Computational Chemistry & Classical Molecular Dynamics

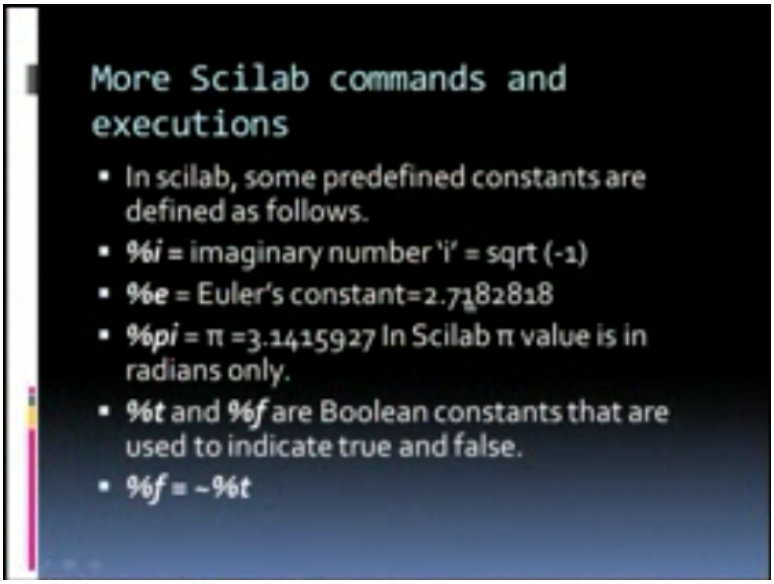
Prof. B. L. Tembe
Department of Chemistry
Indian Institute of Technology – Bombay

Lecture - 24
Scilab-2: Matrix Equations and Roots of Polynomials

Hello and welcome again. So in the last class, we started using Scilab software. So I told you how to do some very simple operations like adding and multiplying. Now we will look at some of the constants in Scilab. Several constants are there in mathematics, so they have specific representations in Scilab. So I will give this example for complex number, you know that you have x+iy, so that i is an imaginary number, square root of -1.

So that is represented as percentage i in Scilab. So percentage i means imagine a number, which is a square root of -1. The next 1 is this E, you know, a number you have this exponential function E to the x.

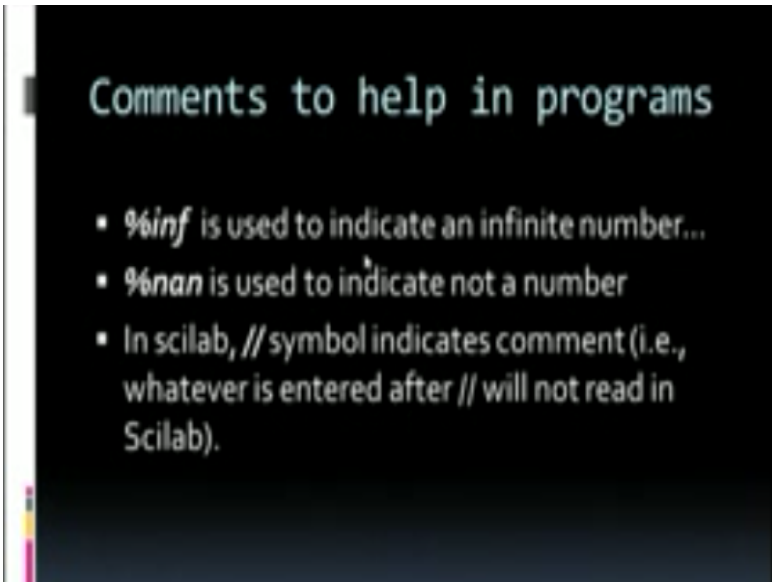
(Refer Slide Time: 01:04)



So that percentage E is an Euler's constant, okay 2.7182818, that is an Euler's constant. then, pi, you know, pi is 3.1415927. So in Scilab, the value is, it is represented as percentage pi, so it is always in radiance. Scilab uses radiance only. It does not use your degrees. Now percentage t and percentage f; they are Boolean constant, which are representation of something is either true or false. If something is true, it is represented as percentage t.

And if it is false, it is percentage f and percent f is, of course, negative of true, so it is tilde percentage t, so this is true and false variables. So these are called logical variables. In Fortran also, we have logical variables. We did not have too much occasion to use it.

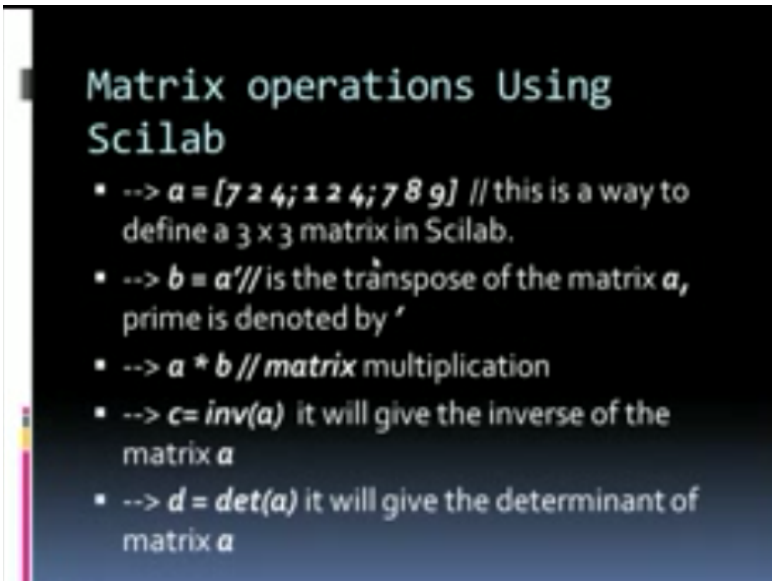
(Refer Slide Time: 02:03)



So next, many times what happens, you get a very large number in your calculation. So how will you represent? So I think many calculators also given infinity as a result, if the number is very large, so Scilab uses percent infinity and sometimes your result is not a number, so Scilab represents it as %nan, so %nan is used to indicate not a number, nan means not a number. I already mentioned earlier that this double slash, it represents a comment character.

Anything you want to write as a comment will be given as, in Fortran it was first column C followed by all your statement. In Scilab, it is 2 slashes and followed by the comment. So this will not be arrived or executed in Scilab, okay.

(Refer Slide Time: 02:58)



So now, we will start matrix operation. Scilab is especially very good for doing all the matrix operation. So in Fortran, remember you have to give lots of do loops to read matrices. In Scilab, there is no requirement, because it assumes that every variable is a matrix, unless you state it otherwise. Now suppose I want to read a 3x3 matrix, Scilab makes it very easy. A=[7 2 4; 1 2 4; 7 8 9;]. So this is a way to define a 3x3 matrix in a Scilab.

How do you define a matrix in a Scilab? You did not give any dimension statement 3x3. The very fact that there are 3 numbers here, each separated by space; next row, each number separated by; then the next row, then bracket ends. So this way, it understands it is a 3x3 matrix. So matrix operations as I mentioned are extremely easy in Scilab. So A=this, it assumes it is a 3x3 matrix. Now how will you give a 4x4 matrix 724 8.

Each of this, suppose I add an extra 8 here, here an extra add 3, then 7 8 9 some other number, then ; another 4 numbers, so then it will be a 4x4 matrix. I do not have to declare the array depending on how many numbers are written between the ; and the square brackets, it assumes the, it gets the size of the matrix through the data itself. So now, suppose I want the transpose of this matrix.

Remember in Fortran, if you want to transpose, you will have to define each element of the transpose matrix as the, suppose I want aij is my element, the element of the transpose will be

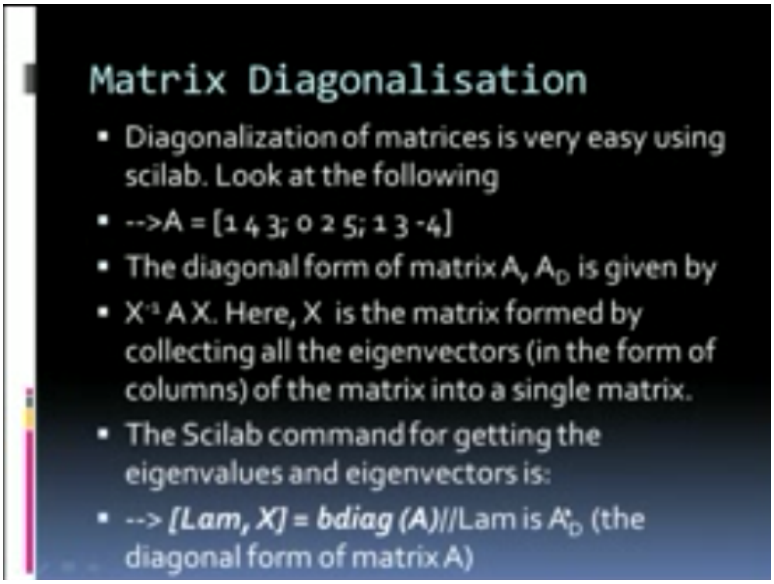
aji. So I have to define a new matrix and define every element of the matrix. So Scilab, it is very simple. B=A/so that particular prime, A prime is the transpose. So transpose is again very simple, one line B=A/. This is how you get a transpose.

Now how do I do matrix multiplication. It is so trivial. Suppose I say A*B, the moment I say A*B, the product will be displayed on your screen. The product will be displayed on your screen, but suppose I want to write the product into some other matrix, so I can do it like this C=, if I want the product to be written in some other matrix, I will say D=A*B. So the matrix D will be the product of A and B. Now we found the inverse of the matrix A, C=inverse A.

It will give the inverse of the matrix A. Now you see what my problem here, okay C=inverse A and if I say it will give the inverse, it will give an error because there are no slashes here. So I should have put 2 slashes here, because this is a comment, C=inverse A //, it will give the inverse of the matrix, that would be a comment. Now if I want the determinant D= determinant as I say, it will give the determinant. So what I will do.

I will illustrate all these commands now and will execute most of them, okay. So this is for the matrix operations.

(Refer Slide Time: 06:40)



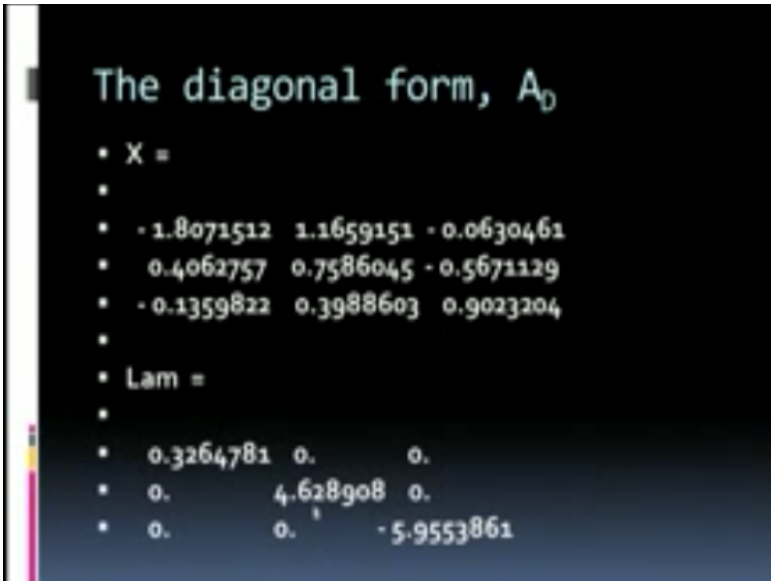
So remember to get the inverse matrix you have to write an entire DOS elimination program, which took several hours. So in Scilab it is very, very simple, just 1 line statement. So is it for determinant. Now let us look at matrix diagonalization. For matrix diagonalization, again it is a very straight forward process. So I will define the matrix. How will I define the matrix A=[1 4 3; 0 2 5; 1 3 -4] that means it is a 3x3 matrix.

So this A is a 3x3 matrix, so what is the process of diagonalization. The process of diagonalization is from the matrix A, I want to get the diagonal form, AD is the diagonal form. What is the formula for the diagonal form? It is x inverse Ax. What is x? x is the matrix formed by the collecting all the Eigen vectors in the form of columns or the matrix into a single column. That is I take all the Eigen vectors of A, arrange them in a column form.

If I take all of them as normalized Eigen vector, that is each vector is normalized. So x is a matrix of Eigen vectors, x inverse Ax will be nothing but AD, so that is my diagonal form. So the Scilab command for getting the Eigen values and Eigen vector is look at this command [Lam,X]=B diagonal * (A)//. So this is my command Lam,X=B diagonal A, so here Lam is lambda d, Lam is nothing but AD that is the diagonal form of the matrix.

And your x is the matrix of Eigen vectors. So with this 1 straight command, I get the entire diagonalization process that is illustrated in the next slide.

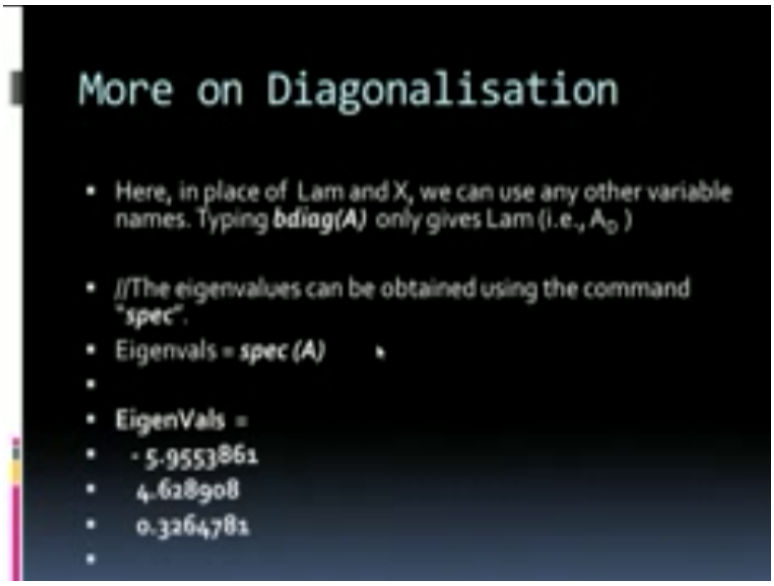
(Refer Slide Time: 08:44)



So once I execute that particular command, so it gives x=3 columns here. These 3 columns are the Eigen vectors of the matrix, then lambda is the Eigen value. Now all the Eigen values, it is writing in a diagonal form. So these are diagonal matrix, which consists of all the diagonal elements of the Eigen values and x is your matrix of Eigen vectors. Now here, instead of Lam,x I can call it by any other name.

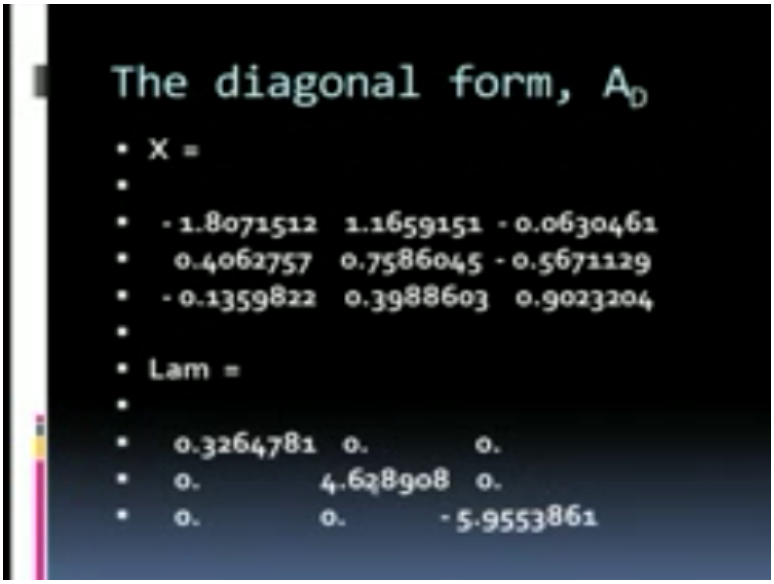
These are just variables, Lam and x are variables. Instead of lam, I could just call it WW in place of x, I could call it y, okay. A is the name of the matrix. Instead of A, I could have it any other name, but this B diagonal is the command. B diagonal is the command, the results of B diagonal is lam, x. These could have some other variable names that is completely alright. So this is how I shall execute the diagonalization in Scilab again extremely easy. Now let us see what is the next thing.

(Refer Slide Time: 09:52)



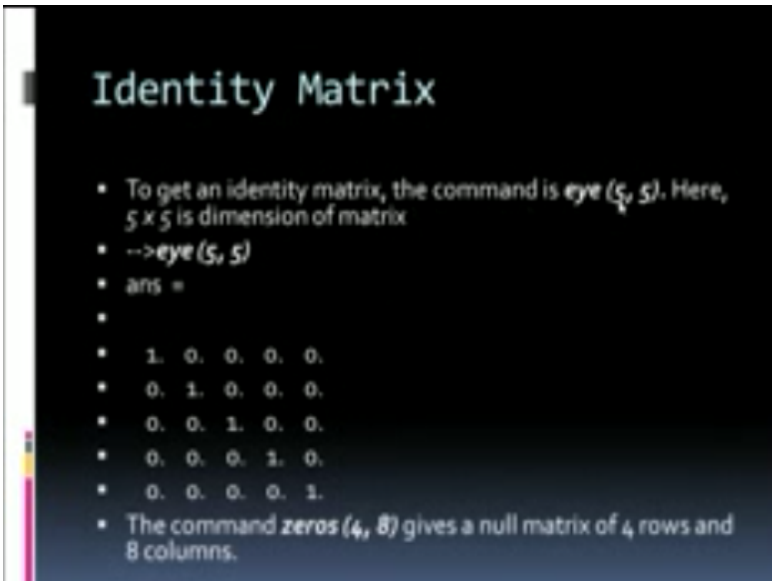
So instead of getting all the Eigen values and Eigen vectors, suppose I just want the Eigen values, so when I type B diagonal A, I have got both lambda as well as the Eigen values. Now if I just say Eigen values=Spec*(A). So it gives me only the Eigen values. So the Eigen values were 5.9, 4.6, 3.2, let us check. These were the same as the earlier Eigen values, except here they are in some order whereas in the next 1.

(Refer Slide Time: 10:31)



I have given in the order of the largest value first. Next value largest in magnitude, next largest in magnitude and third largest in magnitude. So the Eigen values are given here through the Spec. Spec command gives me the Eigen values of the matrix A, okay.

(Refer Slide Time: 10:49)

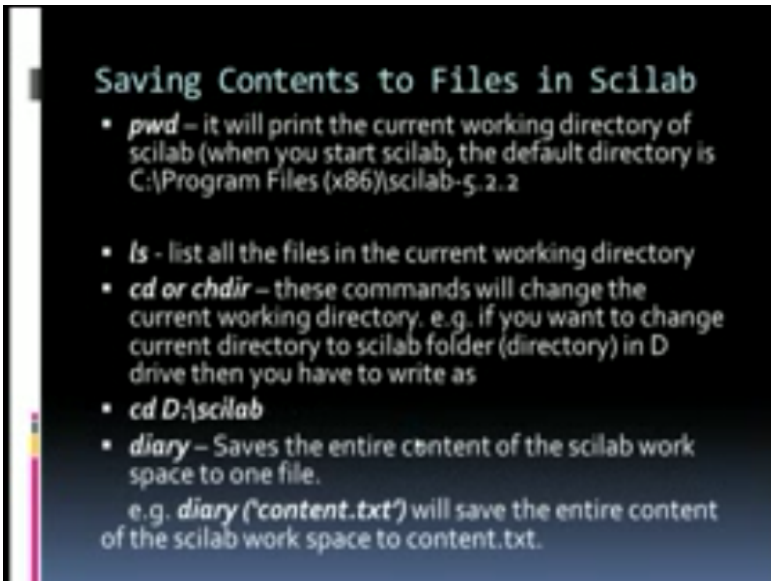


So then, suppose I want to define an identity matrix, to define an identity matrix, the command eye 5,5 okay. So why do I say 5,5 I want a 5x5 identity matrix, so the moment I say eye 5,5 the answer is this 5x5 matrix, okay 1 0 0 0 0, so this is the first row. Second row, the middle, the second diagonal value is 1, third diagonal is 1, fourth diagonal is 1, fifth diagonal is 1. So this is the 5x5 identity matrix. So the moment I say eye 5,5 it gave this as an answer.

Now instead of writing on the screen, suppose I want to assign some other matrix, with this particular identity values, so I will say, let us say W=eye 5,5. So what it will do? W will be my identity matrix of 5,5. It will not write anything on the screen. Here it is writing on the screen because I said eye 5,5 enter. So it will give me all these results. Now suppose I want a null matrix of 4 rows and 8 columns, I will just say 0s 4,8.

So what will this command do? It will have a matrix. This is not a square matrix. It is a rectangular matrix, 4 rows and 8 columns. So it will give 4 rows with 8 0s in each of the rows. So this is the 0 matrix.

(Refer Slide Time: 12:27)



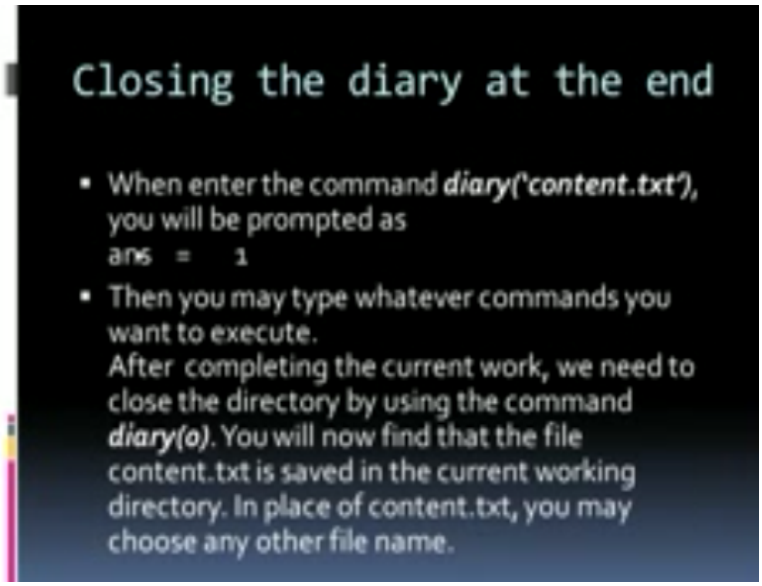
So now, let us see some other commands as well in Scilab. Many of the commands in Scilab also are similar to the Unix commands. Now suppose I want to know the present working directory in the Scilab, I have downloaded Scilab, okay. When you download the Scilab, it may put in particular directory, so if you want the present working directory, it will print the current working directory of Scilab.

So when you started the Scilab, if the default directories C:/Program Files/Scilab 5.2.2, so it will show this as the present working directory. Ls, just like in Unix it will list all the files in that present working directory. Now you can always change directory using cd or chdir. Both these are commands to change the working directory from 1 to another. Suppose you want to change the directory, suppose your Scilab is in D drive, then you want to say cd D:/Scilab, so it will go to the directory Scilab in the D directory.

Now another useful thing when you execute commands in Scilab, there will be many, many commands you are going to execute and you will not remember the entire list of commands you have given, because the screen keeps changing all the time. So you can save all the commands using what is called a diary command. So this diary command, it saves the entire commands of the Scilab work into one particular file.

So how do I go about doing this? When you start say diary (content.txt/), okay. So once you give this command, it will start the diary. So whatever commands you will execute on that day, will all be saved in content.txt and once you finish your work, then you will say okay.

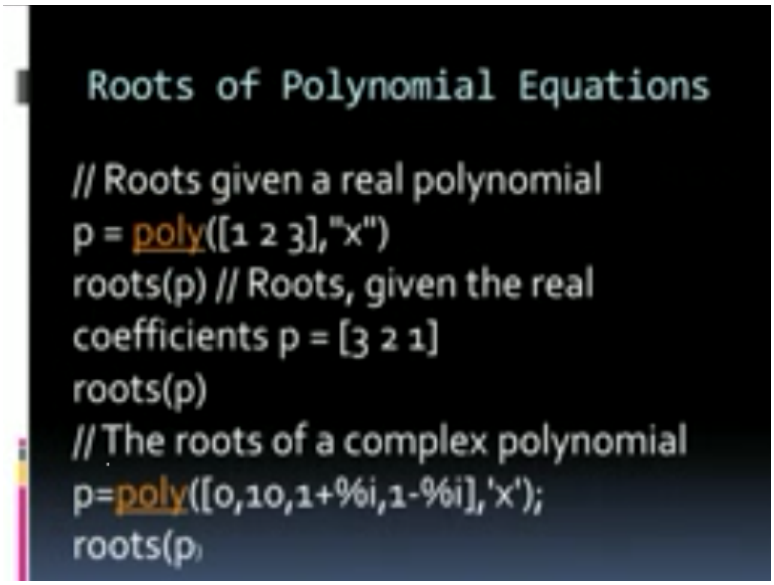
(Refer Slide Time: 14:29)



Now let us see when you enter the command diary (‘content.txt) you will be prompted. You will get answer=1. That means it has started the diary. Then, you type whatever commands you wanted, you type to execute. You type all the commands. After completing the current work, now you want to close, you need to close the diary, so what do you do, you use the command diary 0. Once you give command diary 0, it will save all the contents in this file content.txt.

So next time when you want to see what all commands you have used, you can edit, open this file content.txt, it will have all the commands that you used on that particular day. instead of content.txt, you can give any other name. You can give, for example, date. You give the date, instead of content, so many contents will be there, so one way would be, you just give the date say today is say 28th February, so we will 28th Feb.txt. So it will save all the commands of today into 28 Feb.

(Refer Slide Time: 15:42)



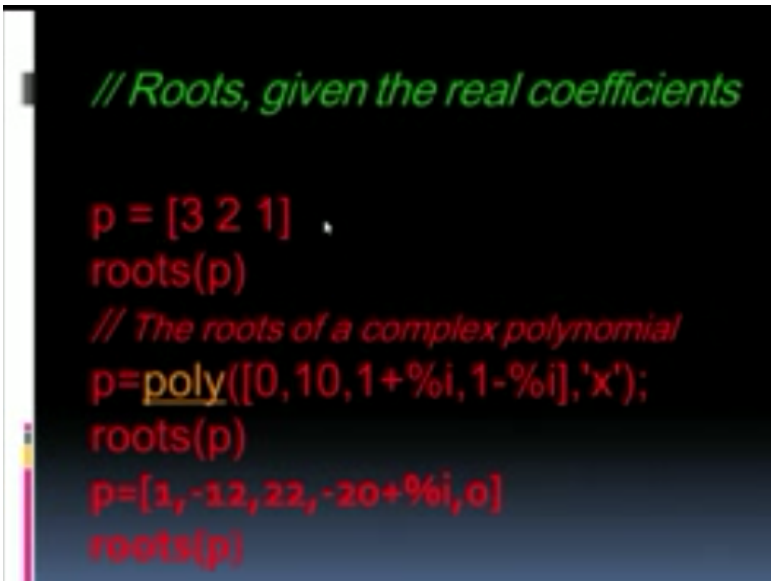
Now the next thing, we want to do is to use these polynomials. Remember, polynomials are nothing, but functions where variable x is raised to large number of powers, okay. So now, suppose I want to find a root of a particular polynomial, okay when I want to give the roots of a polynomial there are 2 different ways of doing it, okay. Do not look at the first 1 right now. The first command really generates a polynomial. The first command generates a polynomial.

So let us now go to the first 1, let us go to the second command, coefficients P=3, 2, 1. So what this does it generates a polynomial, okay. So this coefficient knows that that polynomial has 3 particular terms. So when it has 3 particular terms, it understands that it is 3x square+2x+1, so that is what is your polynomial, then you can find the roots of that polynomial, okay. So this particular set of statements P=3, 2, 1 it generates a polynomial of 3 terms.

The highest power will be the third term, so 3x square+2x+1. Suppose I want a third order polynomial, I will say P=3, 2 ,1, 4. If there are 4 terms, the first term will be the x cube term. So you give the coefficients of the polynomial, when we say roots P, it will give you all the roots of the polynomial, okay. Now suppose, you want to generate a polynomial, okay, this P=poly, it generates a polynomial. So P=poly generates a polynomial.

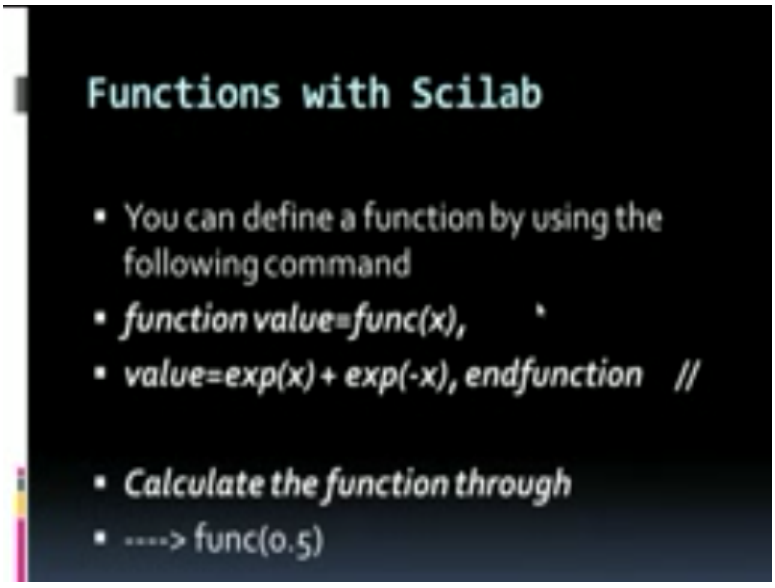
What this will do? It will generate a polynomial x, whose roots are 1, 2, 3. So you execute both the poly command as well as coefficients command to solve polynomial equations.

(Refer Slide Time: 17:48)



So this is the same thing here, I gave the coefficient of the polynomial, find out the roots, okay. Now in this case P=poly, all these commands, so these are the roots of my complex polynomial. So these are the roots of a complex polynomial, so from that I can generate a polynomial. So but I think these polynomial coefficients, what was my earlier 1, okay. Giving the coefficients and finding roots is a much better option. So use these commands P=3, 2, 1 and roots P to find roots of equations.

(Refer Slide Time: 18:27)



So next, now we will slowly go to slightly more advanced topics now. So suppose I want to generate functions in a Scilab, just as in a Fortran, you have generated a function. I can also

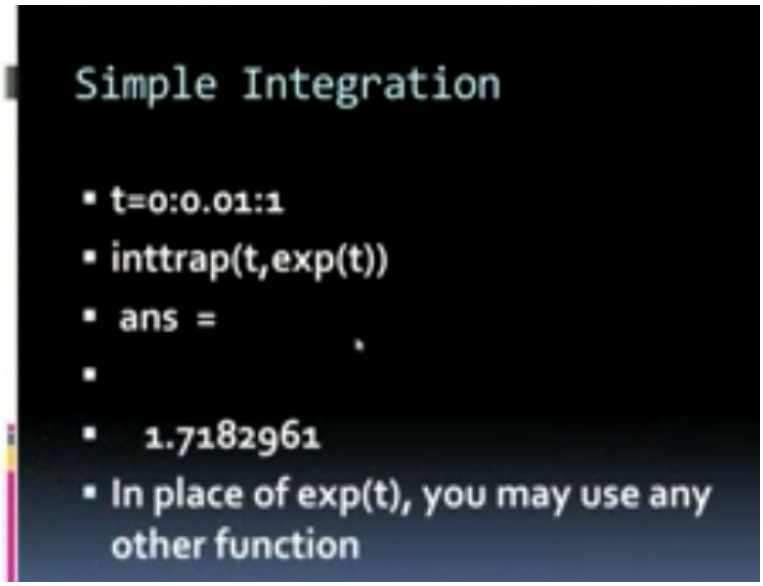
create a function in the Scilab. So how do I do that? You define a function by using the following command, function value=function x and then value=ex(x)+ex(-f), end function. So these 2 lines together define a function. So the first line says, I want to declare a function.

I want to declare a function called function x and that function x will be given some value and the second line says value=ex(x)+ex(-f), end function. So function and end function whatever is given between these 2 is the function definition. So I define a function in this manner. So once I define a function, suppose I want to calculate function x=0.5. So when x=0.5 I type func(0.5) enter, then it will calculate this function value when x=0.5.

So suppose I want the function at x=1, I will say func=1. So this way I define a function and estimate the values of the function for all values of x given in the bracket. This is how a function is defined, okay. Now your next question would be, can I do subroutines in Scilab in principle, yes, but the commands will be far more involved. So before I go to other uses, we should know how to program in Scilab, you know.

In Scilab, I can also do programs. So there will be commands, so you can treat this function as the program lines in a Scilab.

(Refer Slide Time: 20:27)

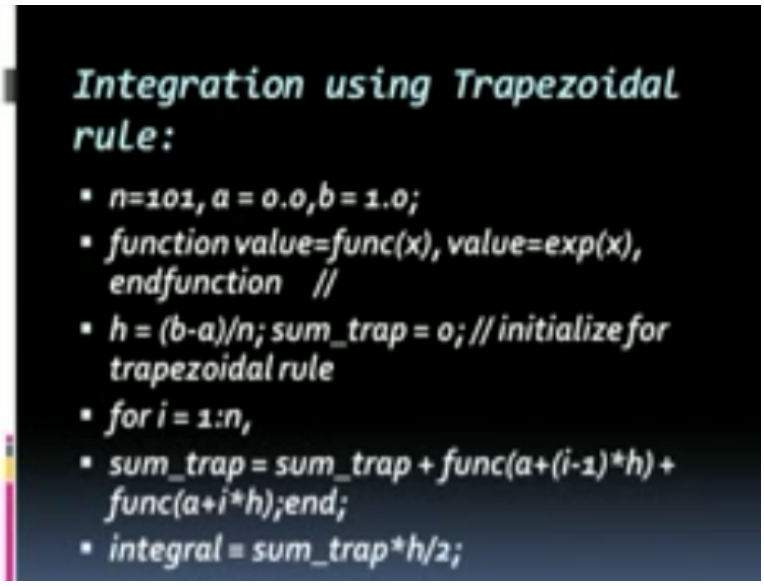


So now suppose I want to do an integration using Scilab. So whatever numerical methods we used in our earlier part of the course, like interpolation, integration, matrix operation we have seen, polynomial you have seen, now integration, differential equation, all these you can also do in Scilab. So I will illustrate now how to integrate. Integration is also quite simple. So now when I want to integrate, I have a variable t.

It takes on values from 0 up to 1. So t takes on values between 0 and 1, spacing is 0.01, okay. So how do I define that. This is like a do loop. Do something from 0 to 1, spacing 0.01, so 0:0.01:1. That means t takes on all the values between 0 and 1 with the spacing of 0.01. So what will be the values, 0, 0.01, 0.02, 0.03, okay. It will have all the 101 values between 0 and 1. Now I want to integrate this using trapezoidal rule, I will say inttrap(t, exp(t)). So you will get the answer.

So this integration using trapezoidal rule, just 1 particular line integrates, inttrap (t, exp(t)) you will get a trapezoidal rule. This is where I get the answer. So in place of exponential t, I can use any other function. So now let us see how to do the same integration in Scilab using the programming statement.

(Refer Slide Time: 22:08)



So the next 1, the next 1 I am going to do trapezoidal rule using some programming statements, okay. So how do I do n=101, these are data points. a=0, that is my initial value, b=1, that is end point. Now I will define a function. How do I define a function? Function value=func(x),

value=exp x, end function//, okay. So this is how I have defined a function, okay. Now, next I will define h, which is the spacing. What was the spacing here?

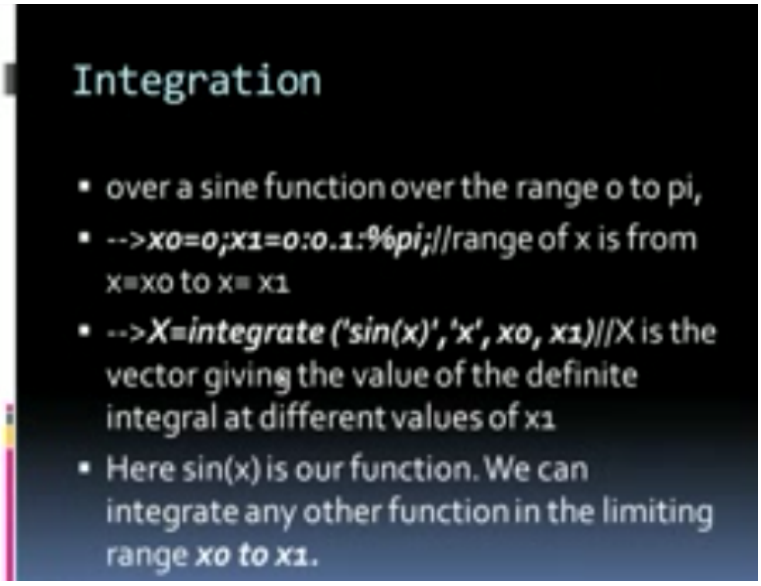
It was 0.01, so b-a/n, okay. So n I have; I should define n here, n is already defined as 101, so this is the spacing between the points. So next is sum trap=0. So this is my initial value. I am going to write a program now. I define the initial value and these are comments. What is the comment? Initialize for trapezoidal rule, okay. So I initialize, then I will use a do loop. How do I use a do loop? For i going from 1 to n, okay. So this is the beginning of my do loop.

So this is like a do i going from 1 to n in Fortran, so I am using the for statement. In C++ and other languages, in place of a do, they have a for statement. So this is the beginning of a do loop for i going from 1, n, then I want to put all the sum of the function into this sum_trap. Sum_trap=sum_trap+func(a+(i-1)*h). So what does this do? My do loop goes from 1 to n.

So the first value of the function is the value of a, the initial point, second value when i=1, it is i-1 is 0, so function of a, that is the first value, next time when I go to the do loop i=2, so 2-1 is 1, so this h, next value of the function is a+h, the third value of the function is a+2h, then a+4h, all the way up to n=101. The last value will be a+100*h and 101, 100*h, so this will be the last value. So I add all the last values and how do I end the do loop ; end.

So this is how the do loop ends. So when the do loop ended, I have added all the values of the function. So to get the integration using the trapezoidal rule, integral=sum_trap, this was the sum*h/2. So this is my integral using the trapezoidal rule. So what I have done through this, I have also illustrated how to program using Scilab. It is very similar Fortran except your do loop in place of do, we have a for statement, functions are defined slightly differently and very close similarity with the Fortran as well.

(Refer Slide Time: 25:13)



So now before I conclude, I will do a small integration of sine x. So what I want to do, I want to integrate a sin function over the range from 0 to pi. So x0=0, x1 goes from 0 spacing 0.1 up to %pi is pi. So x1 is a variable which goes from 0 to pi, x0 is the starting point, okay. So my X integrate sin x, x, x0, x1, okay. So my x0 is my initial value, x1 is a vector, x1 is a vector, which goes from 0.1 up to pi and my x is also a vector.

So I am going to integrate sin function from x0 to x1, so there will be x1 is a vector, whatever 101 parts, so I will get integral of sin(x) for every range in that integration. So 0-x1, sin(x) dx, so the print out will be a vector x and 101 values of the integral of sin(x) between 0 up to the upper value of the function. So sin(x) is our function, so you can integrate this way. You can also integrate any other function in place of sin(x).

So this is how you will do integration just by integration command. So in place of sin(x), you can use any other function. So I will conclude this lecture today and later tomorrow, I will start solving differential equation using Scilab. Now only we will solve differential equation, we will solve many other problems using Scilab and I will also demonstrate the use of this in the next lecture. We will conclude here. Thank you.