10:39) Yes, yes. **“Professor - student conversation ends.”** So every cost function evaluation requires many integrations but so for

gradient, you require several, at each point you may have to compute the gradient that will add to the cost, right. So that is why you do not go in for gradient. Was that your question?

(Refer Slide Time: 00:00)

The slide is titled "Methods for Parameter Estimation" and features the NPTEL logo in the top right corner. The main content consists of a bulleted list:

- ▶ Most real-life cost functions are non-linear and non-differentiable
- ▶ Therefore, methods cannot compute/use gradient information
- ▶ Such methods broadly categorised as *Direct Search Methods*
- ▶ Central aspect of direct search methods
 - ▶ Strategy to vary parameter vector
 - ▶ Strategy to accept/reject a new parameter vector

A small inset image shows a man in a maroon shirt looking at a screen.

So most real life cost functions like the ones we will use in biology are non-linear and non-differentiable, right. Well you can numerically differentiate them but, so you cannot usually compute or use gradient information. So all the methods that do not use gradient information are broadly grouped under direct search methods and central aspect of all direct search methods is the same.

You have a strategy to vary the parameters and a strategy to accept or reject a new parameter. The logic is very simple. You start with a θ_0 , you compute a cost function. You then either use this cost function or whatever to compute another θ_1 and then is this good or not, right? So usually how it works is, you start with θ_0 , go to θ_1 . From θ_1 , you might either go back here or go to θ_2 . From θ_2 , you might either go back here or go to θ_3 .

“Professor - student conversation starts” But does this take most if then compared to gradient, gradient-based algorithms. You still have so many function computations to perform, right. Gradient at each step involves much higher computation. (()) (12:23) you will go in the opposite direction (()) (12:29) to keep evaluating (()) (12:33) but you, if you are taking a wrong state, again you will come back to previous step, again you will evaluate.

Reaching the point (()) (12:43). So in practice, I think, these work quite well because the gradient computations can, can be quite expensive, right. So which, which direction do you compute gradient in? Sir, we will just compute the gradient of the function (()) (13:03). We will take the direction opposite to the gradient of function. No, no. But in, along which, so you have, you are in n-dimensional parameter space, right.

How will you compute the gradient in that space? (()) (13:18) but the convergence state will be very high. It will not be that much converging. For a non-linear system, we cannot have gradient, we cannot rely on gradient. How will you compute gradient? So you will compute x of some. I will say θ parameter vector. y of x , y of $\theta + \Delta \theta$, y of $\theta / \Delta \theta$, right. What is $\Delta \theta$?

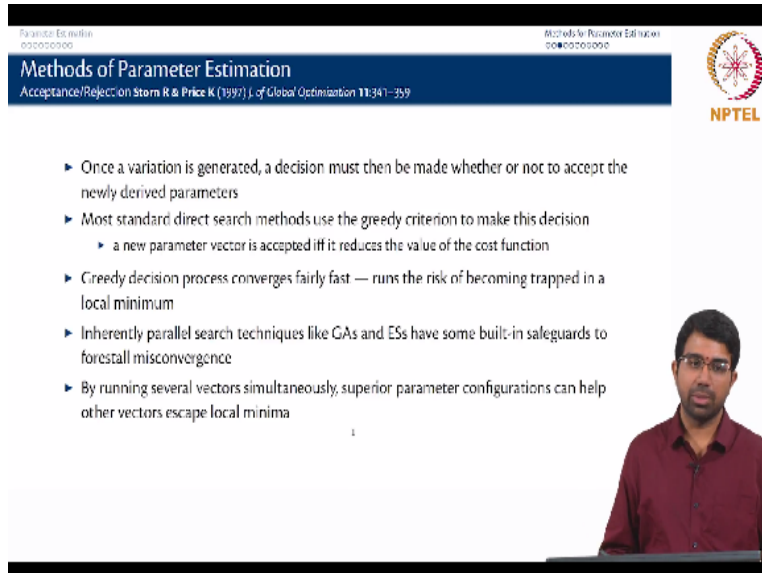
Which direction? (()) (13:56) In all directions. Or you have to go in every possible direction, that is why the problem comes. (()) (14:03) that means you take the maximum and then how much to go? Which direction? What do you mean, just to grad? It will give you 1 direction and then you go in the opposite direction. No matter how many directions you have. But you have to numerically compute grad, right.

See in a, this is how you numerically compute grad. So which means you have to figure out which, what is this $\Delta \theta$. That gives you the maximum. (()) (14:27) may not guarantee optimality, that is what I am saying. If you, we use... Anyway, I think that is your optimality. Even directions also does not guarantee you optimality. But Gauss-New, if we use Gauss-Newton project, as you, system has quadratic locally, that will go towards single step to the center of that ellipse but in, in this method, (()) (14:57) you do not have that.

You cannot have that, because it will assume the, it will (()) (15:05) See how do you numerically compute the gradient? (()) (15:08) no, no, numerically computing, we can, oh yes. (()) (15:19) That is how it fails. Is not it? (()) (15:31) It is unlikely to get hold of the local (()) (15:45) but in direction search, you have no such restriction, right. So that is only (()) (15:56) or something. Let us get to that. Let us get to that in a moment. **“Professor - student conversation ends.”**

So how do these methods work? So this central aspect we said was, there is a strategy to vary parameter vector and a strategy to accept or reject a new parameter vector.

(Refer Slide Time: 16:11)



The slide is titled "Methods of Parameter Estimation" and includes the following text:

Parameter Estimation
○○○○○○○○○○

Methods for Parameter Estimation
○○○○○○○○○○

Methods of Parameter Estimation
Acceptance/Rejection Stern R & Price K (1997) J. of Global Optimization 11:341-359

NPTEL

- ▶ Once a variation is generated, a decision must then be made whether or not to accept the newly derived parameters
- ▶ Most standard direct search methods use the greedy criterion to make this decision
 - ▶ a new parameter vector is accepted iff it reduces the value of the cost function
- ▶ Greedy decision process converges fairly fast — runs the risk of becoming trapped in a local minimum
- ▶ Inherently parallel search techniques like GAs and ESs have some built-in safeguards to forestall misconvergence
- ▶ By running several vectors simultaneously, superior parameter configurations can help other vectors escape local minima

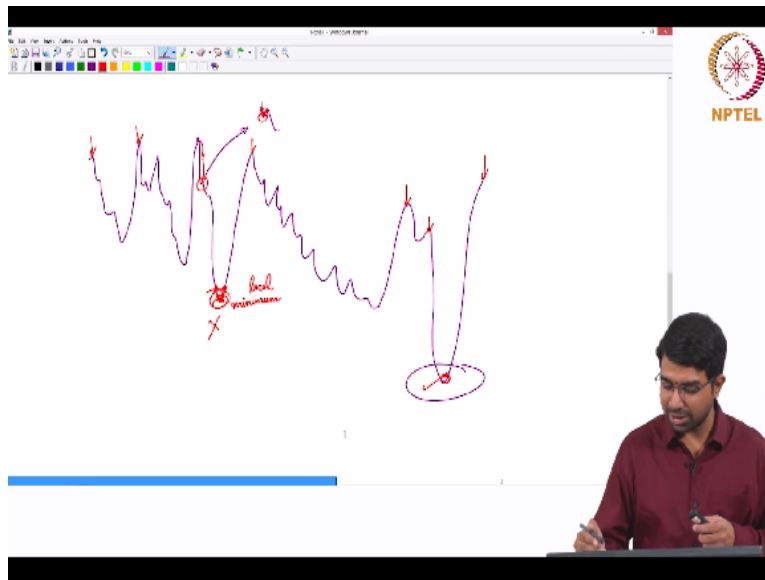
1

The slide also features a video feed of a presenter in a maroon shirt in the bottom right corner.

Once a variation is generated, how do you make the decision? You need to make a decision or should I go in this direction or should I go in the other direction, right. How do I come up with that decision? One way is always accepting a direction where I have reduced cost, right. That is what we normally call hill climbing or actually hill descending but is usually called hill climbing.

So always go in the direction which takes you towards the maximum. So that would be hill climbing, okay. So new parameter is accepted if and only if it reduces, improves the value of the cost function. The Greedy decision process converges fairly fast but has the obvious risk of getting stuck in a local minimum or maximum.

(Refer Slide Time: 17:05)



So let us look at a, slightly, right. This function has a clear local global minimum, right. This is the global minimum. But the problem if you start with a hill climber, let us say you start here, hill climber or descender, let us use the terms interchangeably. You will go here; you will go here. Even here, say I were to zoom in, if you are at this point. You move in this direction, cost value increases.

You move in this direction, cost value increases. So this is the local minimum and you are very likely to get stuck here or you can come and get stuck here. You go in this direction, cost increases. You go in this direction, cost increases. So you are at a local minimum but what you want to get here? How do you get here? So one thing to do is, you have to somehow kick this out of local minima, right.

So one obvious option is you can start exploring at different points. May be you start here, you immediately land here, right. But the problem is, this is 1-dimension. This is 1-dimension, remember. I am just showing you the value of some x as θ changes. So the problem is in higher dimensions, it can get really difficult. You will have many more; between every minima, you will have a few maxima.

Between every few maxima, you will have a few minima. It just really crazy landscapes can exist in high dimensions. How do you handle it? It becomes very hard. And there are many techniques

which are inherently parallel in searching, right. Like genetic algorithms and evolutionary strategies. We will try to talk about them and they have some built in safeguards to avoid misconvergence.

This is what we would call a misconvergence. You go to a wrong minimum. You wanted this but you got here, right. So how do you avoid these. So one way we suggested was, you do a lots of parallel search but many of the algorithms have this parallel search built in as we will see. And we are running several vectors simultaneously, you can get superior estimations and they can help escape the local minima.

(Refer Slide Time: 20:12)

The slide is titled "Methods for Parameter Estimation" and is part of a course on "Desirable Characteristics Stem & Price K (1997) of Global Optimization 11:541-559". It features the NPTEL logo in the top right corner. The main content is a list of five desirable characteristics for parameter estimation methods:

- ▶ Ability to handle non-differentiable, non-linear and multi-modal cost functions
- ▶ Parallelisability to cope with computation intensive cost functions
- ▶ Ease of use, i.e. few control variables to steer the minimization
- ▶ These variables should also be robust and easy to choose
- ▶ Good convergence properties, i.e. consistent convergence to the global minimum in consecutive independent trials

An NPTEL logo is visible in the bottom right corner of the slide. A small number '1' is centered at the bottom of the slide. A video inset shows a man in a maroon shirt speaking.

So what are the desirable characteristics, right. You want the ability to handle non-differential, non-linear, multi-modal cost functions, right. Cost functions essentially very badly behave as you would normally observe in biology because the systems are highly non-linear. It should be parallelisable because we are going to estimating the cost function too many times. So you want the function to be parallelisable in some way.

The algorithm is of use meaning you need few control variables to steer the minimization. What are these? These are what one would call as hyperparameters. They are parameters of your estimation algorithm. So how do you, your estimation algorithm itself will have certain parameters. For example, in a hill climbing algorithm, your parameter will be how many random

searches should I do or how many initial cases from which I should start, right.

Although that is not the best example, you will see better examples for hyperparameters when we look at more complex algorithms and these variable should be easy to chose. You do not want to spend a lot of time tuning your algorithm. You have to tune the algorithm by picking the right set of hyperparameters, right and you it should have good convergence properties and ideally consistent convergence to global maximum or global minimum, across runs, right.

So only if I started a particular, it is not that happens even for hill climbing, right. If I start at this X_0 , hill climbing will happily converge to this minimum whereas if I start at this X_0 , it will just converge here, right. So it is not reproducible. But obviously this is our wish list. So it is not necessary that there is 1 algorithm that will satisfy everything in this wish list.

So again, depending upon the scenario, depending upon the problem at hand, you may want to prioritize 1 or more of these features compared to the others. So what are the features you would like to prioritize.

(Refer Slide Time: 22:16)

The slide content is as follows:

- ▶ Hooke-Jeeves (Pattern Search)
- ▶ Nelder-Mead Simplex / Downhill Simplex Method (`fminsearch`)

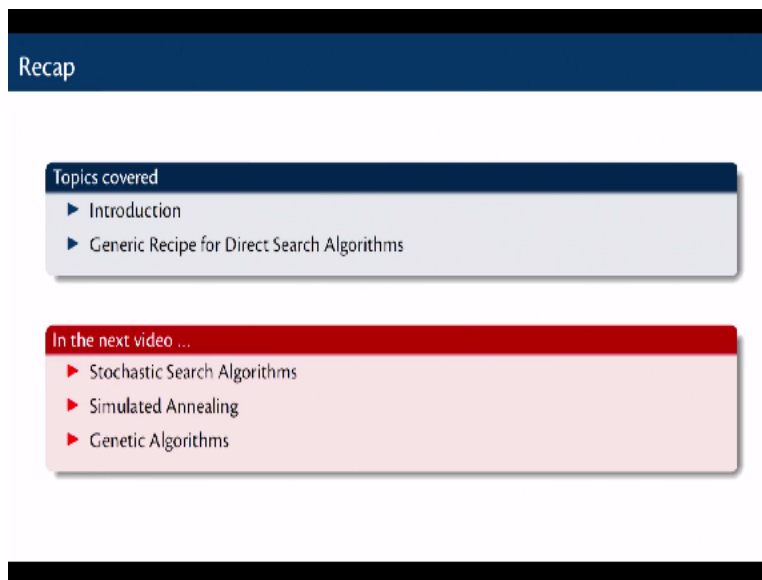
The slide also features the NPTEL logo and the text 'Classic Methods of Direct Search'.

So there are some classic methods of direct search which involves Hooke-Jeeves pattern search and your Matlab `fminsearch` which is called Nelder-Mead simplex. So this, this is direct search, so much as it does not use gradient information. It is not stochastic. So what it does is, it tries to

explore a particular direction. If the direction looks good, it will go faster in the direction. If the direction is bad, it will reflect to an opposite direction and so on.

So you should really try to look at the documentation of `fminsearch` and you can run `fminsearch` with something like a `verbo` mode. It will tell you I did a contraction, I did a reflection, I did an expansion, those kinds of things. So you will know exactly how the algorithm is proceeding and where is it getting stuck in the, in the parameter space.

(Refer Slide Time: 23:04)



The slide content is as follows:

- Recap
- Topics covered
 - ▶ Introduction
 - ▶ Generic Recipe for Direct Search Algorithms
- In the next video ...
 - ▶ Stochastic Search Algorithms
 - ▶ Simulated Annealing
 - ▶ Genetic Algorithms

So in this lecture, you had an, you had an introduction to the basic parameter estimation algorithms and I also gave you the generic recipe for direct search algorithms, right. What are all the classic, so you want to have an initial X_0 and how you go about finding the next point and whether or not you reject it and how do you terminate the optimization. In the next video, we will start looking at some important direct-search algorithms or stochastic search algorithms such as simulated annealing and genetic algorithms.