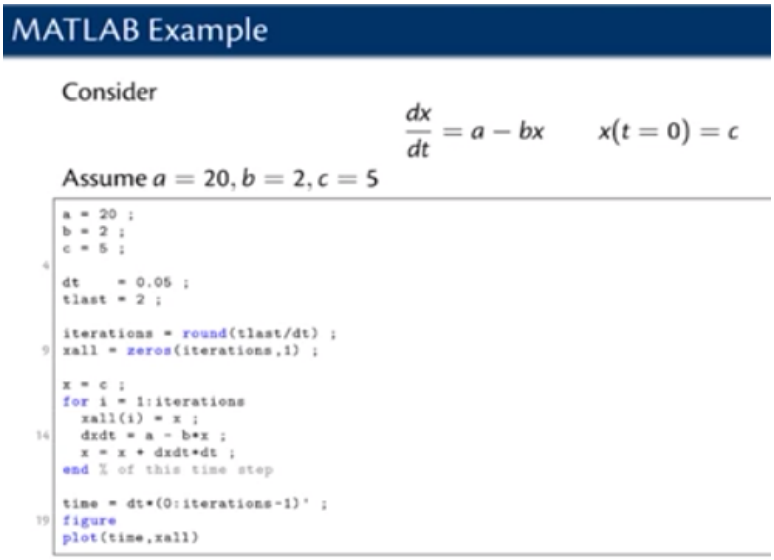
Computational Systems Biology
Karthik Raman
Department of Biotechnology
Indian Institute of Technology – Madras

Lecture - 38
Lab: Solving ODEs in MATLAB

In this lab video, we will look at some simple iterative code to solve ODEs and we will also see how we can use MATLAB ODE solvers, which are even compatible with stiff ODE systems to solve the systems of equations we typically encounter in biochemical networks. Welcome back. Let us look at a MATLAB example, how do we solve ODEs using MATLAB.

(Refer Slide Time: 00:34)



We will start with the simple ODE system something that is very familiar dx/dt is a-bx. I am sure most of you can actually integrate this analytically, but how would you implement this in MATLAB. This is how you would implement it. This is quite straight forward. What do you do? You first set up a vector for all the values of x and the most important part is somewhere here, where you basically compute what is dx/dt, dx/dt is a-bx.

That is your function and x=x+f(x)*delta t. That is what we had here.

(Refer Slide Time: 01:28)

Euler's method

$$\frac{dx}{dt} = f(x) \qquad x(t=0) = x_0$$

$$\frac{dx}{dt} \approx \frac{x(t+\Delta t) - x(t)}{\Delta t} = f(x)$$

$$x(t+\Delta t) = x(t) + f(x) \cdot \Delta t$$

Since we know x_0,

$$x(\Delta t) = x_0 + f(x_0) \cdot \Delta t$$

Since we now know $x(\Delta t)$,

$$\begin{aligned} x(2\Delta t) &= x(\Delta t) + f(x(\Delta t)) \cdot \Delta t \\ x(3\Delta t) &= x(2\Delta t) + f(x(2\Delta t)) \cdot \Delta t \\ &\dots \end{aligned}$$

Leonhard Euler
1707–1783

Basically x, if you were to look at it from a computational coding stand point, you would say this is x is x+f(x)*the current value of x * delta t. that is exactly what we are doing here. x=x+f(x), which is nothing but dx/dt*delta t. That is it. So all the action is happening in line 15 of this code. So let us open this up.

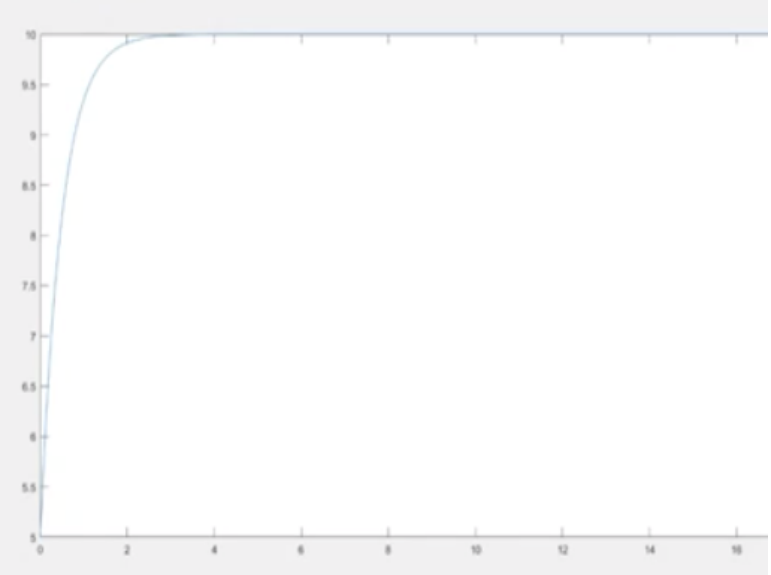
(Refer Slide Time: 02:11)

```
 7
 8 -   if(~exist('dt','var'))
 9 -       dt=0.02;
10 -   end
11 -   tlast = 20 ; % s
12
13 -   iterations = round(tlast/dt) ;
14 -   xall = zeros(iterations,1) ;
15
16 -   x = c ;
17 -   for i = 1:iterations
18 -     xall(i) = x ;
19 -     dxdt = a - b*x ;
20 -     x = x + dxdt*dt ;
21 -   end % of this time step
22
23 -   time = dt*(0:iterations-1)' ;
24 -   plot(time,xall)
25 -   hold on
26
```

I am basically setting a default value of delta t to 0.02 and I am simulating it for 20 units of time. How do you know the units here? If it is seconds or minutes or hours or days? It will depend upon the units of A, B, and C. It is currently arbitrary units, but you can assume it to be seconds or whatever. Is this program clear to you? You are saying, all this is just setting up the code and t last is important. What is the final time till which I want to simulate?

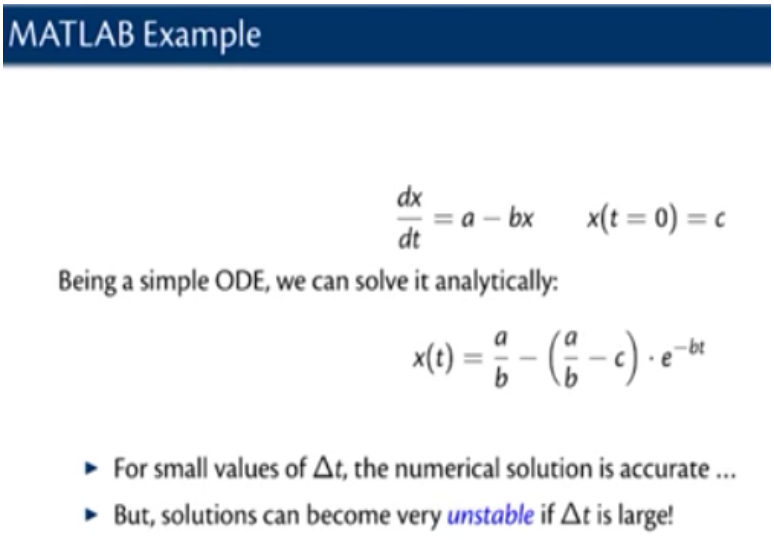
This is till where I want to plot the graph and what is this? Initial condition, x=c and then I basically say x all is x and I am creating an array with all the values of x and dx/dt is a-bx, x=x+f(x)*delta t and end of the time step and I am then plotting the value. Let us see how it looks like.

(Refer Slide Time: 04:28)



Does this look right? What would happen if you analytically solve for it? This is what you get.

(Refer Slide Time: 04:40)



You get an exponential. Some alpha – beta *e to the -bt. So it saturates here. what value does it saturate to, a/b at t=infinity, this term will vanish leaving behind a/b and your a was 20, v is 2a/b,

therefore n. So x basically goes to a/b. So for small values of delta t, the numerical solution is accurate and what is small, we do not know. We have to see what happens, but then for the solutions can become very unstable if delta t is large.

What do I mean by unstable? We look at stability again in the next couple of classes and what do I mean by unstable here. Basically behaves weird. So if you change the delta t, one thing you can imagine is that lower the delta t, better the accuracy. Let us actually do this, dt=dt/10, we make the delta t even smaller. Let us run the code again. You can hardly see the difference. Can you see the difference? I hope you can see there are two curves.

If you really is human, you can see that there are two curves that do not actually overlap. For practical purposes, they do overlap. So let us say dt=, no big deal. This is what we would call stable. The delta t has changed, but your output looks decent. As long as it is less than 1, it will be more or less you can extrapolate it. Well, I am not so sure, gone. But here actually the behavior itself has changed, or so it seems, because we are extrapolating.

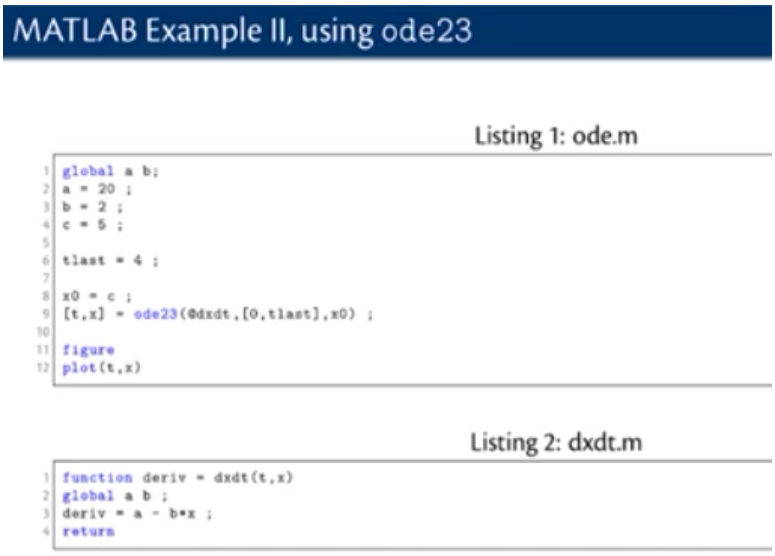
So it has now become discontinuous and if I make, this is what we mean by unstable. It just misbehaves. For a change in delta t, the output is unrecognizable. You are getting busily random stuff. It is almost like gives you some sort of an oscillation. Let us try, maybe 1.5 something, yeah unstable. How do you pick this value of delta t? It is a little tricky. For a system as simple as this, you do not even need to. You can actually analytically integrate it.

You do not have to worry about it, but you want to do it numerical integration for systems, where it is quite difficult to do an analytical integration. From there, you need to use, there are various ways to pick delta t. You have to kind of find out how the system changes and then try to slowly increment delta t and if you are able to get a sort of linear behavior over that domain, it is okay. So this is the curve rate. So in 0-about 1.5, the curve is behaving reasonably well.

But here there is a rapid, it saturates and so on. So you can split into a few linear bits and you might be okay. If you take a dt of 0.3, it is still reasonable. It more or less mimics the original response, maybe I can superimpose. It is not too far of. But you do not want to be worrying about

finding delta t or giving out a value for delta t. So what would you do. There is a simpler way to do it. We just use a built-in ODE solver from MATLAB.

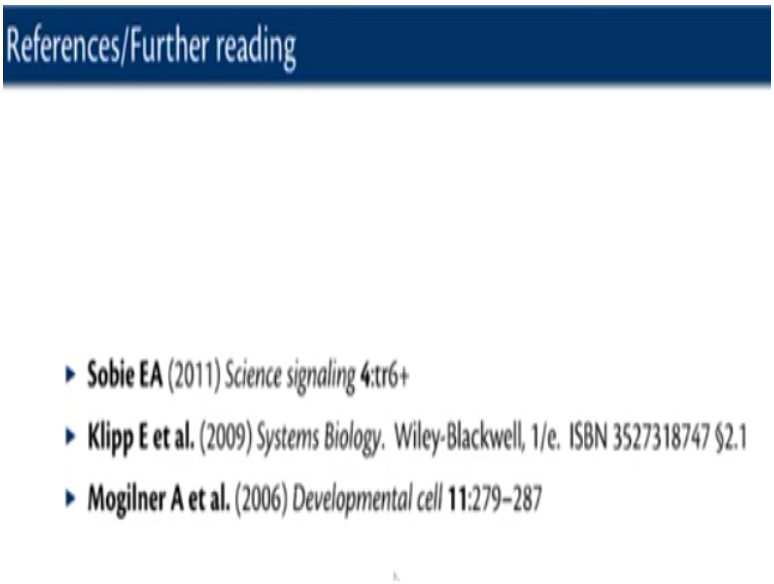
(Refer Slide Time: 10:33)



So here if you see the code has become even more simple. All these lines have basically been replaced by just t, x is ODE 23 of dx/dt, 0 t last, x not. That is it. I still have to give a t last, it is 4 here and actually I have to give a function. What is this? This is your f(x) and what is this act symbol out here. It is a pointer to a function. So this is how you give a function as input in MATLAB. As an ODE 23 function itself requires a function as input, your f(x) basically.

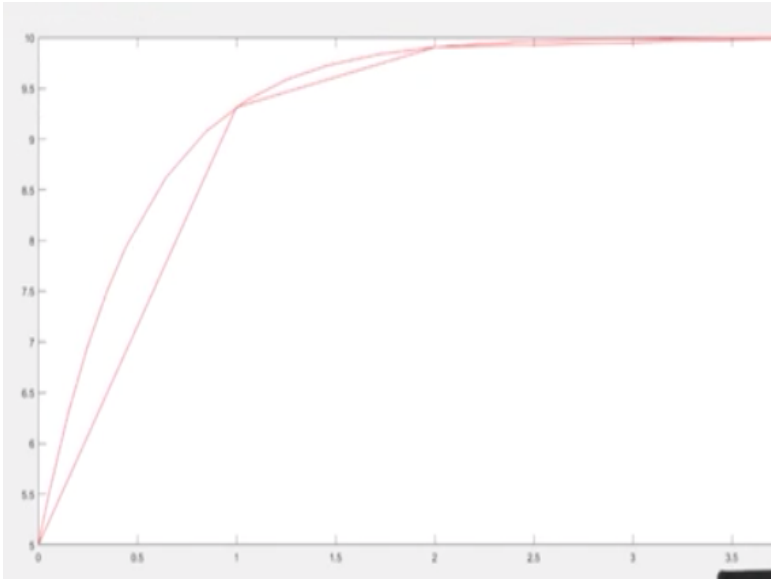
This is a very good reference for you to look up. So do look at this, of course there is book by Eda Klipp and this paper by Mogilner.

(Refer Slide Time: 11:49)



I will share all those materials with you. I think I have already uploaded it. So let us understand what is going on here. This was your original code. This has come down to this and all the action is happening in this line. If you see there are ODE 15s, there are many solvers that you can use. There are a few interesting things here. So let us check them out. Let us just another piece of code, so it is called ODE.

(Refer Slide Time: 13:37)



Is there a problem here? You have to give more t values, that seems to be the problem. So let us see. Actually those values are all correct. It is just that we had joined them through these lines, which was incorrect. These values were correct. So this is something that you need to always

worry about when you are modeling an ODE system. Let us actually zoom and so that you can see it better. So your solver will compute at different points.

You have to tell the solver at which points to compute. If you see the code, these are the arguments that were passed. What were the arguments? First is the pointer to the function that we want, your f(x) function. The last is the initial value and then, these are the points. What does this mean? MATLAB actually handles this quite intelligently in some sense or problematically in another sense, because it assumes a few things.

If you give 2 numbers, it is assumed as 0 and the infinity. If you give more than 2 numbers, it will give you exactly at those points, like we just had. So let me actually unsuppress the output here and run it. What do you expect here for t, x, t should have how many elements 0, 1, 2, 4 and x again 4. So let us run it.

(Refer Slide Time: 15:20)

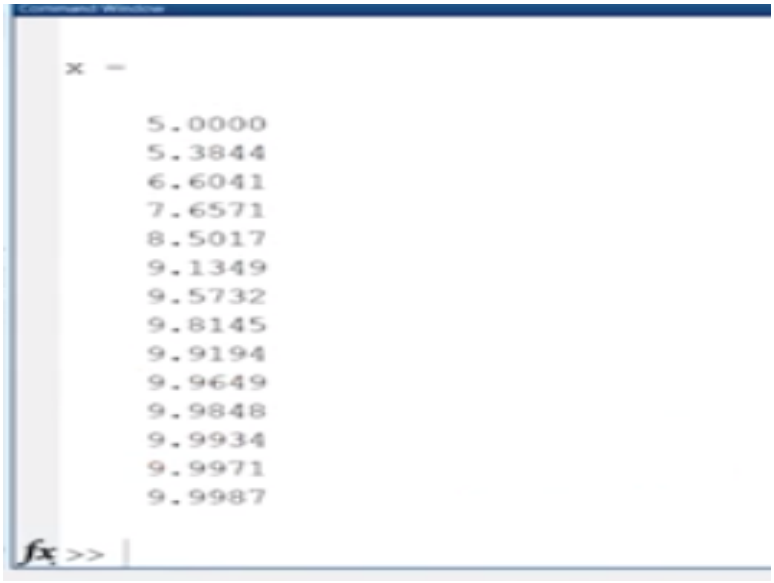
```
1-  global a b;
2-  a = 20 ;
3-  b = 2 ;
4-  c = 5 ;
5
6-  tlast = 4 ;
7
8-  x0 = c ;
9-  [t,x] = ode15s(@dxdt,[0 1 2 tlast],x0)
10
11  %figure
12- plot(t,x,'r')
13- hold on
```

Fair enough, c is 5, what do you expect now. We would expect 2 values. Instead you will get how many values? 22, what is this magic number 22? ODE15s chose to take these many steps and let us actually look at the step size. Initial step size was 0.0179, it kept that for a while, then changed it to 0.0971, so this was the changing step size. So t value is 0.0179 delta t values. This is the value of delta t.

So delta t value was 0.0179 for the first 3 steps and 0.0971 increased, 0.2038 increased further, 0.2574 increased further, 0.4 increased further and finally it stopped at a particular point. It computed at the last point that you wanted to. You wanted this t last. This is the difference 4.0. It ended up computing 3.94 and then you wanted 4.0, so it just took that value. Let us actually do this experiment. Now let us make it ODE 15, there is no ODE 15.

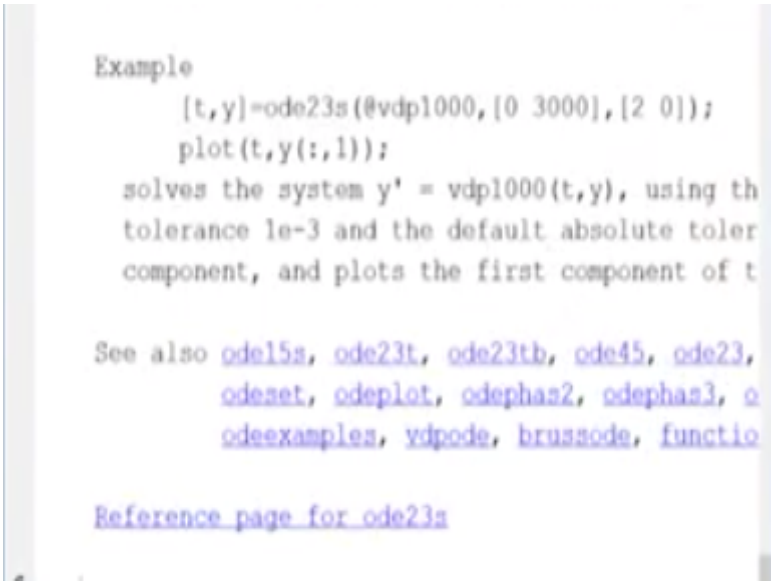
This is ODE 23 is there, let us look at ODE 23.

(Refer Slide Time: 17:27)



Yeah, see this gives you practically indistinguishable quality of output, but only 14 points were used. Let us look at 23s now. It has used 2 more points, that is it 16. The difference between 23 and 23s is 23s is helpful s function is the suffix of x, s are helpful for stiff systems. What are the stiff systems? Stiff systems are systems that are essentially difficult to simulate, difficult to solve through ODEs, usually because they have multiple time scales.

(Refer Slide Time: 18:31)



You see there is ODE 15s, there is 23t, b, 45, 23, 113, 15i. Essentially you have so many different functions. They all work in the same way. It takes something like t, x or t, y as ODE function at function points initial condition. We will have it in multiple dimensions as well. So this is the standard, t out, y out is ODE function of ODE function t span and y not. In this lab, we had a brief overview of a simple iterative code to solve ODEs and also MATLAB ODEs solvers.

In the next video, we will continue with another lab and look at a paper which describes sustained oscillations and glycolysis and we will see how to model a slightly more complex system using MATLAB.