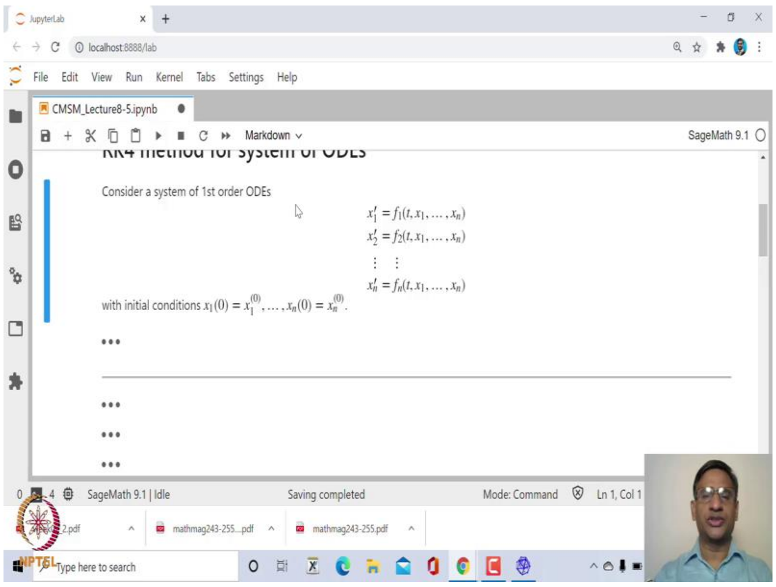
Computational Mathematics with SageMath
Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

Lecture - 53
RK4 Method for System of ODE and Applications

(Refer Slide Time: 00:15)



RK4 method for system of ODEs

Consider a system of 1st order ODEs

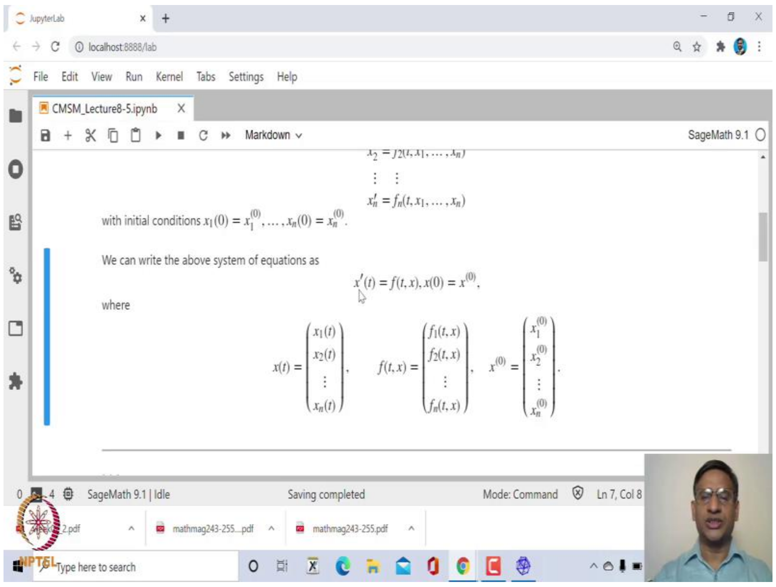
$$\begin{aligned} x_1' &= f_1(t, x_1, \ldots, x_n) \\ x_2' &= f_2(t, x_1, \ldots, x_n) \\ &\vdots \\ x_n' &= f_n(t, x_1, \ldots, x_n) \end{aligned}$$

with initial conditions $x_1(0) = x_1^{(0)}, \ldots, x_n(0) = x_n^{(0)}$.

Welcome to the 53rd lecture on Computational Mathematics with SageMath. In this lecture, we will look at RK4 method for solving a system of first order Ordinary Differential Equations. Let us look at what we want to do. We want to solve a system of first order ODEs of this form.

So, what are these equations? x1 dash is equal to f1(t, x1, x2, …,xn); x2 dash is f2(t, x1, x2, …,xn); xn dash is fn(t, x1, x2, …,xn). This f1, f2, fn need not be linear functions, and with the initial condition x1 at 0 is x1 0 and so on.

(Refer Slide Time: 01:10)



$$x_2' = f_2(t, x_1, \ldots, x_n)$$
$$\vdots \quad \vdots$$
$$x_n' = f_n(t, x_1, \ldots, x_n)$$

with initial conditions $x_1(0) = x_1^{(0)}, \ldots, x_n(0) = x_n^{(0)}$.

We can write the above system of equations as

$$x'(t) = f(t, x), x(0) = x^{(0)},$$

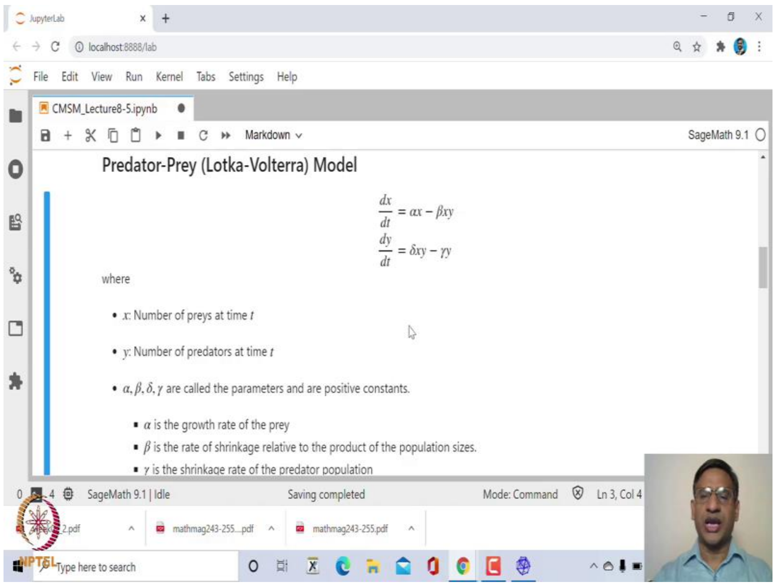
where

$$x(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{pmatrix}, \quad f(t, x) = \begin{pmatrix} f_1(t, x) \\ f_2(t, x) \\ \vdots \\ f_n(t, x) \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{pmatrix}.$$

This set of equations, you can write as a single equation, that is x dash t is equal to f(t, x), and x at 0 is x0, where x(t) is a vector (x1(t), x2(t),…, xn(t)).

Similarly f(t, x) will also be a vector, whose first coordinate is f1(t, x), the second coordinate is f2(t, x) and so on. This is the initial condition. So, this can be thought of as vector equation.

(Refer Slide Time: 01:47)



Predator-Prey (Lotka-Volterra) Model

$$\frac{dx}{dt} = \alpha x - \beta xy$$
$$\frac{dy}{dt} = \delta xy - \gamma y$$

where

- x: Number of preys at time t
- y: Number of predators at time t
- $\alpha, \beta, \delta, \gamma$ are called the parameters and are positive constants.
  - α is the growth rate of the prey
  - β is the rate of shrinkage relative to the product of the population sizes.
  - γ is the shrinkage rate of the predator population

Now, let us look at an example, where we come across such a system. One of the example that we will be looking at is predator-prey, which is known as Lotka-Volterra model. This is very popular dynamical system model.

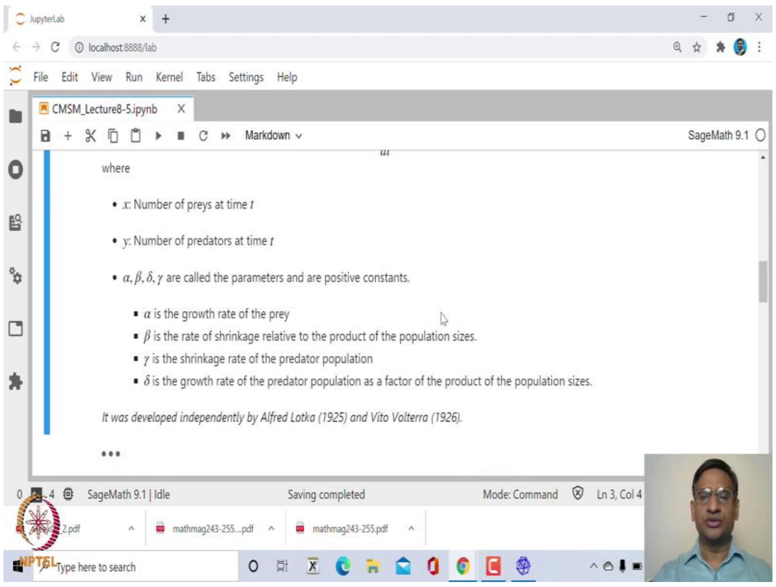
So, let us see what is it? It is represented by dx by dt as alpha x minus beta xy, dy by dt is equal to delta xy minus gamma y, where x is number of preys at time t and y is number of predators at time t.

For example, you can think of preys as, let us say rabbits and predators as foxes. So, foxes and rabbits, This particular system represents growth of these two species together namely rabbits and foxes. Here alpha, beta, gamma, delta are parameters and they are positive constants.

You can see here for example, here alpha is growth rate of prey and in absence of this term, for example, dx by dt will be alpha x, that is same as saying preys grows exponentially in the absence of predators. Assuming that there are enough grass other things etcetera available. So this growth is exponential.

However, in presence of this predator, this is the shrinking, rate of shrinkage relative to the product of the population size.

(Refer Slide Time: 03:31)



where

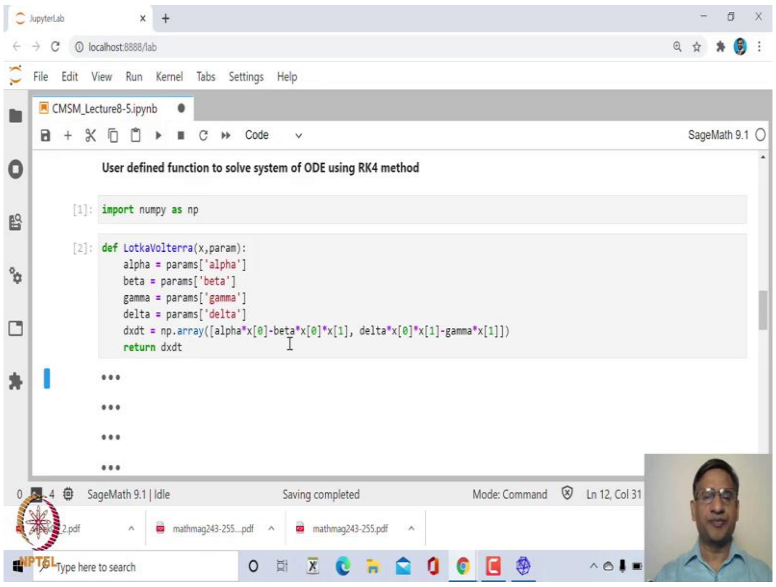
- x: Number of preys at time t
- y: Number of predators at time t
- $\alpha, \beta, \delta, \gamma$ are called the parameters and are positive constants.
  - α is the growth rate of the prey
  - β is the rate of shrinkage relative to the product of the population sizes.
  - γ is the shrinkage rate of the predator population
  - δ is the growth rate of the predator population as a factor of the product of the population sizes.

It was developed independently by Alfred Lotka (1925) and Vito Volterra (1926).

Similarly, dy by dt is delta xy plus gamma y. This gamma is known as the shrinkage rate of the predator population in absence preys. This population will come down. And in case this preys population is present, then this will be the rate at which it grows, right.

There are two particular equations in two variables x and y and you can see here this is a non-linear differential equations. We want to solve this system and using RK4 method. This particular model was developed actually independently by two mathematicians Alfred Lotka and Vito Volterra; Lotka in 1925 and Volterra in 1926.

(Refer Slide Time: 04:39)



User defined function to solve system of ODE using RK4 method

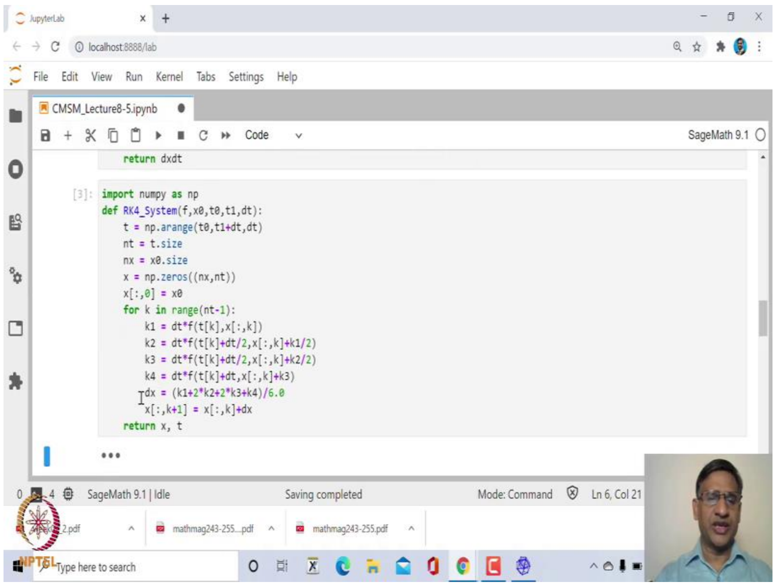
```
[1]: import numpy as np
```

```
[2]: def LotkaVolterra(x,param):
         alpha = params['alpha']
         beta = params['beta']
         gamma = params['gamma']
         delta = params['delta']
         dxdt = np.array([alpha*x[0]-beta*x[0]*x[1], delta*x[0]*x[1]-gamma*x[1]])
         return dxdt
```

Now, let us look at how we will make user defined function for this particular problem. First let us import numpy as np, we will write user defined function basically as a python program. Let us first define this Volterra function, which is going to be a array representing the right hand side of this particular system of differential equations.

Let us give input as x and that x is the variables (x1, x2) and parameters will be as a dictionary form, where this is alpha, beta, gamma, delta. Alpha is equal to params, parameters in the square bracket, and so on. So, this is the model, right!

(Refer Slide Time: 05:38)



```
        return dxdt
```

```
[3]: import numpy as np
     def RK4_System(f,x0,t0,t1,dt):
         t = np.arange(t0,t1+dt,dt)
         nt = t.size
         nx = x0.size
         x = np.zeros((nx,nt))
         x[:,0] = x0
         for k in range(nt-1):
             k1 = dt*f(t[k],x[:,k])
             k2 = dt*f(t[k]+dt/2,x[:,k]+k1/2)
             k3 = dt*f(t[k]+dt/2,x[:,k]+k2/2)
             k4 = dt*f(t[k]+dt,x[:,k]+k3)
             dx = (k1+2*k2+2*k3+k4)/6.0
             x[:,k+1] = x[:,k]+dx
         return x, t
```

Now, let us write this RK4 method. We have already seen how to write this user defined function for RK4 method in case you have only one equation. Now, you have two equations for example, in this case. These k1, k2, k3, k4 in in RK4 method have to be computed for each of this equation, that is what it we have to do.

In general, for example, this program is a genetic program, where x could be any number of variables, f could be any function. So, here we are giving the input f, namely the right hand side of the x dash is equal to f (t, x), and x0 is the initial condition, t0 is the starting time, t1 is the end time, that is final time and dt is the step length in terms of time.

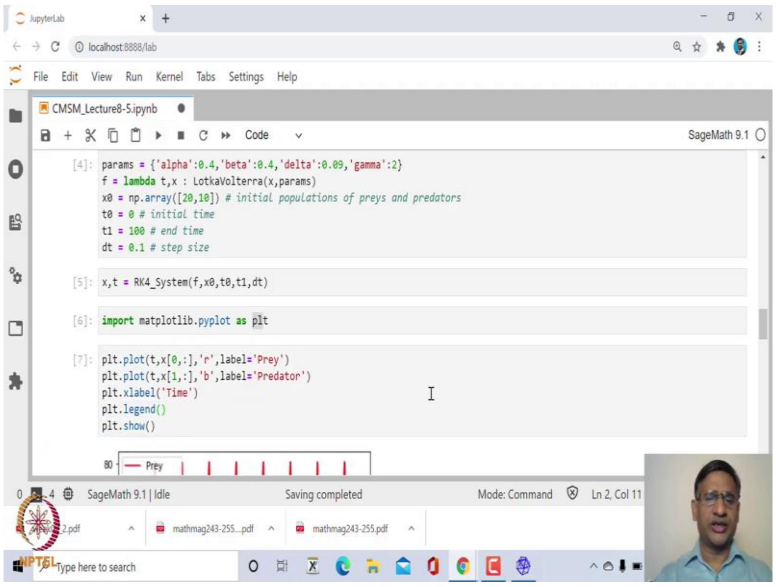
Let us first make t as the array going from t0 to t1 plus dt with a step length dt. Here, we are taking t1 plus dt, because the last end time should also be at which we want to evaluate this x. The number of points or node points in this, we will call as nt, and nx is the size of this x, which number of variables, in this case, it will be 2.

And let us initialize x as zero matrix of size nx cross nt. So nx is in this case 2, nt will be the number of node points. At so many node points, you have to evaluate. Initialize x as x0, this is all rows and first column; first column of x is initialized as x0, namely whatever is the initial population of rabbits and foxes in this case.

This is quite exactly similar; except that we are replacing this k1, k2, k3, k4 as a vector function. It is dt, in earlier case, it was h times f of t k and at x all the rows and kth column. Similarly, k2, k3, k4 and then define dx, that is, next iterate which is k1 plus 2 times k2 plus 2 times k3 plus k 4 divided by 6 and then increment x to the next point that is k plus 1.

So, here this is x all the rows and next column will be initialized as this plus dx, right. That is how we can write this RK4 method for solving system of first order differential equations.

(Refer Slide Time: 09:09)



```
[4]: params = {'alpha':0.4,'beta':0.4,'delta':0.09,'gamma':2}
     f = lambda t,x : LotkaVolterra(x,params)
     x0 = np.array([20,10]) # initial populations of preys and predators
     t0 = 0 # initial time
     t1 = 100 # end time
     dt = 0.1 # step size

[5]: x,t = RK4_System(f,x0,t0,t1,dt)

[6]: import matplotlib.pyplot as plt

[7]: plt.plot(t,x[0,:],'r',label='Prey')
     plt.plot(t,x[1,:],'b',label='Predator')
     plt.xlabel('Time')
     plt.legend()
     plt.show()
```

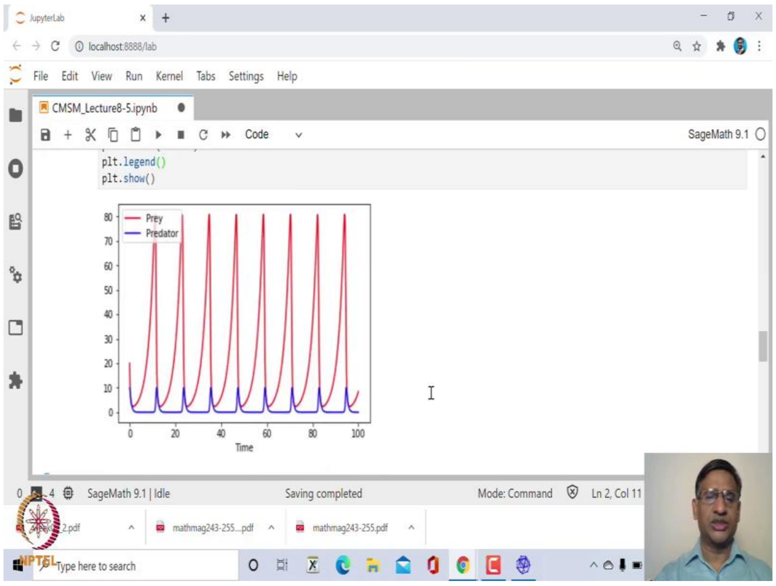
Now let us first run this and then let us now define the parameters.

We are taking the parameters alpha is equal to 0.4, beta is equal to 0.4, delta is equal to 0.09 and gamma is equal to 2 and f is the lambda function defined over t, x of lotkavolterra x and with parameters. Then initialize x0, which is the initial population of the preys and predators.

Let us say there are 20 preys and 10 predators to begin with, t0=0, that is initial time, t1 is, let us say 100, that is, the end time, and dt the step length is 0.1. So, let us run this and now. Let us call this RK 4_system which we have created. It will give you two outputs, so one is x which is the x1, x2, xn at time t, and also the time with step length delta t.

Let us run this, and once we have run this, then now let us try to plot. Let us try to visualize the growth of each of these species namely the preys and predator, in this case we are considering rabbits and foxes. For example, let us import first of all py plot from matplotlib as plt and first let us plot the preys in red color. It is plotted against time and x0 colon, that is all the zeroth row, that is the first variable and all the columns that is for each time you are considering. Similarly, plot the predator and then label the x axis as time and also plot the legend, that means it will show the name of the each curve.

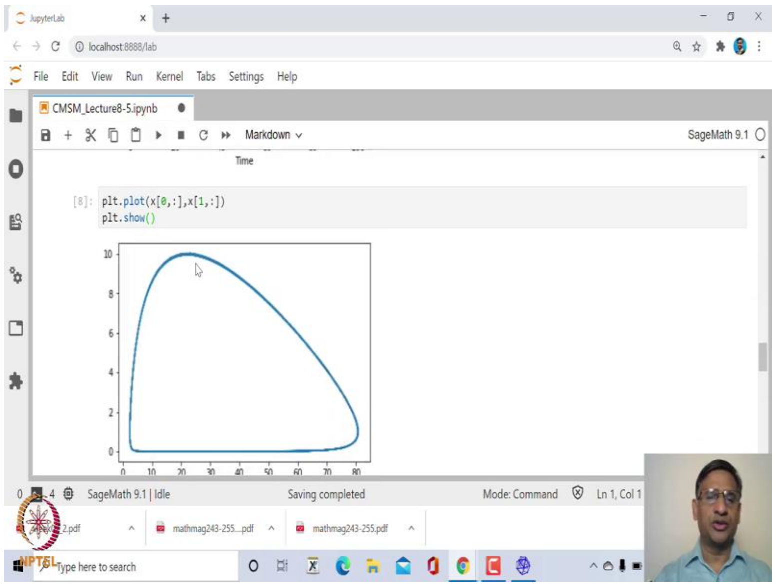
(Refer Slide Time: 11:14)



```
plt.legend()
plt.show()
```

Let us plot and see how they look like. This red one is the prey and blue one is the predator. As you can see here, whenever the there is a growth in the preys; the prey predator also grows and once the preys population comes down, the predator population also comes down. That is what you expect.

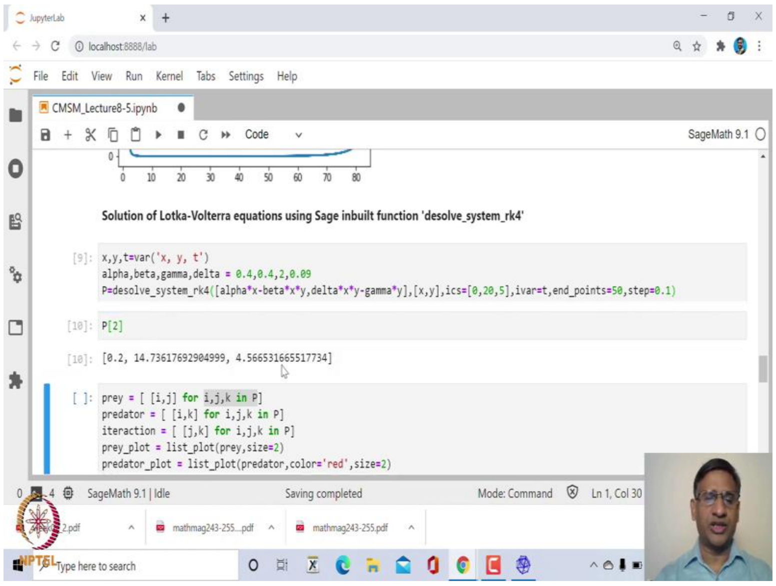
(Refer Slide Time: 11:37)



```
[8]: plt.plot(x[0,:],x[1,:])
     plt.show()
```

Now let us plot both these together, that will give you interaction between the two species namely prey and predator. This is how it looks like. You can do it with different initial guess and different parameters.

(Refer Slide Time: 11:57)



Solution of Lotka-Volterra equations using Sage inbuilt function 'desolve_system_rk4'

```
[9]: x,y,t=var('x, y, t')
     alpha,beta,gamma,delta = 0.4,0.4,2,0.09
     P=desolve_system_rk4([alpha*x-beta*x*y,delta*x*y-gamma*y],[x,y],ics=[0,20,5],ivar=t,end_points=50,step=0.1)

[10]: P[2]

[10]: [0.2, 14.73617692904999, 4.566531665517734]

[ ]: prey = [ [i,j] for i,j,k in P]
     predator = [ [i,k] for i,j,k in P]
     iteraction = [ [j,k] for i,j,k in P]
     prey_plot = list_plot(prey,size=2)
     predator_plot = list_plot(predator,color='red',size=2)
```

Let us look at the solution of the same problem using Sage inbuilt function desolve_system_rk4. As I mentioned earlier, Sage also has an inbuilt function for solving this system of first order differential equations using rk4 method. So, let us use that. We will define x, y, t as variables; instead of calling x1, x2, we are calling now x, y, t and alpha, beta, gamma are taken as 0.4, 0.4, 2 and 0.09.

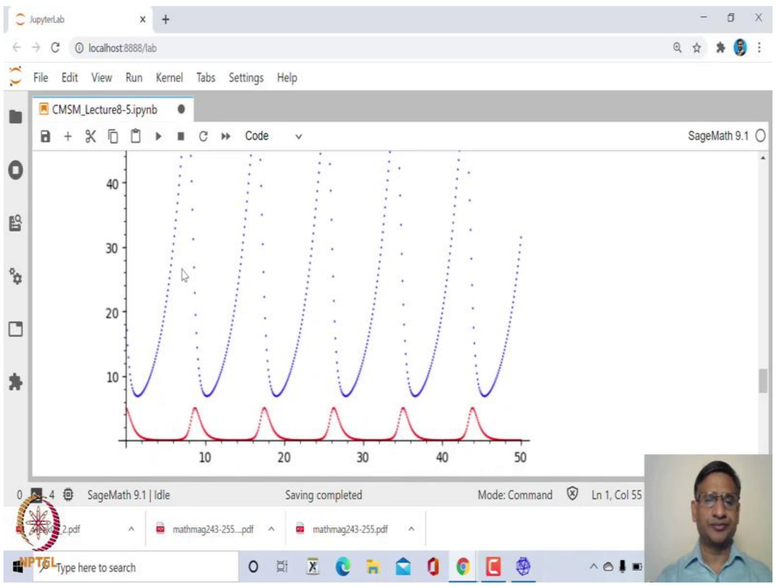
And then let us call this desolve_system_rk4 and we need to give the set of equations as a list, this is the first equation namely dx by dt, the right hand side of that and this is the right hand side of dy by dt.

And we are solving with respect to variables x and y, with initial conditions 20 comma 5. So, 20 rabbits and 5 foxes and beginning time is 0, by default beginning time is 0, the step length is 0.01, which is quite small. If you want, you can make it 0.1. Let us run this, it may take few seconds. Once we have obtained this; then we can plot graph of growth of predator and prey separately and both together, and then also the interaction between the two. Let us just wait for a second, it is still computing, it has done the computation.

Now, let us look at, how we are going to plot. Let us store the the preys as list of i comma j, where i, j, k will vary in P. If you look at P, P actually is going to give you the xi, value of preys, yi, values of predator, and the t values. That is why it will give you three components, as a list, so it will be a list of three real numbers.

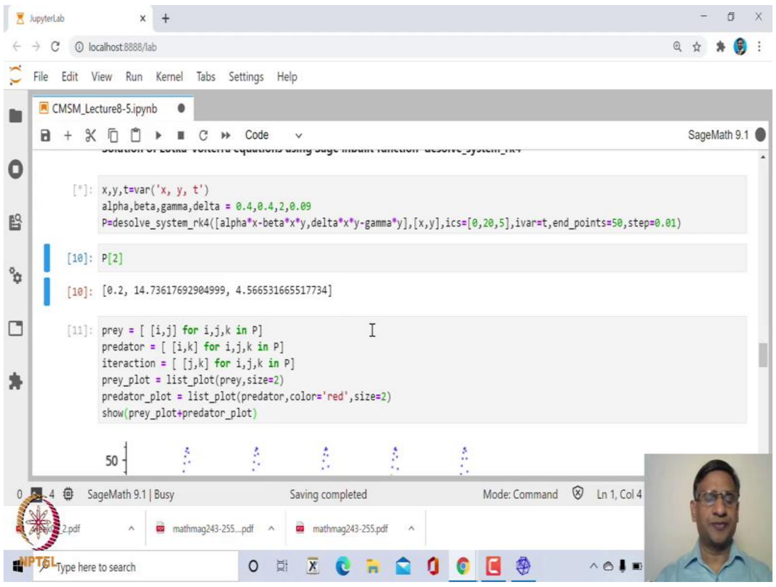
If you want, you can just see here. Let me run this separately. P comma, let us say P of 2. So, P of 2, you can see here, this is a three elements. Similarly store the predator and then the interaction term will be in terms of j, k and then plot the prey and predator. In this we are using list plot. The List of the points you are plotting and then we are showing these two together right.

(Refer Slide Time: 15:07)



This is how it looks like and that is why in case we make this step size very small; then it will look much better.

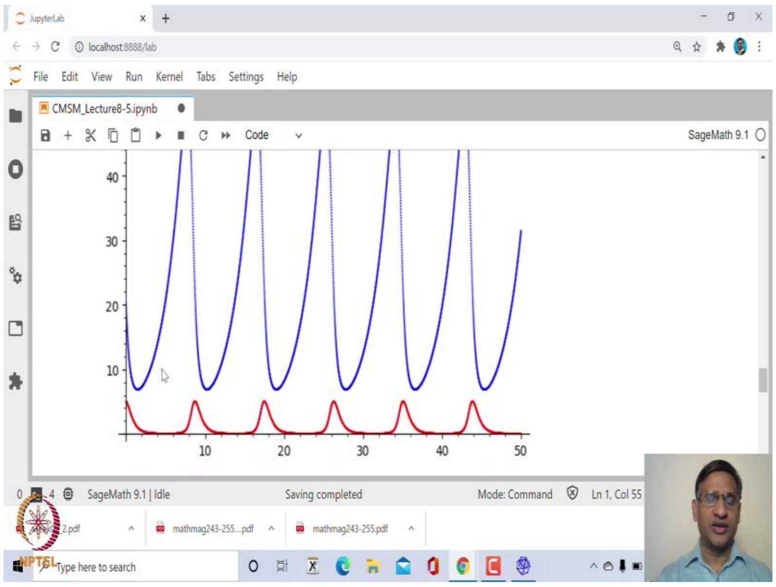
(Refer Slide Time: 15:16)



```
[*]: x,y,t=var('x, y, t')
     alpha,beta,gamma,delta = 0.4,0.4,2,0.09
     P=desolve_system_rk4([alpha*x-beta*x*y,delta*x*y-gamma*y],[x,y],ics=[0,20,5],ivar=t,end_points=50,step=0.01)

[10]: P[2]

[10]: [0.2, 14.73617692904999, 4.566531665517734]

[11]: prey = [ [i,j] for i,j,k in P]
      predator = [ [i,k] for i,j,k in P]
      iteraction = [ [j,k] for i,j,k in P]
      prey_plot = list_plot(prey,size=2)
      predator_plot = list_plot(predator,color='red',size=2)
      show(prey_plot+predator_plot)
```

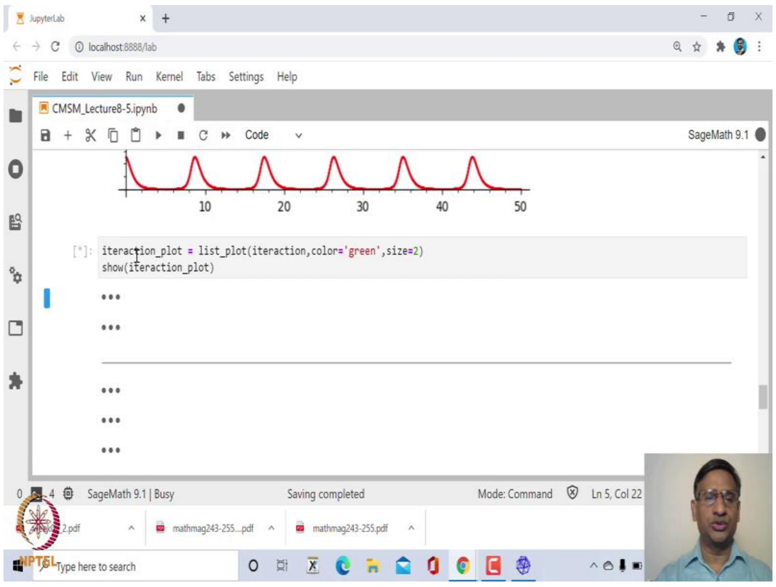
Of course, it will take more time to solve, let us just wait, we have decreased, now it has done. So, generally when you are running this the first time it will take little more time and next time it becomes faster.

(Refer Slide Time: 15:35)



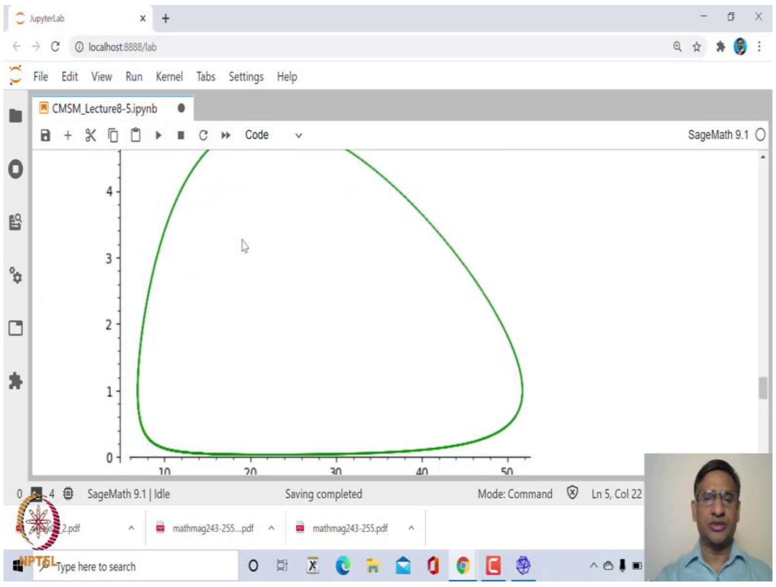
When you plot, it is almost like a curve. Again you can see here, when the preys population grows, the predator population also grows and vice versa.

(Refer Slide Time: 15:51)


```
[*]: iteraction_plot = list_plot(iteraction,color='green',size=2)
     show(iteraction_plot)
```

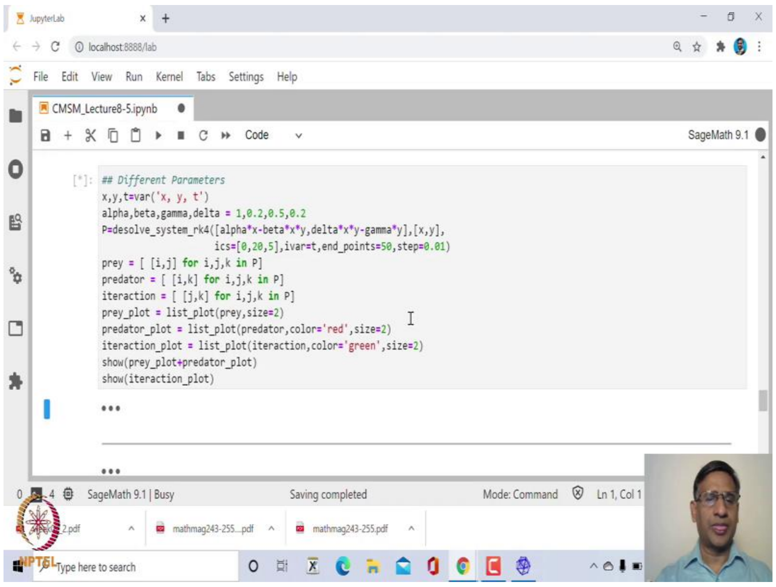
Let us plot the interaction between the two. When you plot the interaction between two, this is how it looks like.

(Refer Slide Time: 15:54)



This is exactly same as how we obtain the solution using our inbuilt function RK4_System.

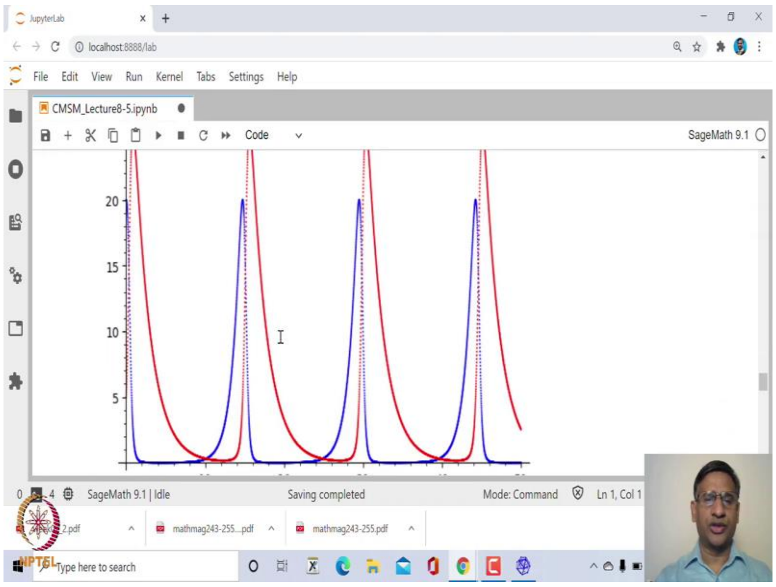
(Refer Slide Time: 16:04)



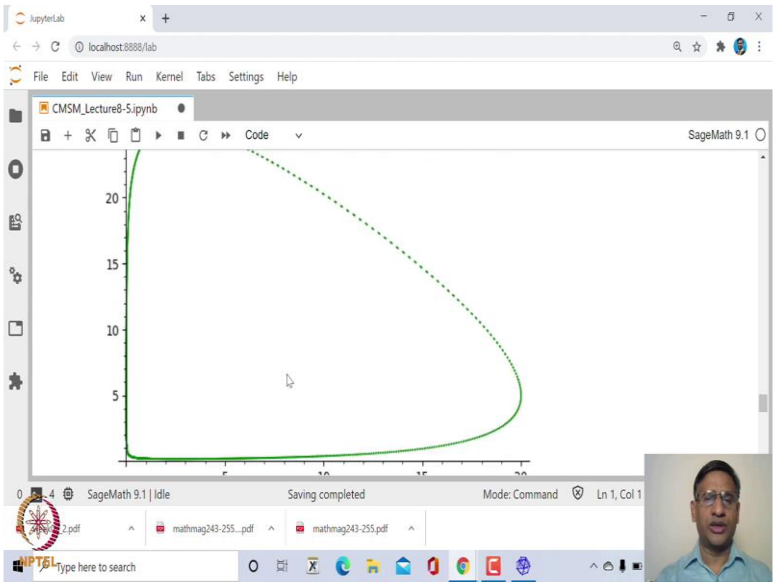
```
## Different Parameters
x,y,t=var('x, y, t')
alpha,beta,gamma,delta = 1,0.2,0.5,0.2
P=desolve_system_rk4([alpha*x-beta*x*y,delta*x*y-gamma*y],[x,y],
                     ics=[0,20,5],ivar=t,end_points=50,step=0.01)
prey = [ [i,j] for i,j,k in P]
predator = [ [i,k] for i,j,k in P]
iteraction = [ [j,k] for i,j,k in P]
prey_plot = list_plot(prey,size=2)
predator_plot = list_plot(predator,color='red',size=2)
iteraction_plot = list_plot(iteraction,color='green',size=2)
show(prey_plot+predator_plot)
show(iteraction_plot)
```

You can explore with the different parameters, for example, let us take alpha to be 1, beta to be 0.2, gamma to be 0.5 and delta to be 0.2, and then when you try to plot all this together, this is how it looks like now.

(Refer Slide Time: 16:26)

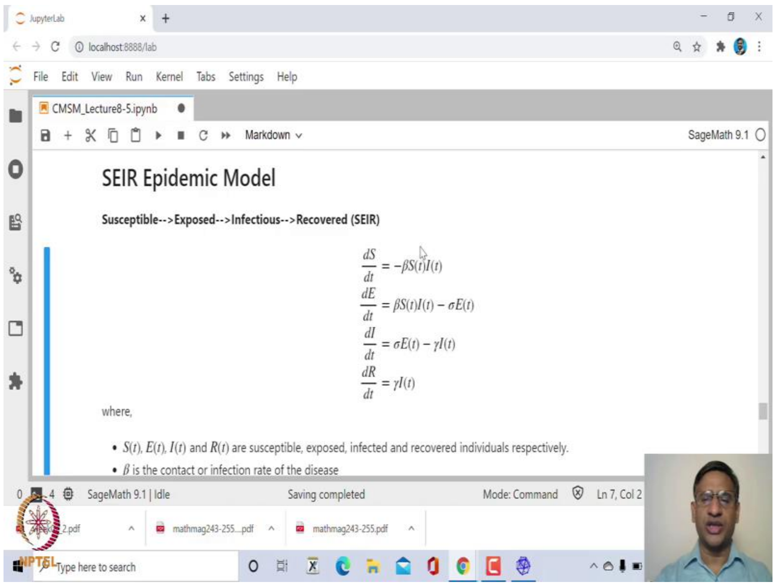


(Refer Slide Time: 16:29)



You can look at what is effect of these parameters on population of or the growth of predator and preys by exploring the different populations and different parameters.

(Refer Slide Time: 16:48)



SEIR Epidemic Model

Susceptible-->Exposed-->Infectious-->Recovered (SEIR)

$$\frac{dS}{dt} = -\beta S(t)I(t)$$
$$\frac{dE}{dt} = \beta S(t)I(t) - \sigma E(t)$$
$$\frac{dI}{dt} = \sigma E(t) - \gamma I(t)$$
$$\frac{dR}{dt} = \gamma I(t)$$

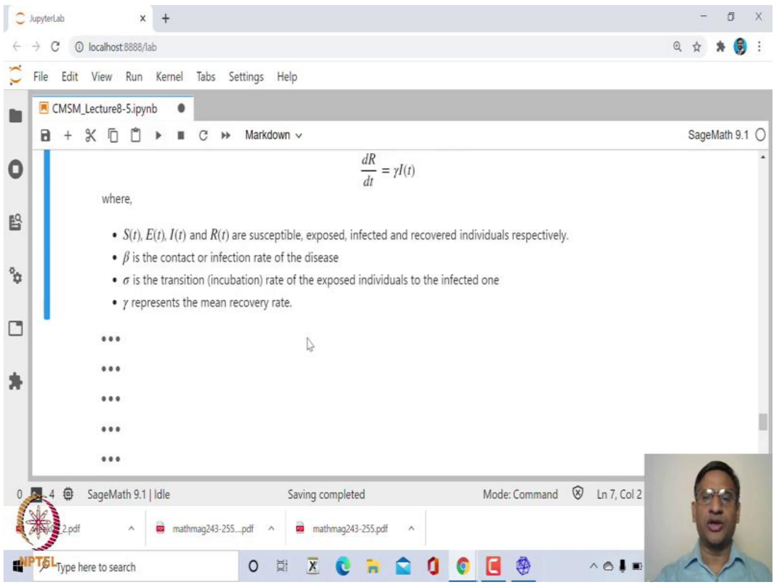
where,

- $S(t)$, $E(t)$, $I(t)$ and $R(t)$ are susceptible, exposed, infected and recovered individuals respectively.
- β is the contact or infection rate of the disease

Now, let us look at one more problem, this is what is called epidemic model, SEIR Epidemic model and actually we are at present in this unfortunate pandemic situation of this COVID-19. if you try to just Googl search pandemic models or let us say COVID pandemic models, you will get lots of mathematical models and one of them is SEIR, there are several of them.

So, let us look at how we can explore this particular model. How is this model defined? This is actually SEIR, S stands for Susceptible, E for Exposed, I is Infections and R is Recovered. Here the death is not taken into consideration. This is a system of four first order non-linear equations.

(Refer Slide Time: 17:54)



$$\frac{dR}{dt} = \gamma I(t)$$

where,

- $S(t)$, $E(t)$, $I(t)$ and $R(t)$ are susceptible, exposed, infected and recovered individuals respectively.
- β is the contact or infection rate of the disease
- σ is the transition (incubation) rate of the exposed individuals to the infected one
- γ represents the mean recovery rate.

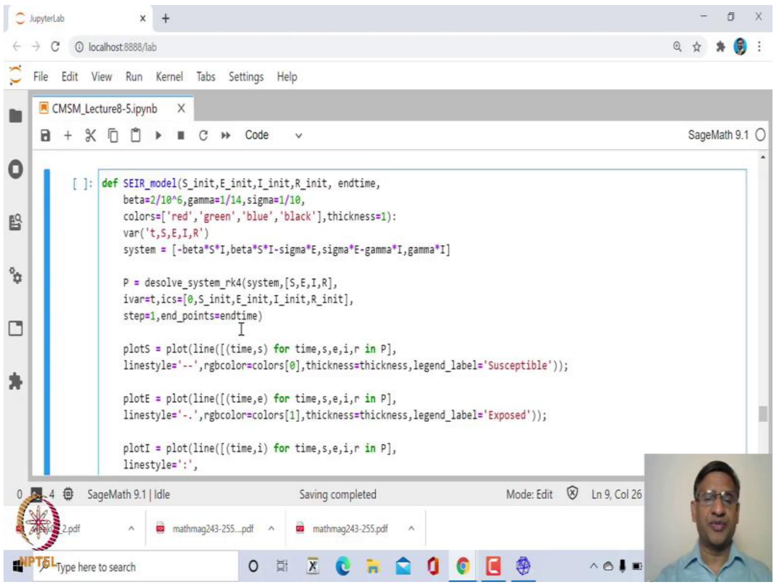
dS by dt that is the susceptible rate of increase of susceptible patient is given by minus beta times S(t) I(t), where beta is the contact or infection rate of the disease;

dE by dt is given by beta S(t) I(t) minus sigma E(t), here sigma is the transition rate or incubation rate.

dI by dt is sigma E(t) minus gamma of I(t), where as gamma is called the mean recovery rate.

These parameters beta, sigma and gamma governs this particular system of equations, which tells you how this susceptible patient exposed to this disease, infectious and recovery takes place. Let us look at how we can program this in order to basically visualize the solution of this particular model.

(Refer Slide Time: 19:02)

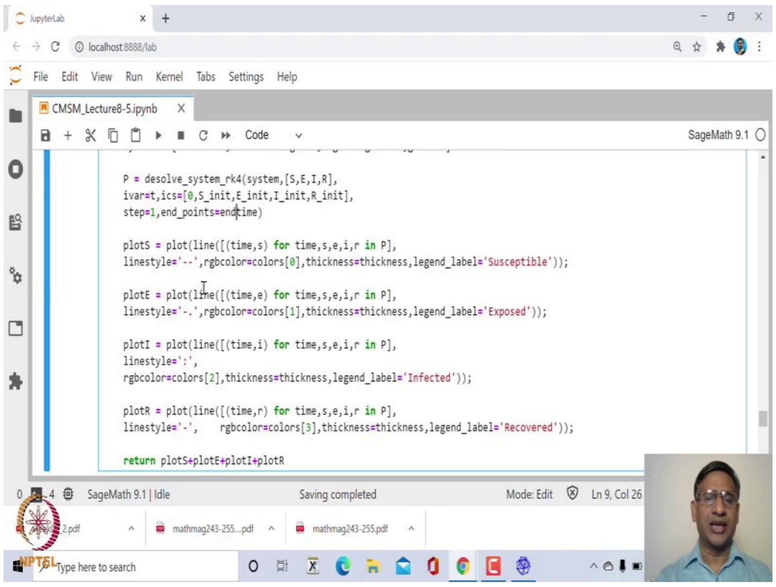


```
[ ]: def SEIR_model(S_init,E_init,I_init,R_init, endtime,
         beta=2/10^6,gamma=1/14,sigma=1/10,
         colors=['red','green','blue','black'],thickness=1):
         var('t,S,E,I,R')
         system = [-beta*S*I,beta*S*I-sigma*E,sigma*E-gamma*I,gamma*I]

         P = desolve_system_rk4(system,[S,E,I,R],
         ivar=t,ics=[0,S_init,E_init,I_init,R_init],
         step=1,end_points=endtime)

         plotS = plot(line([(time,s) for time,s,e,i,r in P],
         linestyle='--',rgbcolor=colors[0],thickness=thickness,legend_label='Susceptible'));

         plotE = plot(line([(time,e) for time,s,e,i,r in P],
         linestyle='-.',rgbcolor=colors[1],thickness=thickness,legend_label='Exposed'));

         plotI = plot(line([(time,i) for time,s,e,i,r in P],
         linestyle=':',
```

So, we are making a user defined function, namely is SEIR_model and we are giving initial population, that is, initialize S, E, I and capital R and also define the end time. Beginning time will always be 0, you can mention beginning time also, and we are taking beta to be 2 upon 10 to power minus 6, which is very small, gamma 1 by 14, sigma 1 by 10. If you want you can just change and take the different colors, so there will be four plots. So, these four plots are in different colors and this is the thickness of the curve. Now let us define the variable t, S, E, I, R and then define the system. So, it will be a list of four coordinates. The first coordinate is the right hand side of dS by dt, second coordinate is right hand side of dI by d t, third is the right hand side of dI by dt, and the last one is the right hand side of dR by dt.

So, now let us make use of inbuilt function desolve_system_rk4 and then solve this system which we have defined with respect to S, E, I, R, and this, initial variable is t, initial condition is 0, S_initial, E initial, I initial, and R initial and step length, let us do it, per day for example, and then end time.

(Refer Slide Time: 20:46)

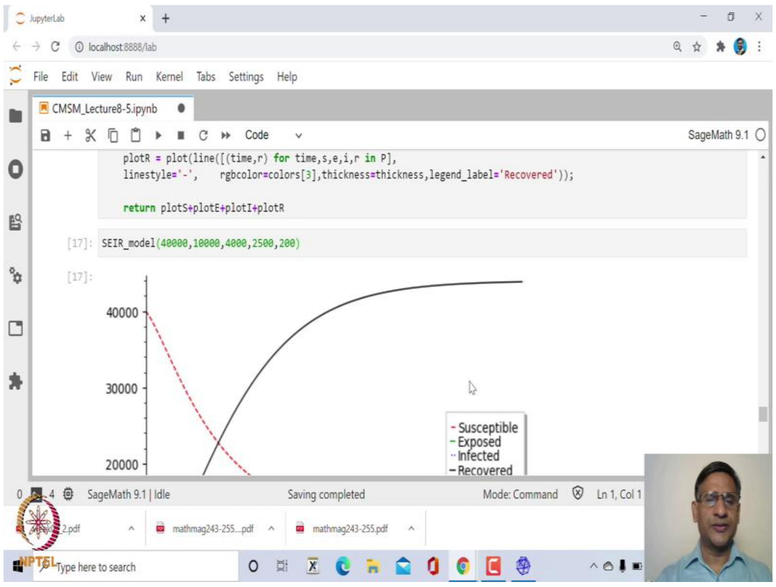


```
P = desolve_system_rk4(system,[S,E,I,R],
ivar=t,ics=[0,S_init,E_init,I_init,R_init],
step=1,end_points=endtime)

plotS = plot(line([(time,s) for time,s,e,i,r in P],
linestyle='--',rgbcolor=colors[0],thickness=thickness,legend_label='Susceptible'));

plotE = plot(line([(time,e) for time,s,e,i,r in P],
linestyle='-.',rgbcolor=colors[1],thickness=thickness,legend_label='Exposed'));

plotI = plot(line([(time,i) for time,s,e,i,r in P],
linestyle=':',
rgbcolor=colors[2],thickness=thickness,legend_label='Infected'));

plotR = plot(line([(time,r) for time,s,e,i,r in P],
linestyle='-',    rgbcolor=colors[3],thickness=thickness,legend_label='Recovered'));

return plotS+plotE+plotI+plotR
```

And then each one you are just plotting. What we are going to do? This particular, when you solve this P will have four coordinates, namely S, I, E, R and of course t will also be there. Let us plot this first one as a line in terms of time, and the time is going to the first coordinate in this P. So, time S, E, I, R will be the value of each element in the list.

And then plotE, plotI and plotR. This is very fairly simple model, I could have written some explanation here. Let me just once again show you this model. Let me run this.

(Refer Slide Time: 21:57)

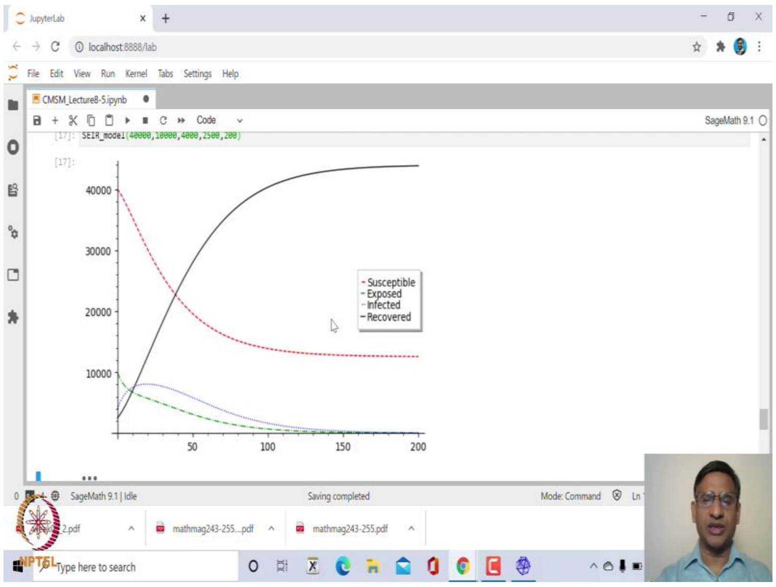


```
plotR = plot(line([(time,r) for time,s,e,i,r in P],
linestyle='-',    rgbcolor=colors[3],thickness=thickness,legend_label='Recovered'));

return plotS+plotE+plotI+plotR
```

```
[17]: SEIR_model(40000,10000,4000,2500,200)
```

Once you have run this, then next we can plot. So, let us call this SEIR model with initial population 40000 and this E is the exposed one which is10000, sorry, this is I which is the infected ones which is 4000 and this is the R which is the recovered 2500 and over the period of 200 days.

(Refer Slide Time: 22:25)

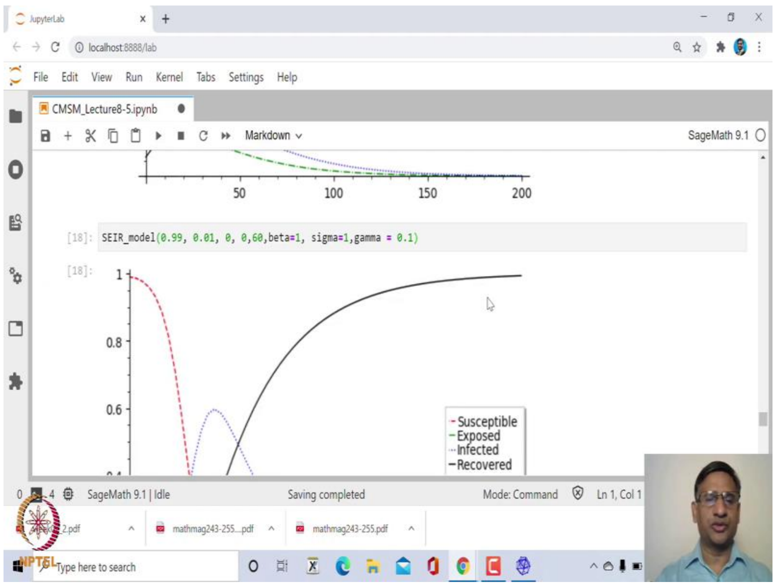


```
[17]: SEIR_model(40000,10000,4000,2500,200)
```

So, when you plot this, this is what you get. Let me make it slightly small, so that you can see the entire curve.

You can see here this red one is susceptible, E is exposed that is the green one is exposed and this is the infected one, the blue one has been infected and this is the recovery, this is the recovered one. So, you can see here in the beginning the recovery is small and slowly it goes up and then slowly very body gets recovered. This is the susceptible ones; initially there everybody is susceptible and then slowly it the goes down and it stabilizes after some time. So, this is the plot of this solution of this particular SEIR mod pandemic model.

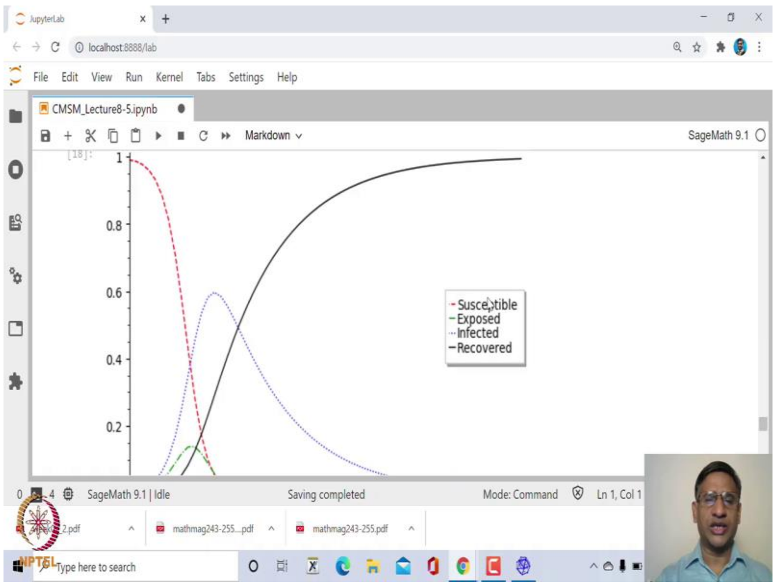
(Refer Slide Time: 23:26)



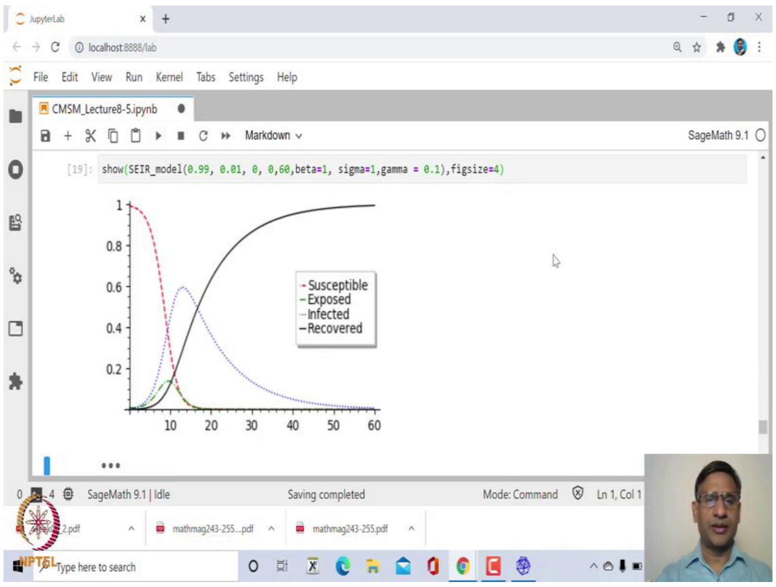
```
[18]: SEIR_model(0.99, 0.01, 0, 0,60,beta=1, sigma=1,gamma = 0.1)
```

Now, you can try with let us say different parameters. For example, let us take initial population about 99 percent, let us say infected and only one 0.1 that is 1 percent are exposed to this virus and 0 infected, 0 recovered to begin with and over the period of 60 days and start with beta is equal to 1, sigma is equal to 1 and gamma is equal to 0.1 and then if you look at the solution curve, this is how it looks like.

(Refer Slide Time: 24:01)



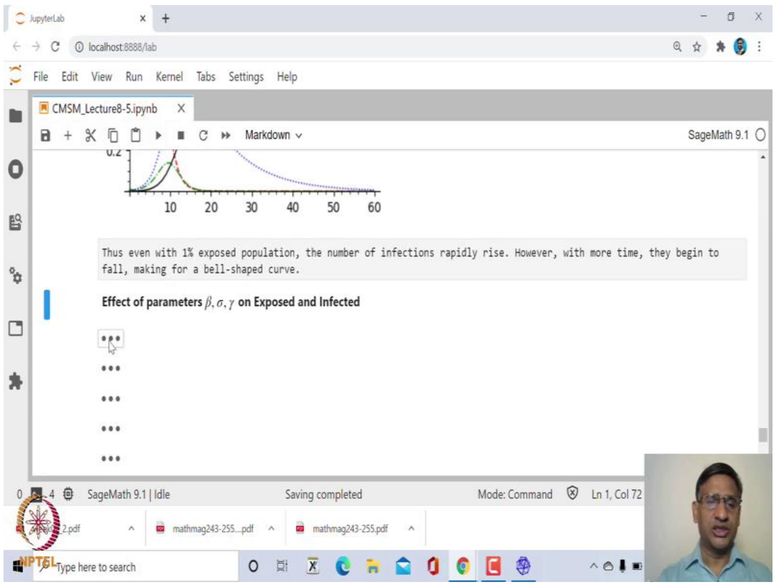
(Refer Slide Time: 24:10)



```
[19]: show(SEIR_model(0.99, 0.01, 0, 0,60,beta=1, sigma=1,gamma = 0.1),figsize=4)
```

Let me again reduce the figure size, let me say show this and figsize is equal to let us say 4, it should work. Now, you can see here, with 1 percent population exposed, you can see here what is happening is, the infection grows quite rapidly, but then it follows Gaussian distribution and then it goes down and slowly it becomes 0.

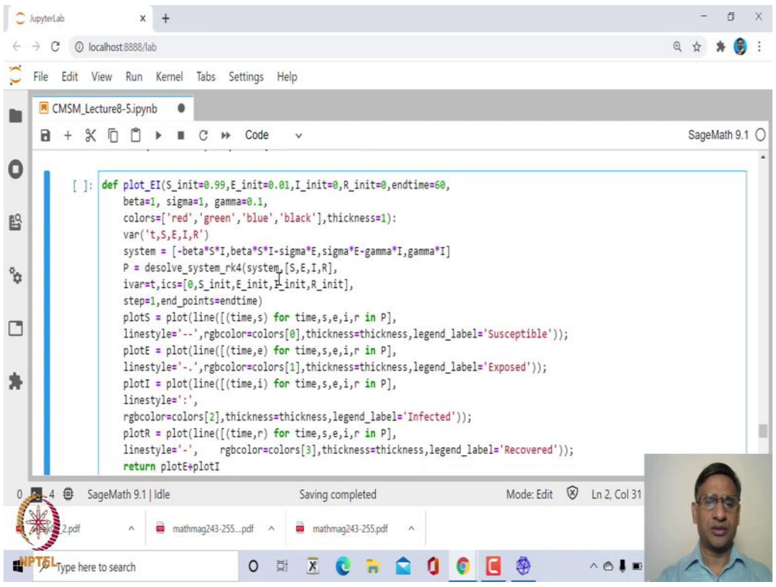
(Refer Slide Time: 24:58)


Thus even with 1% exposed population, the number of infections rapidly rise. However, with more time, they begin to fall, making for a bell-shaped curve.

Effect of parameters β, σ, γ on Exposed and Infected

That is how you can visualize this. You can also look at what happens when you change these parameters and just look at the effect of these parameters on exposed and infected individuals.

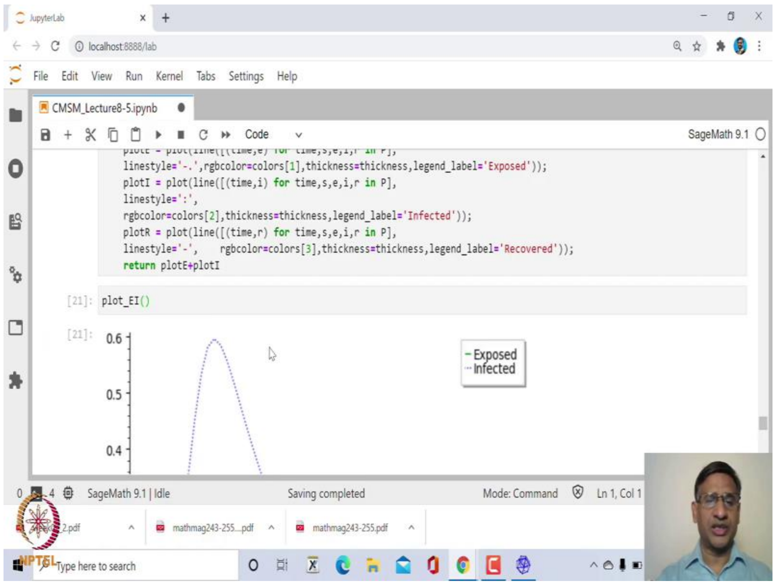
(Refer Slide Time: 25:10)


```
[ ]: def plot_EI(S_init=0.99,E_init=0.01,I_init=0,R_init=0,endtime=60,
         beta=1, sigma=1, gamma=0.1,
         colors=['red','green','blue','black'],thickness=1):
         var('t,S,E,I,R')
         system = [-beta*S*I,beta*S*I-sigma*E,sigma*E-gamma*I,gamma*I]
         P = desolve_system_rk4(system,[S,E,I,R],
         ivar=t,ics=[0,S_init,E_init,I_init,R_init],
         step=1,end_points=endtime)
         plotS = plot(line([(time,s) for time,s,e,i,r in P],
         linestyle='--',rgbcolor=colors[0],thickness=thickness,legend_label='Susceptible'));
         plotE = plot(line([(time,e) for time,s,e,i,r in P],
         linestyle='-.',rgbcolor=colors[1],thickness=thickness,legend_label='Exposed'));
         plotI = plot(line([(time,i) for time,s,e,i,r in P],
         linestyle=':',
         rgbcolor=colors[2],thickness=thickness,legend_label='Infected'));
         plotR = plot(line([(time,r) for time,s,e,i,r in P],
         linestyle='-',    rgbcolor=colors[3],thickness=thickness,legend_label='Recovered'));
         return plotE+plotI
```

How do we do that? Again the same user defined function; in this case we have just plotted only the exposed one and the infected ones, apart from that everything is actually the same and we have taken this beta to be 1, sigma to be 1 and gamma to be 0.1.

So, this is quite easy, what possibly we can do is, you can take four outputs plot of E, plot of I, plot of R and plot of S separately and then we can combine together, that can also be done.

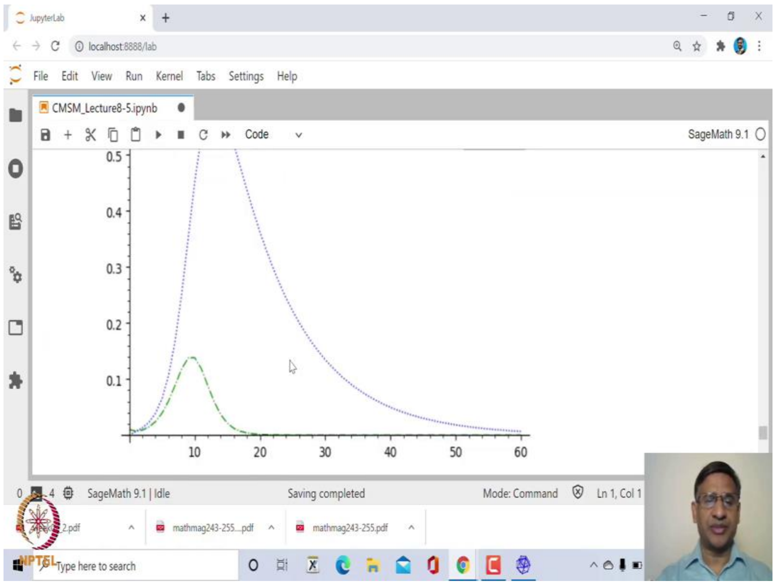
(Refer Slide Time: 25:49)



```
    plotE = plot(line([(time,e) for time,s,e,i,r in P],
    linestyle='-.',rgbcolor=colors[1],thickness=thickness,legend_label='Exposed'));
    plotI = plot(line([(time,i) for time,s,e,i,r in P],
    linestyle=':',
    rgbcolor=colors[2],thickness=thickness,legend_label='Infected'));
    plotR = plot(line([(time,r) for time,s,e,i,r in P],
    linestyle='-',    rgbcolor=colors[3],thickness=thickness,legend_label='Recovered'));
    return plotE+plotI

[21]: plot_EI()
```

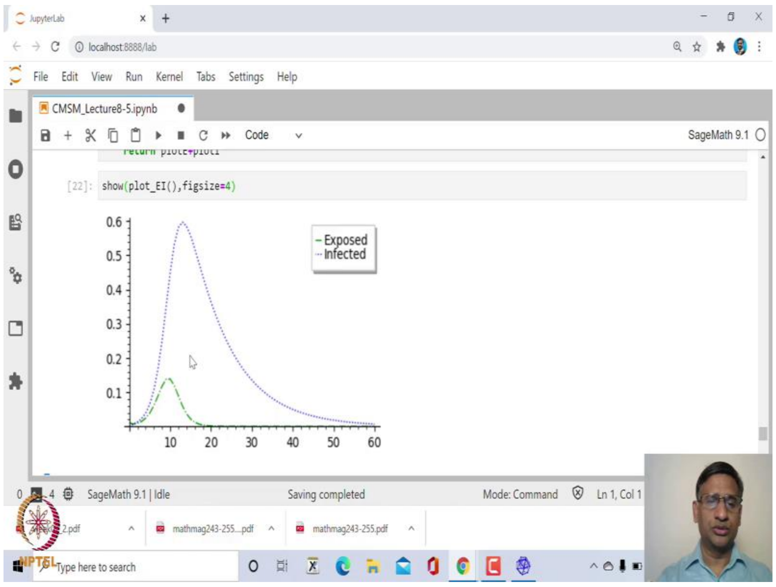
But in any case, let us see, first let me run this and after that let me call this with the default initial parameters.

(Refer Slide Time: 25:59)



Let me again show this with figsize is equal to 4.

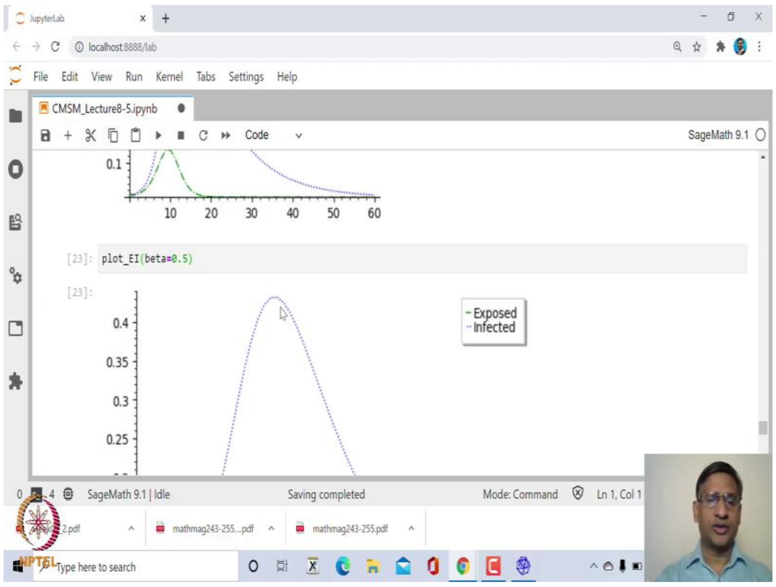
(Refer Slide Time: 26:02)



```
[22]: show(plot_EI(),figsize=4)
```

Then you can see here, this is the how the exposed and infected individuals look like under this default parameter, namely beta equal to 1, gamma equal to 0.1 and delta equal and sigma equal to 1.

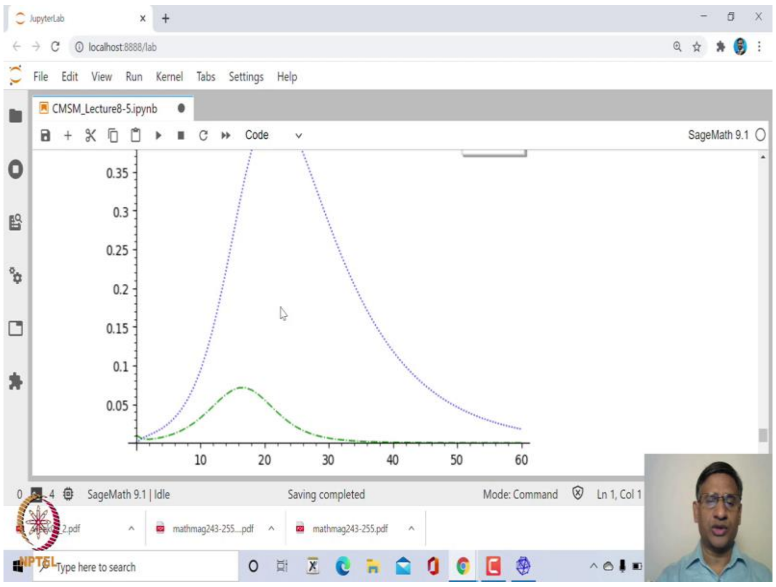
(Refer Slide Time: 26:36)



```
[23]: plot_EI(beta=0.5)
```

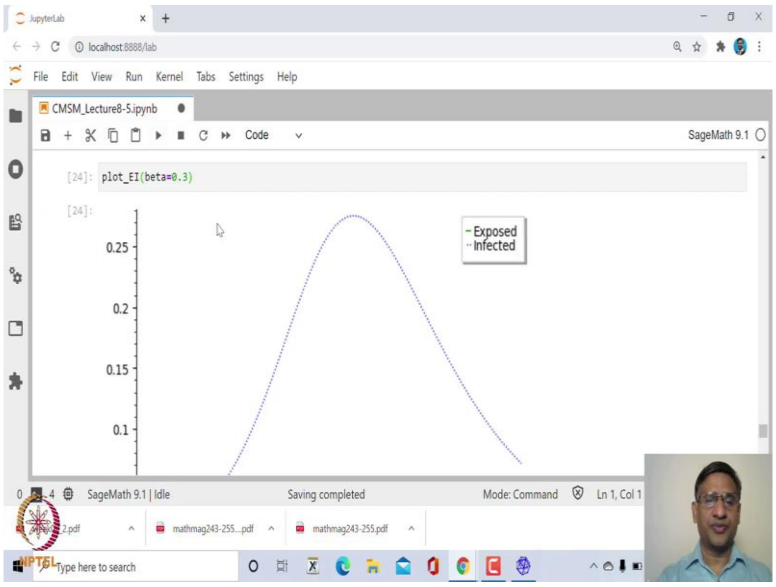
If you try to change this parameter. For example, if I plot this for beta equal to 0.5 and still other gamma and delta will be default ones and then the plot will be different.

(Refer Slide Time: 26:47)



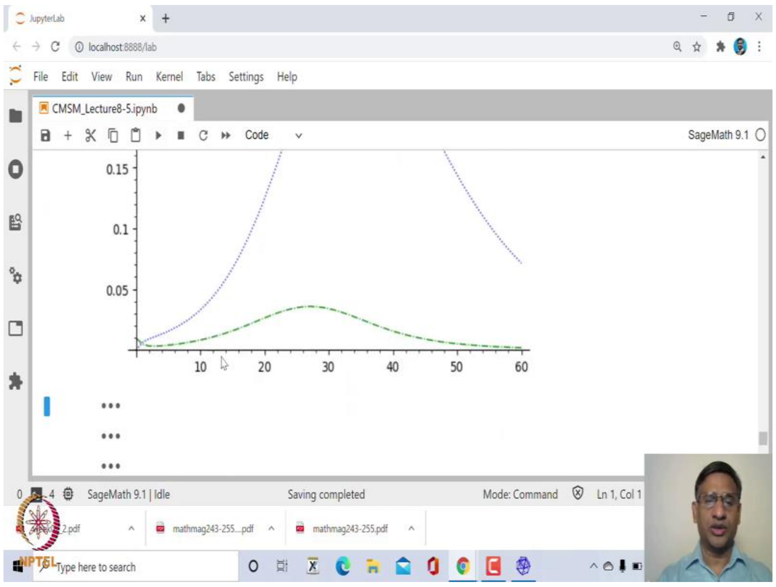
You can see it is a kind of more flat than the previous one.

(Refer Slide Time: 26:57)



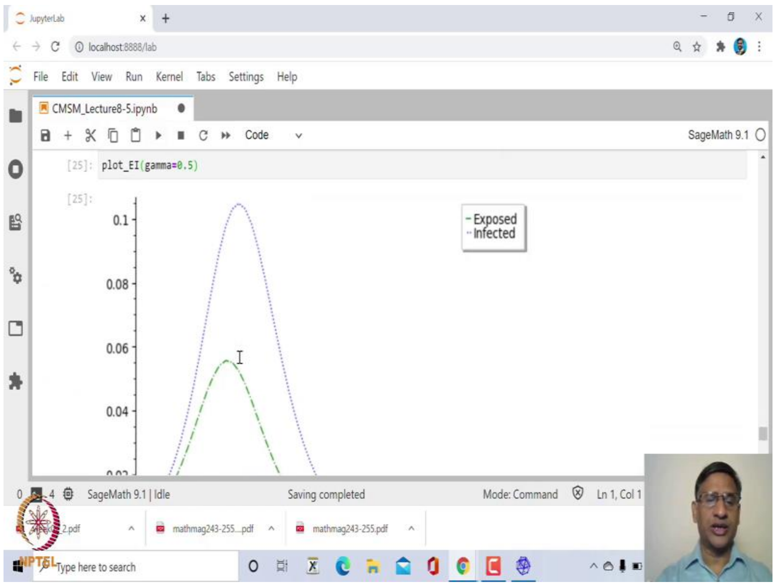
```
[24]: plot_EI(beta=0.3)
```

(Refer Slide Time: 26:59)



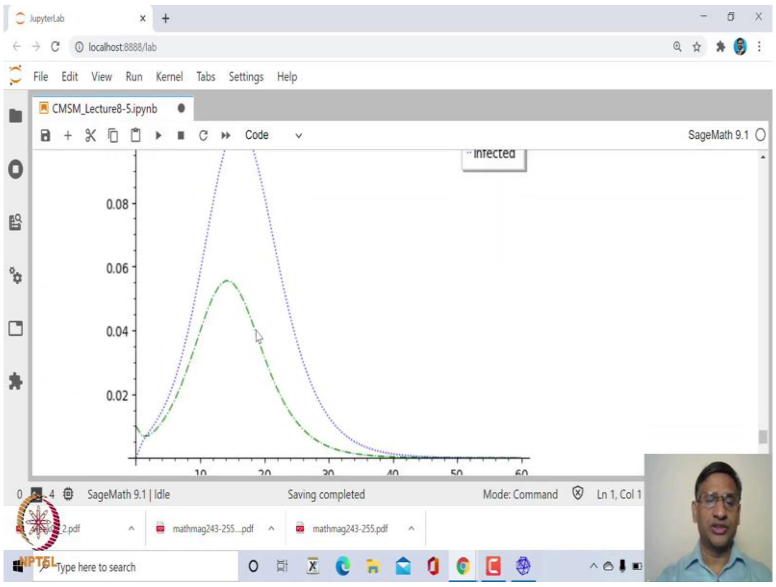
If I further reduce for example, make it 0.3, it will be even the flatter. So, this is how this beta has effect on exposed and infected individual.

(Refer Slide Time: 27:12)



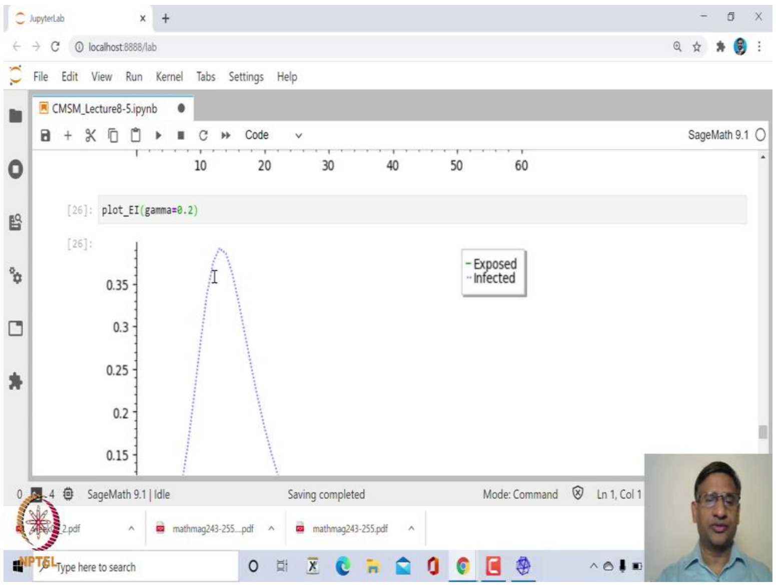
```
[25]: plot_EI(gamma=0.5)
```

(Refer Slide Time: 27:17)



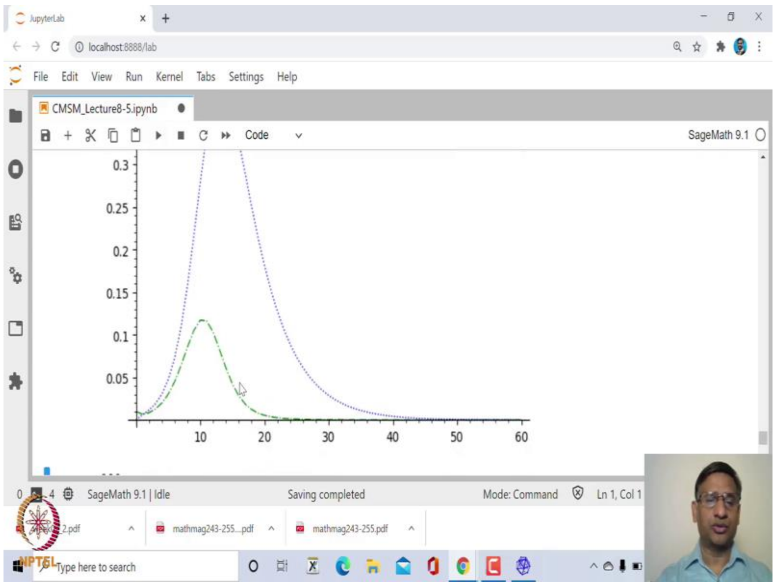
Similarly, you can try with other parameters. For example, if you change or reduce this gamma and then look at the plot, this is how it looks like.

(Refer Slide Time: 27:21)

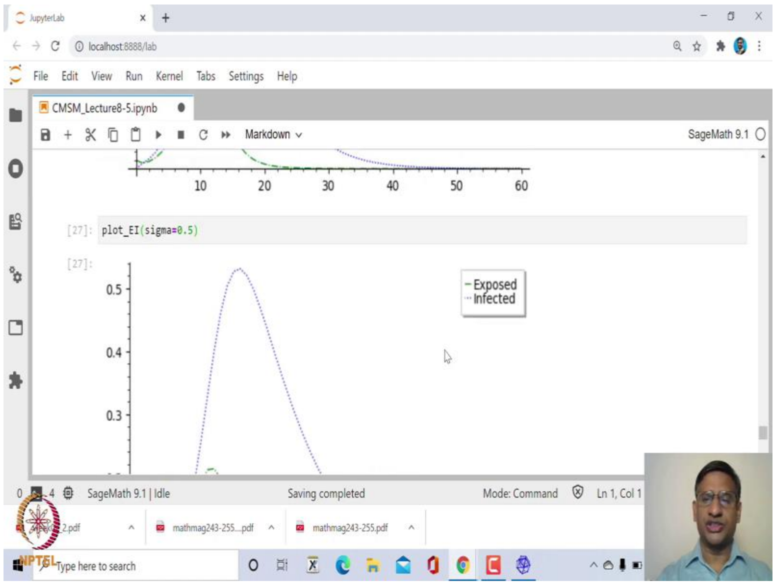

```
[26]: plot_EI(gamma=0.2)
```

If I reduce further, for example, if I make it 0.2, then you can see this is how it looks like and so on.

(Refer Slide Time: 27:25)

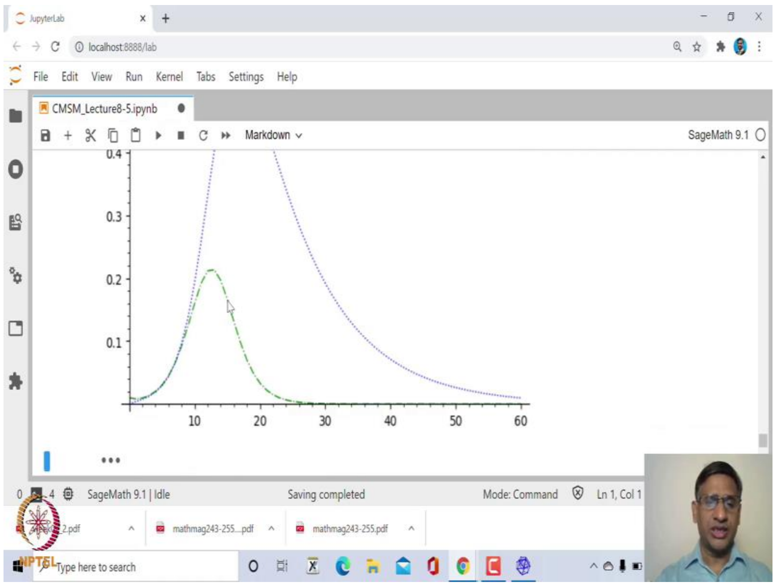


(Refer Slide Time: 27:29)



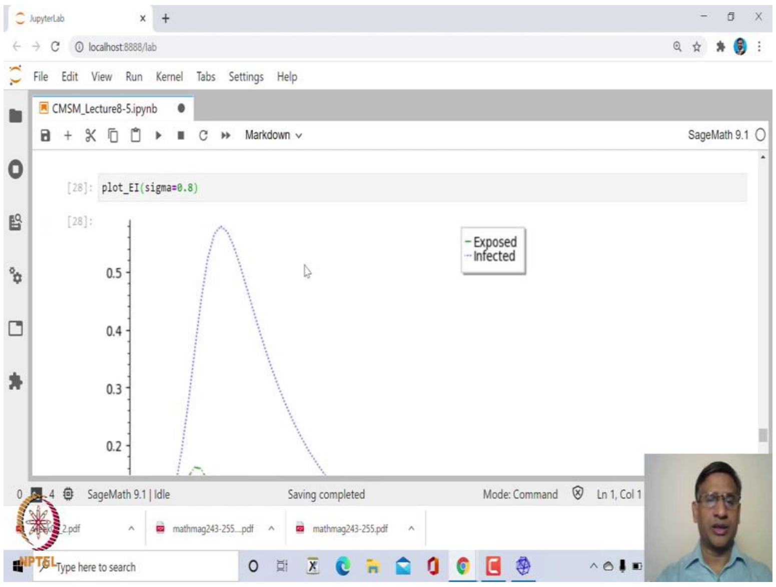
```
[27]: plot_EI(sigma=0.5)
```

(Refer Slide Time: 27:33)

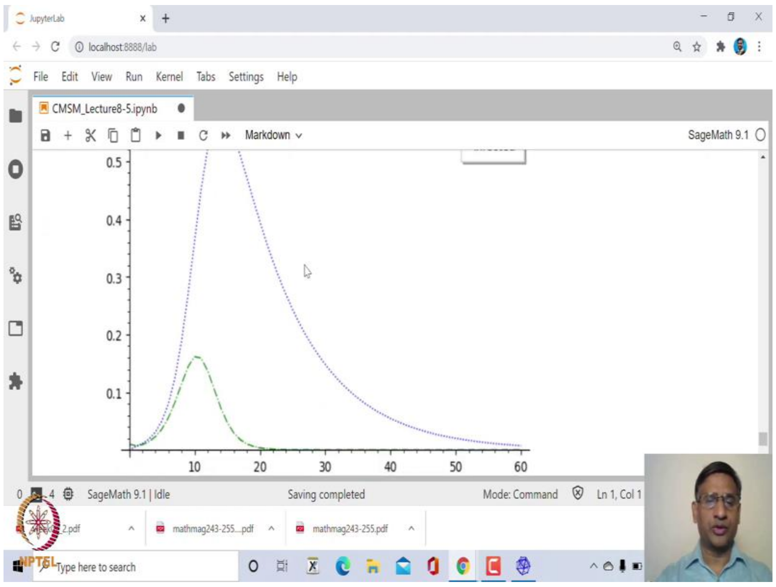


Similarly, we can look at how this sigma changes this output. So, when I reduce sigma to 0.5, this is how it looks like.

(Refer Slide Time: 27:50)

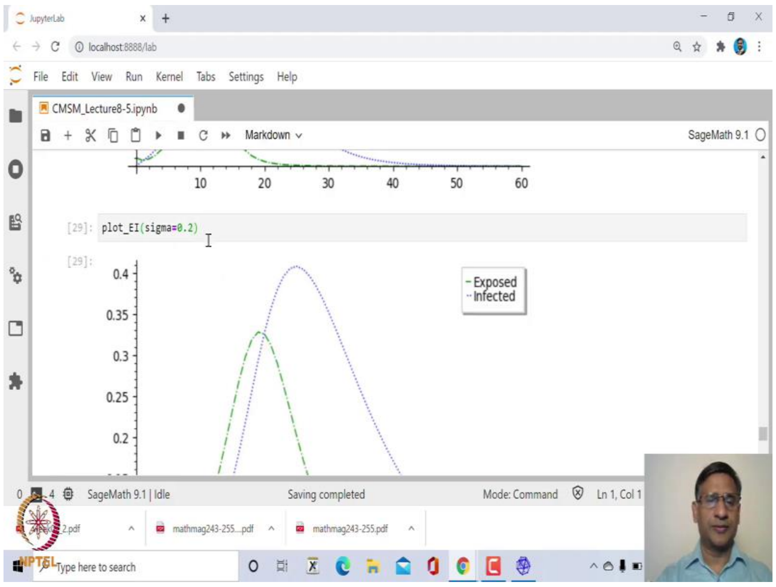

```
[28]: plot_EI(sigma=0.8)
```

(Refer Slide Time: 27:53)



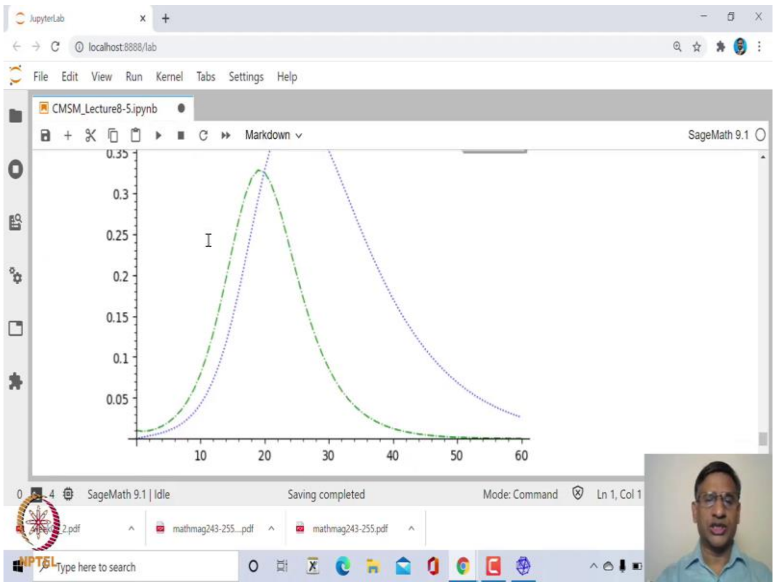
If I reduce further or let me increase it, if I say 0.8; then this is how it looks like.

(Refer Slide Time: 27:56)



```
[29]: plot_EI(sigma=0.2)
```

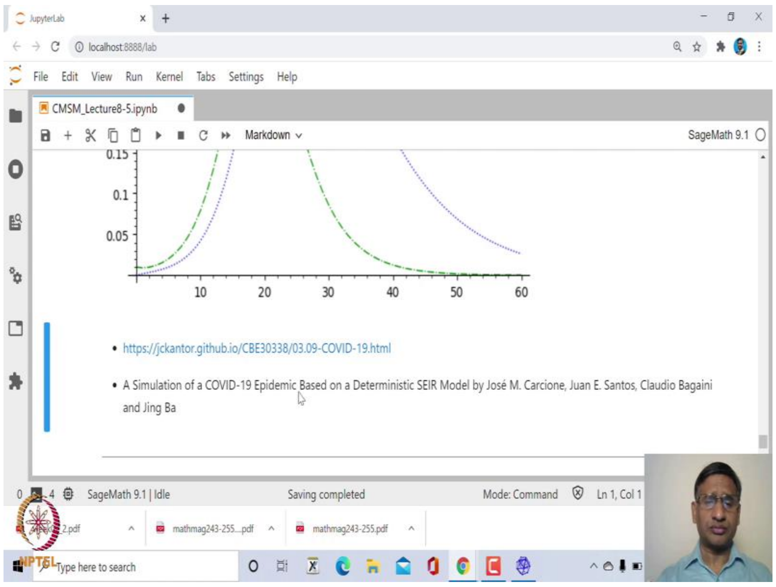
(Refer Slide Time: 27:59)



If I reduce to 0.2, then this is how it looks like.

You can explore effect of these parameters on each of these four variables, exposed, infected and recovered and the susceptible.

(Refer Slide Time: 28:20)



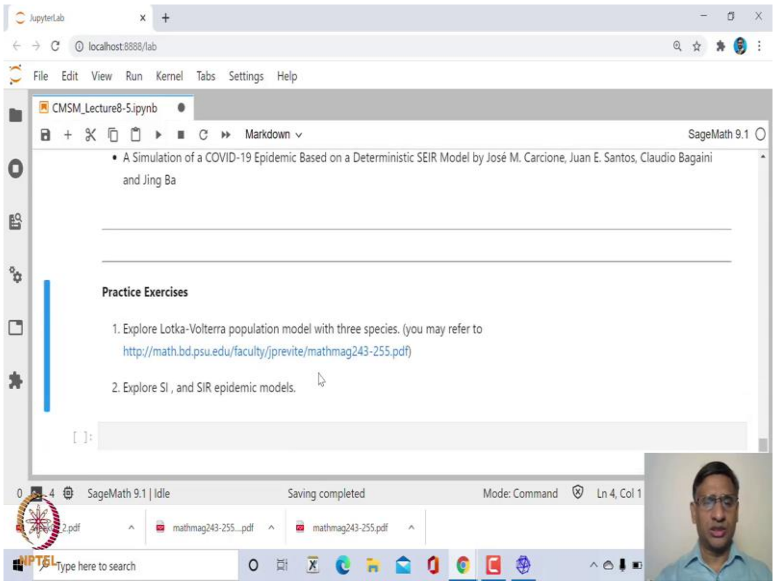
- https://jckantor.github.io/CBE30338/03.09-COVID-19.html
- A Simulation of a COVID-19 Epidemic Based on a Deterministic SEIR Model by José M. Carcione, Juan E. Santos, Claudio Bagaini and Jing Ba

You can do this experiment and try to see the effect and analyse accordingly.

As I said if you just Google search on COVID-19 epidemic model, you will get tons of references, papers; but you can look at one of them as a simulation of COVID-19 epidemic based on deterministic SEIR model by these four people, this is basically done for I think data obtained from Italy.

You can also look at this github page, where a python program is written, again for these different models.

(Refer Slide Time: 29:14)



- A Simulation of a COVID-19 Epidemic Based on a Deterministic SEIR Model by José M. Carcione, Juan E. Santos, Claudio Bagaini and Jing Ba

Practice Exercises

1. Explore Lotka-Volterra population model with three species. (you may refer to http://math.bd.psu.edu/faculty/jprevite/mathmag243-255.pdf)
2. Explore SI , and SIR epidemic models.

Let me leave you these two explorations. Try to explore this Lotka-Volterra population model for three species, for example, you can refer to this particular website in which this particular model is given for three species.

You can also explore this SI and SIR epidemic models. There are many other models; but try to at least explore these two models. So, what we have explored is the third in this series of SI, SIR and SEIR.

So, thank you very much; we will look at how to solve differential equations with initial conditions using Laplace transform in the next class.