Computational Chemistry & Classical Molecular Dynamics

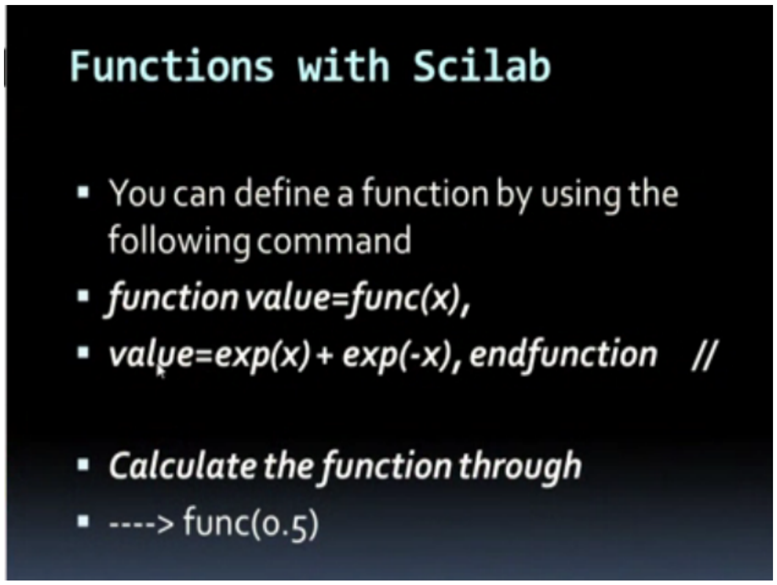
Prof. B. L. Tembe
Department of Chemistry
Indian Institute of Technology – Bombay

Lecture - 25
Scilab-3: Functions, Integrals, Differential Equations and Graphs

Hello and welcome to this lecture, which continues the Scilab presentation. We told in the earlier lecture that Scilab is a very powerful software, which can do all your numerical computations very, very effectively and this Scilab is being used on an international level for many, many important projects. So it is a very effective software and it costs nothing.

So you know, we should always, in academic communities, we should always go for those softwares, which are freely available, so that we have to save on our expenses. So last time, we considered several elementary operations, which can be done on Scilab. Of special interest was this definition of a function.

(Refer Slide Time: 01:03)



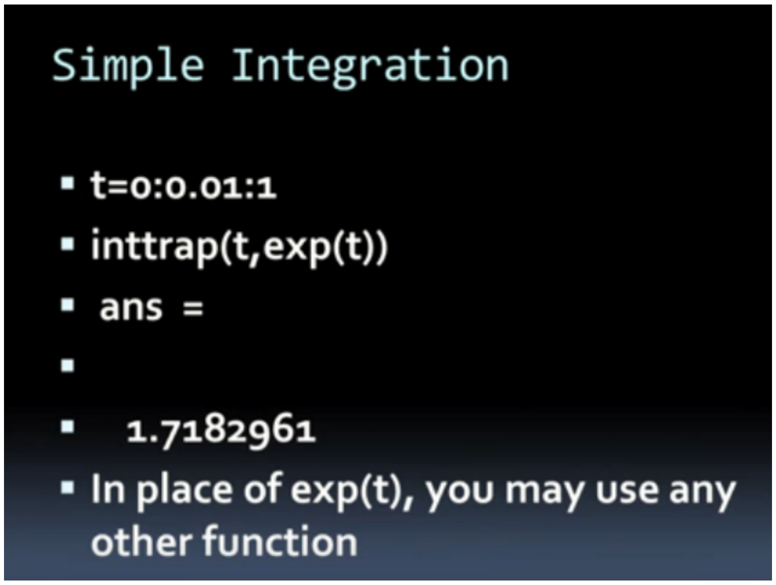
Functions with Scilab

- You can define a function by using the following command
- *function value=func(x),*
- *value=exp(x) + exp(-x), endfunction //*
- *Calculate the function through*
- ----> func(0.5)

In a Scilab, we can also define a function. So I want to define a function func(x). The first line is function value=function of x, so that is the first line and the second line gives meaning to that value, value=exp(x)+exp(-x), end function and double slash. Double slash is only whatever follows after the double slash is a comment card. Now we notice that in the function from the beginning function up to end function, there are many, many statements and they are separated by commas not semicolons, okay.

So this is how I define a function and once I define a function, I can calculate the value of that function at any value of x, x could be any value. This is how I will calculate function in the rest of the Scilab operation.

(Refer Slide Time: 02:01)



Simple Integration

- t=0:0.01:1
- inttrap(t,exp(t))
- ans =
-
- 1.7182961
- In place of exp(t), you may use any other function

After this, we also considered how to use trapezoidal rule. I discussed it in the last time. I will not repeat this part again. So you can define a variable t in the range between 0 and 1 in a spacing of 0.01. So this is like a do loop. So I define all the values of t, 101 values through this single, then using trapezoidal rule, I obtain an integral of exponential of t, t is already given at 101 points exp(t) is calculated at 101 points, t has 101 points and this function gives me the integration using trapezoidal rule.

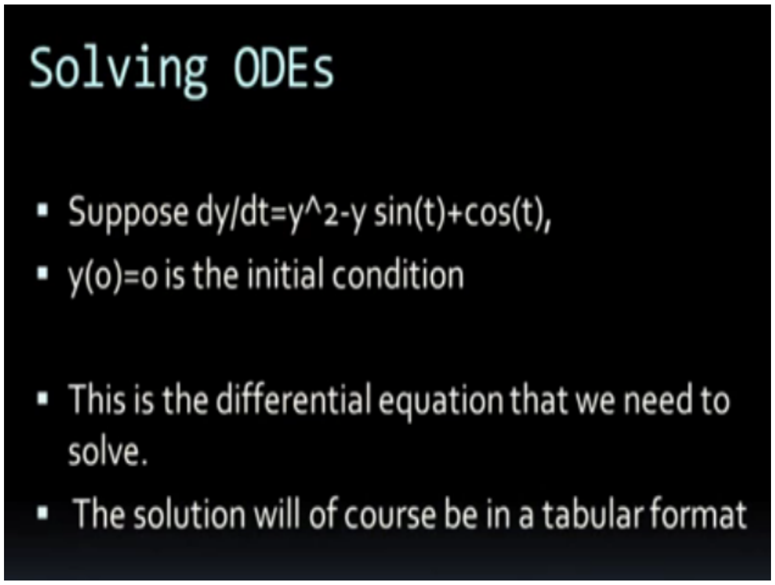
So the same thing can be done using a program and I have also discussed it in the last class. So what I want to do today, this integration also we discussed. In particular, I want to integrate now some function, say sin(x), so I want to integrate this function sin(x) from x, starting value is x, okay, sorry function is x, function is sin(x), variable is x, starting value is x0, x0 is the starting value, which I have given as 0 and x1 is an array, x1 is an array consisting of 0 to pi, spacing of 0.1.

So there are many, many points here from 0 to pi. In fact, there will be 31 points and I will be integrating sin(x) from the starting value of x0 to each one of these 31 points, so I will have the integral for those 31 values of the upper limit of integration. Lower limit is x0, upper limit is this x1. So you execute it and you will see that you will be able to integrate this function sin(x) in this range. So in place of sin(x), I could have any other function.

Suppose I have a very complicated function, which is defined through the function as I have discussed earlier. So I can define any complicated function. Then I have a variable x and I integrate that function between the starting point and ending point. So this is a very, very powerful way to integrate functions for which we do not know the analytical values of the integral. So this is how I do integration.

And since I can do the integration, then I can also solve differential equations. How will I solve differential equations?

(Refer Slide Time: 04:36)

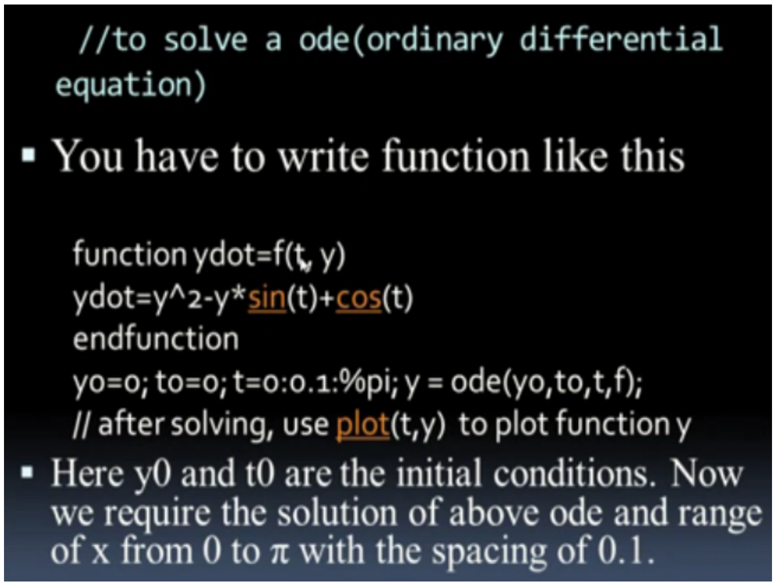


Solving ODEs

- Suppose dy/dt=y^2-y sin(t)+cos(t),
- y(o)=o is the initial condition
- This is the differential equation that we need to solve.
- The solution will of course be in a tabular format

So this is a method to solve differential equations using Scilab. So I have this differential equation dy/dt, okay. Let this equation be dy/dt=y square-y sin(t)+cos(t). So this is my differential equation, y0=0 is the initial condition. So I want to solve this differential equation.

(Refer Slide Time: 05:02)


//to solve a ode(ordinary differential equation)

- You have to write function like this

```
function ydot=f(t, y)
ydot=y^2-y*sin(t)+cos(t)
endfunction
yo=o; to=o; t=o:o.1:%pi; y = ode(yo,to,t,f);
// after solving, use plot(t,y) to plot function y
```

- Here y0 and t0 are the initial conditions. Now we require the solution of above ode and range of x from 0 to π with the spacing of 0.1.

So what are the commands for that solution. 1 would be, I again now define that y dot. Y dot is my function, function y dot=f(t,y). So this is my function. Then, y dot=y square-year-old sin(t) +cos(t). So this is my value of y dot end function. So this defines the function and now I want to solve the differential equation. So what is the command to solve the differential equation. So command is ODE. ODE is the command for differential equation, okay.

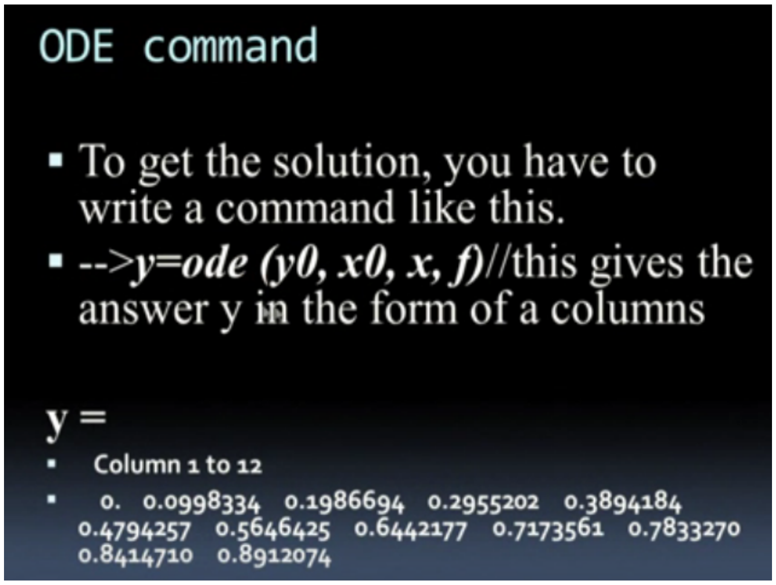
What are the variables that go into that ODE? ODE will require y0, t0, t and f, okay. So y0 is my initial value of y, t0 is the starting value of t, okay and what the values of t now, t will take on values from 0.1 up to value of pi. So this particular line t=0:0.1:%pi, %pi means, it is the values of pi in radiance. So this t is a variable that takes values from 0 to 3.1415 in increments of 0.1. So these are the values of t.

So I want to solve the differential equation, so that I have the solution y for all these values of t, okay. So that is my solution. So y=ODE y0, t0, t, f, f is the same function I defined earlier. So you will see that this is almost like a subroutine. Remember in Fortran, we used to call subroutine. When I call a subroutine, it takes the values of whatever variables there are and gives me the solution to the main program.

So you assume that your Scilab is like a main program that is running and all these functions and routines give you ways of calculating solutions to these equations. So this is to solve an ordinary differential equation. Now after you solve, you can also plot using Scilab. 1 of the most important advantages of Scilab is that whenever you solve a problem, you can plot it and see y as a function of x or y as a function of t.

So we will demonstrate it as we execute all these programs in Scilab. So this is our solution of a differential equation.

(Refer Slide Time: 07:47)



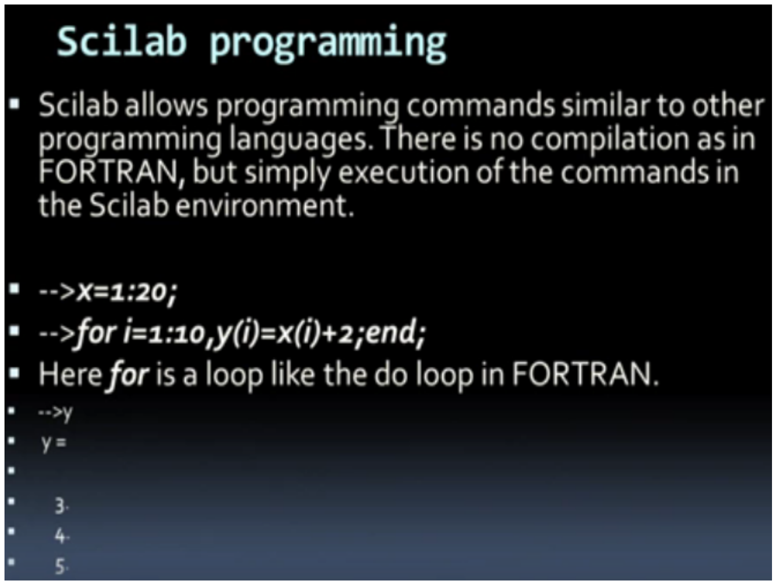
ODE command

- To get the solution, you have to write a command like this.
- -->*y=ode (y0, x0, x, f)*//this gives the answer y in the form of a columns

```
y =
▪   Column 1 to 12
▪    0.  0.0998334  0.1986694  0.2955202  0.3894184
   0.4794257  0.5646425  0.6442177  0.7173561  0.7833270
   0.8414710  0.8912074
```

So next we will, when you give this command y=ODE y0, x0, x, f, it gives the answer of y in the form of columns. So what are these columns. These are all the solutions of y for different values of x, okay. So 1 to 12, these are the solutions column 1 to 12.

(Refer Slide Time: 08:09)



Scilab programming

- Scilab allows programming commands similar to other programming languages. There is no compilation as in FORTRAN, but simply execution of the commands in the Scilab environment.
- -->x=1:20;
- -->*for i=1:10,y(i)=x(i)+2;end;*
- Here *for* is a loop like the do loop in FORTRAN.

```
-->y
y =

  3.
  4.
  5.
```

Then, they will have all up to full, all the columns will be there, okay. So now I have given only between 1 and 12, others you will see when you execute yourself. So after doing all this, the next thing I want to do a little bit is what is called Scilab programming. Remember in Fortran, we know how to do programs in Fortran because it is a structure to that language. Everything you want to do mathematically, there is an equivalent programming language to do that operation.

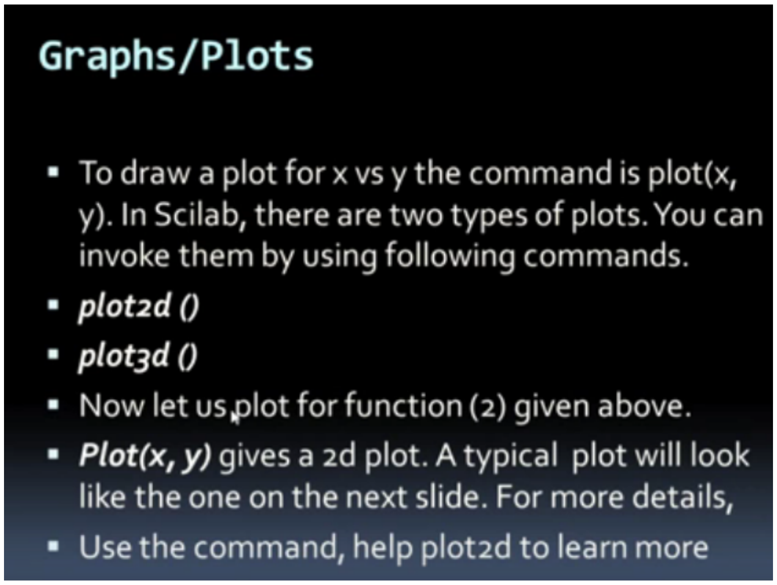
So in Fortran, in particular the do loop was do 10, i going from 1 to 10, then all the statements in the do loop will be executed until this i takes the last value of that index variable for the do loop.

Now, for example, here I have given x going from 1:20, so that means x will take these 20 values, okay. That is this is almost like a, it should have been x, okay, let us go to the next 1. I think there is something not correct in this. So there should be x:1:20, in which case it will take x from 1 to 20 with the spacing of 1, so this may also take on 1 to 20 in increments of 1. This could be the default value. Now this is how I will do a do-loop in a Scilab program.

How do I do this do loop? For i going from 1 to 10. So this is my do loop statement, yi=xi+2:;end. So this is the do loop. In an earlier statement, I have already defined x1 to 20. There will be 20, there will be 20, x will go from 1 to 20. So in my second loop, i goes from 1 to 10 and yi goes from xi+2. Each yi is xi+2. Now you see that this x was going from 1 to 20. So I calculate y for those values 1 to 10 and this is listing all the values.

I have listed a few values here, first 4 values. So this is how I will estimate a do loop and what I have done here y and enter. When I enter y, it will give all the 10 values of y, because y was only taking 10 values. So this is how I will execute and give the output for y. So this is how I will do some elementary program in Scilab.

(Refer Slide Time: 10:55)



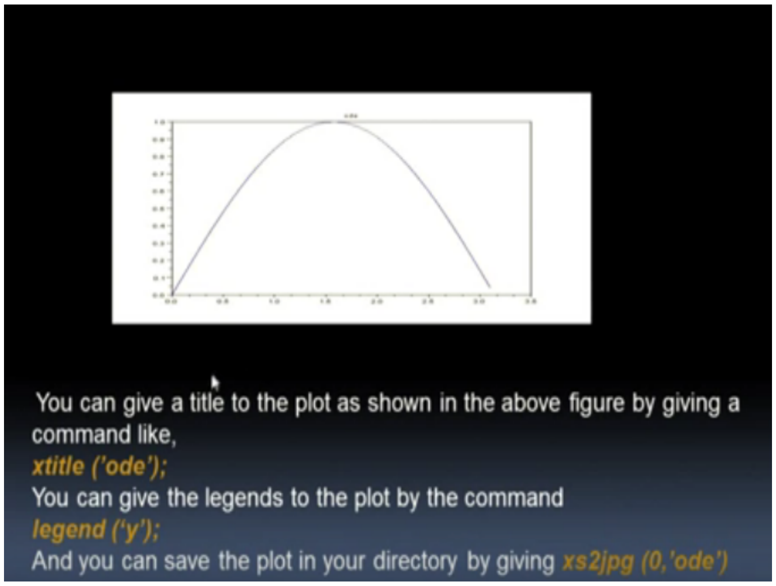
Graphs/Plots

- To draw a plot for x vs y the command is plot(x, y). In Scilab, there are two types of plots. You can invoke them by using following commands.
- ***plot2d ()***
- ***plot3d ()***
- Now let us plot for function (2) given above.
- ***Plot(x, y)*** gives a 2d plot. A typical plot will look like the one on the next slide. For more details,
- Use the command, help plot2d to learn more

So now, a very interesting thing, how will I plot using Scilab. So there are many, many options to plot in Scilab. In fact, it is extremely powerful. So there are different kinds of plots. There is a 2 dimensional plot. There is a 3 dimensional plot. So you can practice with several types of plots, okay. So let us take the simplest example. Normally, we need this 2D plots. What does a 2D plot do? You have a variable x along the x axis, you have a variable y along the y axis.

So the plotting command is very, very straight forward. Plot (x,y), it will give you a 2 dimensional plot, okay. Example of this plot is given in the next slide.

(Refer Slide Time: 11:44)



You can give a title to the plot as shown in the above figure by giving a command like,

xtitle ('ode');

You can give the legends to the plot by the command

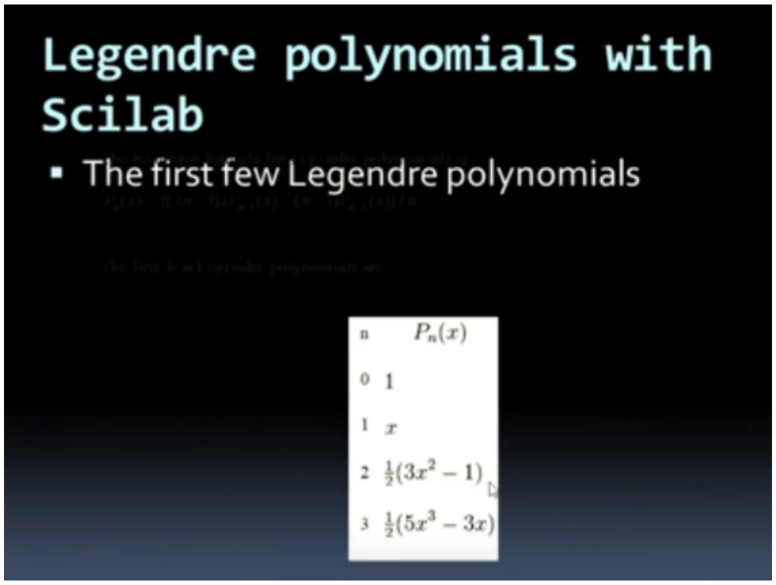
legend ('y');

And you can save the plot in your directory by giving *xs2jpg (0,'ode')*

So this plot, in fact what this plot does, it is the solution to that ordinary differential equation. Remember we solved an ODE. Once you solve an ODE, you can just plot the result in this manner. You can just plot the result, okay and once you plot, you also need to give labels to the x axis, and y axis. So there is a procedure for that x title=ODE. So this is the title I want to give on the x axis and this is the title I want to give on the y axis, okay.

And once you plot this graph, you can also save this graph into your directory by giving this command xs2jpg(0,ODE). So I can save these in my hard disk using this command, xs command. So this is how you will plot and save.

(Refer Slide Time: 12:42)



Legendre polynomials with Scilab

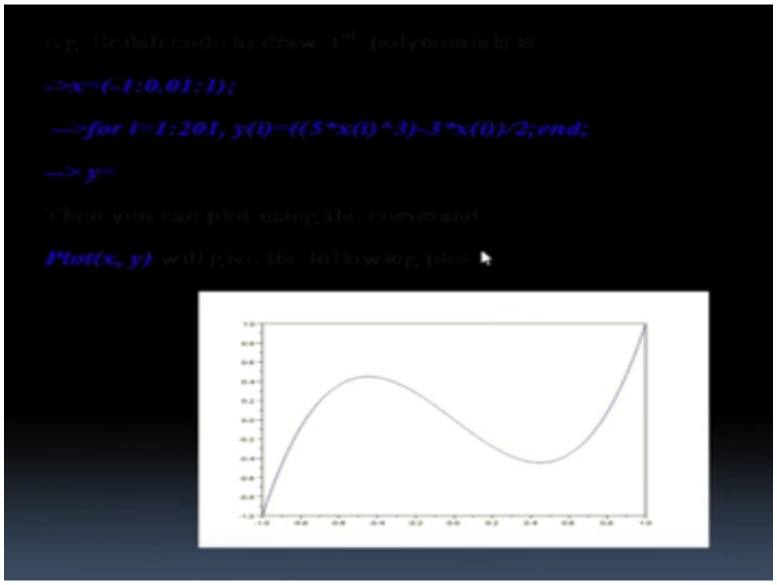
- The first few Legendre polynomials

n	$P_n(x)$
0	1
1	x
2	$\frac{1}{2}(3x^2 - 1)$
3	$\frac{1}{2}(5x^3 - 3x)$

Now the next thing I want to do. Remember in chemistry we study a lot of orbitals and the radial part of the orbital is given by a certain set of polynomials and angular plots are given by another sets of polynomials. Those angular plots are called Legendre polynomials and the Legendre polynomials p and x, okay. So n is the index. So I have first p0x is a constant function, p1x is just x, p2x is half of 3x square-1, p3x is 1/2*5x cube-3x.

These are nothing but, these are your shapes of orbitals. This is your p orbital. This is your d orbital. This is your f orbital. So similarly there are many, many functions now. So what I want to do, I want to calculate all these Legendre polynomials and plot the entire set. Plot the entire set, so that is what the next slide.

(Refer Slide Time: 13:50)



e.g. Scilab code to draw 3rd polynomials is:

```
-->x=(-1:0.01:1);
-->for i=1:201, y(i)=((5*x(i)^3)-3*x(i))/2;end;
--> y=
```

Then you can plot using the command

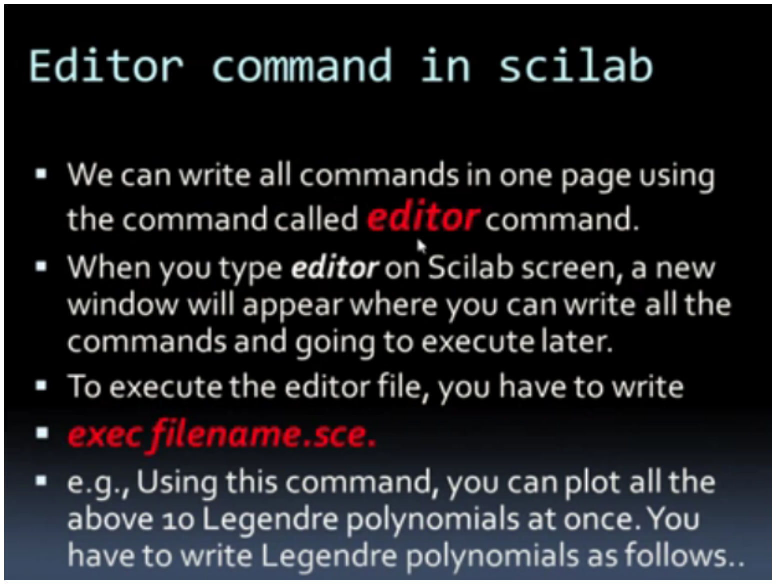
Plot(x, y) will give the following plot

So this is an example of a third polynomial. So let us say, I just want to plot the third polynomial. Here the visibility is not so good, okay. So I want to plot the value of x takes on the values -1:0.01:1, so that is the values of x or from -1 to 1 with a spacing of 0.01. So this is my do loop. So this will generate 201 values of x and once you generate 201 values of x, now we want to generate all the values of y. So what are the values of y now?

Go back to the slide, values of yr 1/2 of 3x square-1, so this is my function. So that is what this is doing, yi=, this is not 2x square-1, this is 5x cube-3x, so what it does? It calculates for those 201 values of x, yi=5*x cube-3*x whole thing inside a bracket/2:;end. So this is a do loop now. This is a do loop, which execute for 201 values of x and calculate the function yi. Now when I say plot xy, this will give the plot as shown below.

So this is the plot of that third legendre polynomial. This is how it will look like. Now once you plot one graph, our interest would be can I plot several graphs on the same sheet. Because many times when you have lots of calculations, you want to plot more than 1 graph on the same paper or on the same setting. So that is what is shown in the next slide, but before I come to that, I will also tell you about another command, which is called an editor command.

(Refer Slide Time: 15:51)



Editor command in scilab

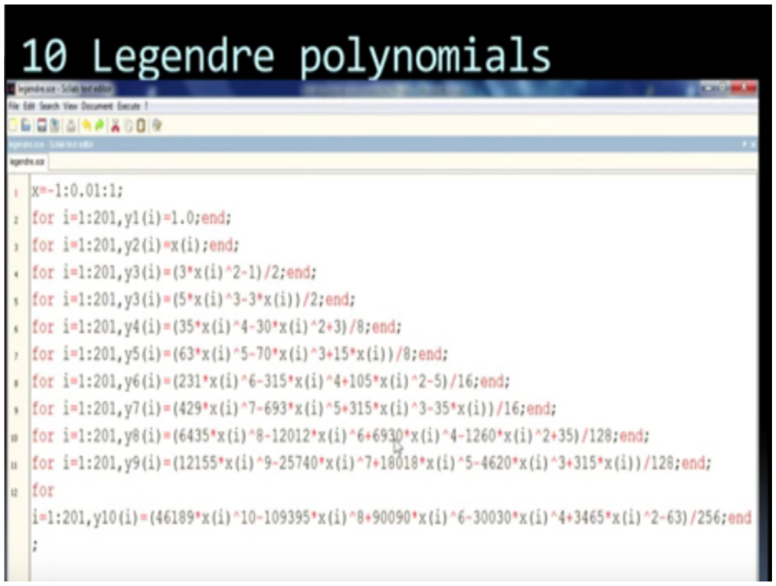
- We can write all commands in one page using the command called *editor* command.
- When you type ***editor*** on Scilab screen, a new window will appear where you can write all the commands and going to execute later.
- To execute the editor file, you have to write
- ***exec filename.sce.***
- e.g., Using this command, you can plot all the above 10 Legendre polynomials at once. You have to write Legendre polynomials as follows..

Now the use of this editor command, suppose you have a large number of statements, which you want to execute and you want to save it in a file and then execute all of them together. See for example, you may want to invert a matrix, you may want to calculate some functions, you may want to do many, many things and you want to save it in 1 file and execute. This is like you program in Fortran.

So in a program, you may want to do interpolation. You may want to do fitting, then you may want to do integration. So all those commands, you want to put in one file. So to do that I can use a command called editor. So the moment you type editor on a Scilab screen, a new window will appear. A new window will appear, where you can write all the commands that you are going to execute later. So this is like now a program inside a Scilab, okay.

So write all those commands using your editor and save it, okay and you save it when you save an editor, it will save it with an sce extension, okay. It will save it with an sce extension and when you want to execute that particular file, just type execute.filename.sce. So this is how you will execute all the lines that are saved using an editor command. So this is an executable file. So you can execute using a filename.sce.

So what I will do now, I will create using an editor, set of lines, so that is shown in the next slide.
(Refer Slide Time: 17:31)



10 Legendre polynomials

```
x=-1:0.01:1;
for i=1:201,y1(i)=1.0;end;
for i=1:201,y2(i)=x(i);end;
for i=1:201,y3(i)=(3*x(i)^2-1)/2;end;
for i=1:201,y3(i)=(5*x(i)^3-3*x(i))/2;end;
for i=1:201,y4(i)=(35*x(i)^4-30*x(i)^2+3)/8;end;
for i=1:201,y5(i)=(63*x(i)^5-70*x(i)^3+15*x(i))/8;end;
for i=1:201,y6(i)=(231*x(i)^6-315*x(i)^4+105*x(i)^2-5)/16;end;
for i=1:201,y7(i)=(429*x(i)^7-693*x(i)^5+315*x(i)^3-35*x(i))/16;end;
for i=1:201,y8(i)=(6435*x(i)^8-12012*x(i)^6+6930*x(i)^4-1260*x(i)^2+35)/128;end;
for i=1:201,y9(i)=(12155*x(i)^9-25740*x(i)^7+18018*x(i)^5-4620*x(i)^3+315*x(i))/128;end;
for
i=1:201,y10(i)=(46189*x(i)^10-109395*x(i)^8+90090*x(i)^6-30030*x(i)^4+3465*x(i)^2-63)/256;end
;
```

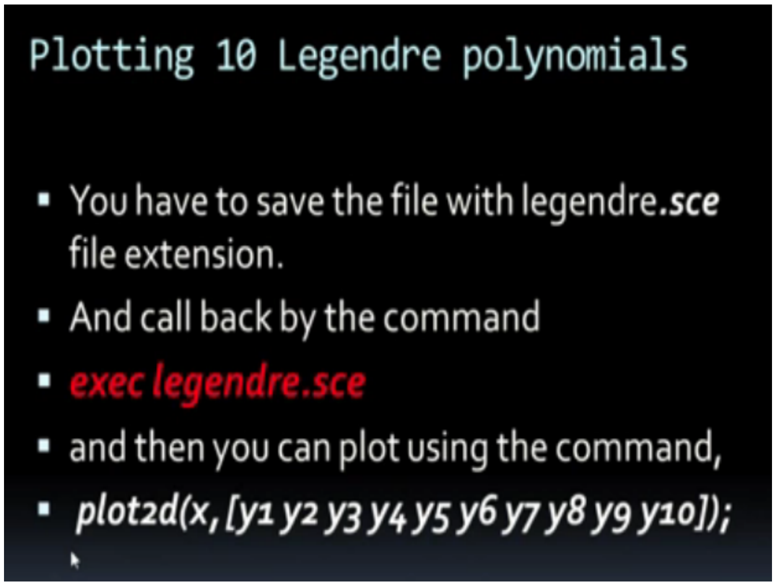
So look at this slide now. What this slide is showing, I am going to create a large number of Legendre polynomials and I want to plot all of them at the same time. So how do I create this? As I did earlier, my x is a variable that variable goes from -1:0.01:1; so that means now this is like a do loop. It takes all the values of x between -1 up to +1 with a spacing of 0.01, so there will be 201 values of x.

Now to determine all the Legendre polynomials, okay, to determine all the Legendre polynomials, I have a separate loop for each of the polynomial. For example, the first Legendre polynomial was a constant. So I execute this loop i going from 1:201 yi=1 end. What was my second polynomial. Second polynomial is nothing but the value of x. So for i going from 1:201, y2i=xi; end. So this is my second function. So similarly y3, this is my third function, okay.

So there is some error here, this should have been y4, this should have been y4, this should have been y5, this is y6, y7, y8, y9, y10. So there are these errors, but of course you can correct them, because you know that this is the first function, which is constant. Second function is x, third function is x square-1/2, fourth function is xi cube-3xi/2. This is my fifth function 6, 7, 8, 9, 10. So once I generate these functions, now all these functions have been generated for 201 values of the independent variable x.

Now I want to plot all of them, okay. I want to plot all of them so the command is given in this particular line.

(Refer Slide Time: 19:35)



Plotting 10 Legendre polynomials

- You have to save the file with legendre.***sce*** file extension.
- And call back by the command
- *exec legendre.sce*
- and then you can plot using the command,
- *plot2d(x, [y1 y2 y3 y4 y5 y6 y7 y8 y9 y10]);*

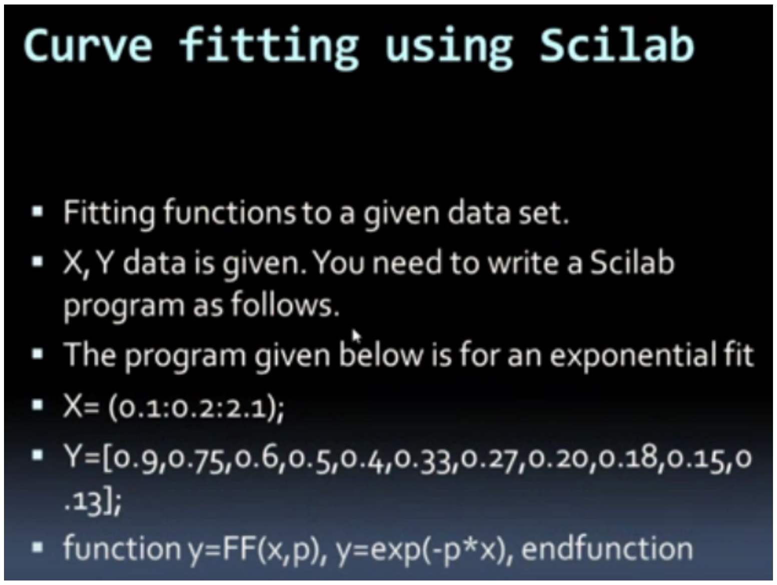
So I want to plot these 10 Legendre polynomials, okay. So but when I type all these things, I save it, remember I created using an editor command. I save it as a Legendre.sce. Now I will execute that entire file. So when I execute the file, it will generate me all the polynomials. Now I want to plot. I want to plot all of them and you know that the plot commands, if I wanted to just plot y1, I will say plot 2D (x, y1).

But since I have 10 of them, so I will plot it like this, plot 2D(x, this is a round bracket, this is a square bracket (x, [y1, y2, y3, y4 all the way up to y10]); so this will plot all the functions. So let us see whether I have this. I do not have it. We will execute it. We will execute it in reality when he do the demonstration part of the Scilab. So this is plotting function. Now there is another important function I want to spend time on.

It is called a fitting function. Remember when we did programming, we fitted a polynomial to a set of data points, okay. So we also fitted a straight line. So Scilab also gives a procedure to fit function to a given set of data. This is a slightly more involved set of statements than the ones so far. Because so far, you saw that everything was either a 1 line statement or a 2 line statement. Now this fitting function is little more involved.

So I shall illustrate all the statement first and then we will execute it in Scilab, okay. Now curve fitting using Scilab.

(Refer Slide Time: 21:32)



Curve fitting using Scilab

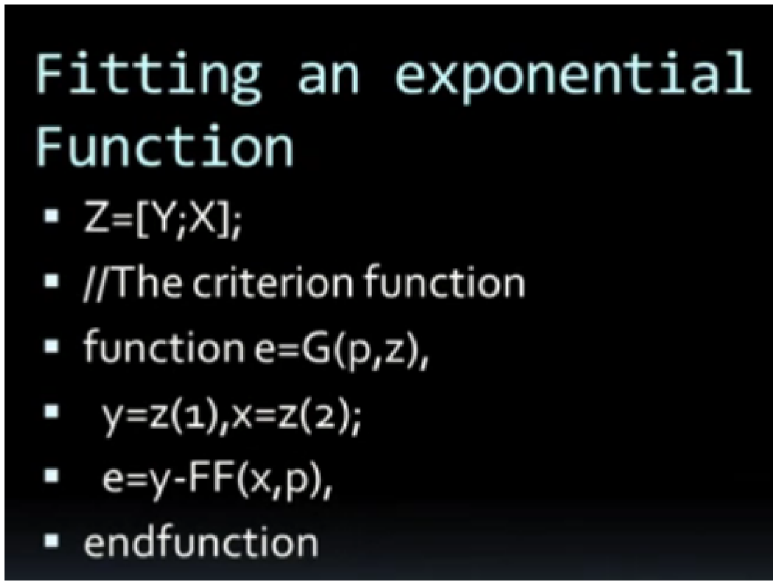
- Fitting functions to a given data set.
- X, Y data is given. You need to write a Scilab program as follows.
- The program given below is for an exponential fit
- X= (0.1:0.2:2.1);
- Y=[0.9,0.75,0.6,0.5,0.4,0.33,0.27,0.20,0.18,0.15,0.13];
- function y=FF(x,p), y=exp(-p*x), endfunction

I shall use it first for an exponential fit. Then I shall demonstrate it for a linear fit. So for curve fitting what is my starting point. My starting point is x and y, x is my independent variable and y is the dependent variable. This is the data that is given. So before I start my Scilab program, I need all the data points. So I take some simple set of data. So what is my data now for this exponential fit x is an independent variable, it goes from 0.1 starting value:0.2 this is the spacing:2.1, okay.

So starting point is 0.1, last value is 2.1, spacing is 0.2. So you will have as many values, 0.1, 0.3, 0.5. So the last value will be 2.1. For these independent values of x, I have dependent values now. What are my dependent values 0.9, 0.75, 0.6, how many do I have, 1, 2, 3, ,4, 5, 6, 7, 8, 9, 10, 11. I have 11 values of the dependent variable y. I have 11 values of the independent variable x and now this is my fitting program. So these are all the lines.

You just look at the lines first and then execute it. So how do I execute? After I define x and y, my next line is function y=xx, function y=ff(x,p). So this is definition of my function ff(x, p), y=exp(-p*x), because I want to fit my function as an exponential function. So it is –p*x end function. So the next line was definition of these functions y, okay.

(Refer Slide Time: 23:38)



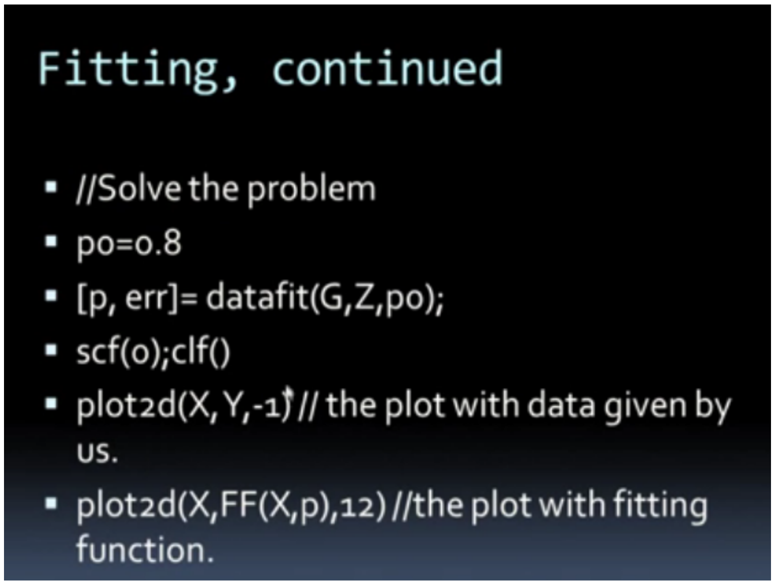
Fitting an exponential Function

- Z=[Y;X];
- //The criterion function
- function e=G(p,z),
- y=z(1),x=z(2);
- e=y-FF(x,p),
- endfunction

So then, we need some more statements, then next statement should be z=[y:x];. So now there is a criterion function. So what this does, this particular fitting function calculates a criterion function e and it does several operations and giving me the final result. Function e=G(p,z), then y=z1, x=z2; e=y-ff(x,p). So y was my original data ff(x,p) is the calculated fitted data. So what the program does, it will keep on iterating until this e, which is an error, which is reduced to a minimum, okay.

So the whole tic in fitting is you calculate the fitted function, compare with the data. If the error is too much, then I reduce the error, keep on iterating. So this particular set of lines does all that and finally it will give you fitted function, okay.

(Refer Slide Time: 24:59)



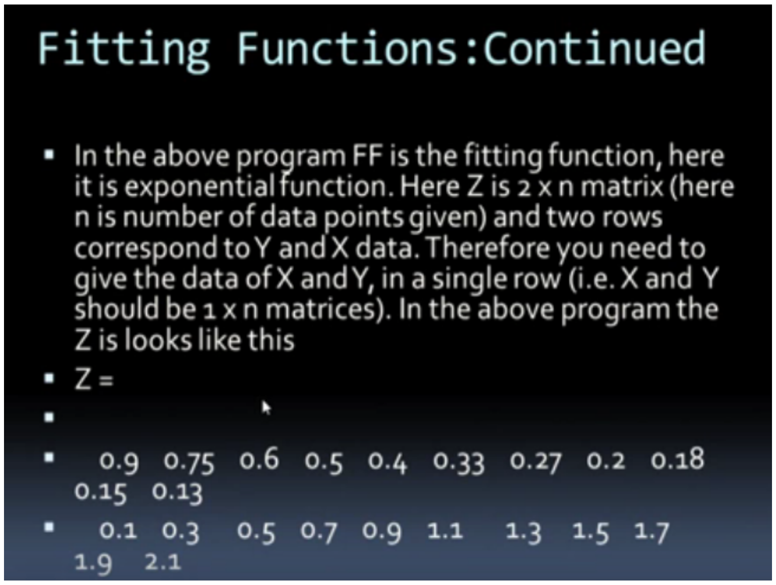
Fitting, continued

- //Solve the problem
- po=o.8
- [p, err]= datafit(G,Z,po);
- scf(o);clf()
- plot2d(X,Y,-1) // the plot with data given by us.
- plot2d(X,FF(X,p),12) //the plot with fitting function.

So after this set, so to solve the problem now, to solve the problem I need to give a value of p0 and p1. So p had 2 values, p0 is 0.8, okay and [p:err]=data fit(G, z, p0). So this is the function. The next function then scf=0, it will clear the screen, then plot 2D x,y,-1, so this is plotting the date, okay. Not next is xy,-1 plots you the data and plot 2D x, ff(x,p,12) it plots the fitting function. So the first 1, plot D is to plot x and y and the second 1 is to plot your x.

And the fitted function and the third line in that plot function tells you what kind of plot you have. Suppose, you want stars for your data points, you will give some number, you want some other symbols, so the third line, the third variable in the plot gives you, how I should identify the data, so we will execute it shortly.

(Refer Slide Time: 26:12)

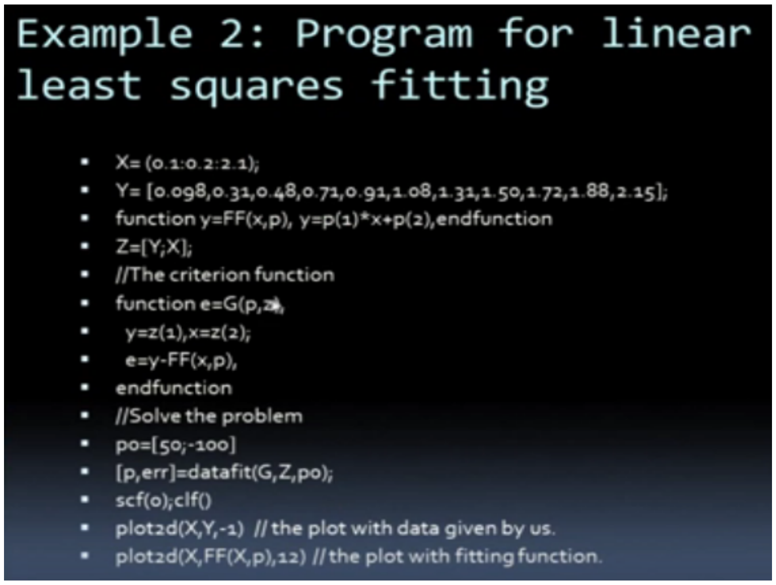


Fitting Functions:Continued

- In the above program FF is the fitting function, here it is exponential function. Here Z is 2 x n matrix (here n is number of data points given) and two rows correspond to Y and X data. Therefore you need to give the data of X and Y, in a single row (i.e. X and Y should be 1 x n matrices). In the above program the Z is looks like this
- Z =
-
- 0.9 0.75 0.6 0.5 0.4 0.33 0.27 0.2 0.18 0.15 0.13
- 0.1 0.3 0.5 0.7 0.9 1.1 1.3 1.5 1.7 1.9 2.1

So now, since I have done so this execution. So some more things here. So z, so just execute it. I think the best thing is to execute it and get a feel for it. It is a little involved function. So this is just the more details on what your fitting function is going to do, okay. So this tells you little more of the nature of x and y and the nature of this z, okay. this is how the z will look like. So this is how your criterion function works like, okay.

(Refer Slide Time: 26:48)



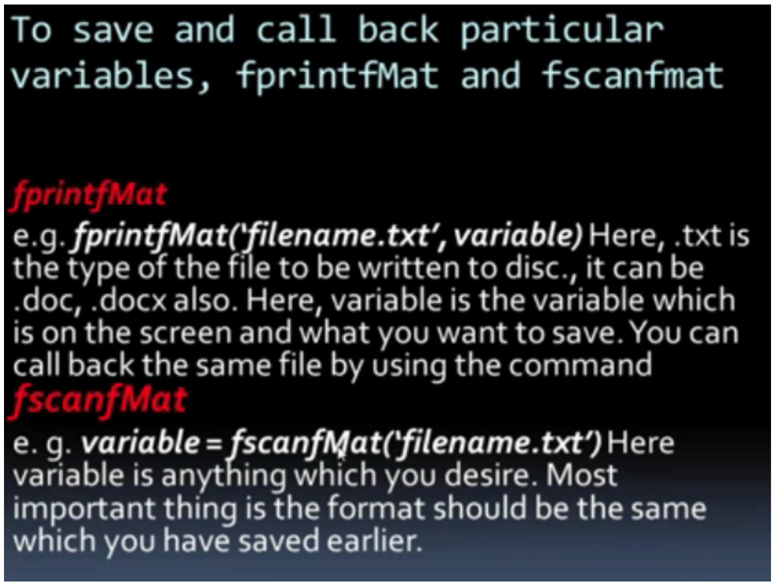
Example 2: Program for linear least squares fitting

```
X= (0.1:0.2:2.1);
Y= [0.098,0.31,0.48,0.71,0.91,1.08,1.31,1.50,1.72,1.88,2.15];
function y=FF(x,p), y=p(1)*x+p(2),endfunction
Z=[Y;X];
//The criterion function
function e=G(p,z),
  y=z(1),x=z(2);
  e=y-FF(x,p),
endfunction
//Solve the problem
po=[50;-100]
[p,err]=datafit(G,Z,po);
scf(0);clf()
plot2d(X,Y,-1)  // the plot with data given by us.
plot2d(X,FF(X,p),12) // the plot with fitting function.
```

So if I want to extend the same thing to a linear fit. So I have done an exponential fit. Now how will I do it for a linear fit, just see this. This is a little easier x=0.1:0.02:2.1. So this is my value of x. These are set of data points. Now I want to fit a linear function. When I fit a linear function what do I do, that y ff(x,p), this y function now is p1x+p2. This is a linear function. In the past, it was an exponential function. So end function.

Then z is the same, y:x criterion function e=G(p, z), y=z1, x=z2, e=y-ff(x,p) end function. How do I solve the problem, okay p0 I want to give an initial value of p0, remember this p is having 2 values p1 and p2, these are my initial values of p, then I define this p error, clear the screen and plot not only y, but also fitted function y, okay.

(Refer Slide Time: 27:57)



To save and call back particular variables, fprintfMat and fscanfmat

fprintfMat

e.g. ***fprintfMat('filename.txt', variable)*** Here, .txt is the type of the file to be written to disc., it can be .doc, .docx also. Here, variable is the variable which is on the screen and what you want to save. You can call back the same file by using the command

fscanfMat

e. g. ***variable = fscanfMat('filename.txt')*** Here variable is anything which you desire. Most important thing is the format should be the same which you have saved earlier.

So what I will do? I will conclude this lecture here. I have 2 more rather very simple statements. I will explain these and then we will execute the Scilab in the next lecture. So I will close here. Look at all these commands, look at all these slides. So that you will be familiar with all the commands that you need to use in Scilab. I will conclude here. Thank you.