Computational Chemistry & Classical Molecular Dynamics

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

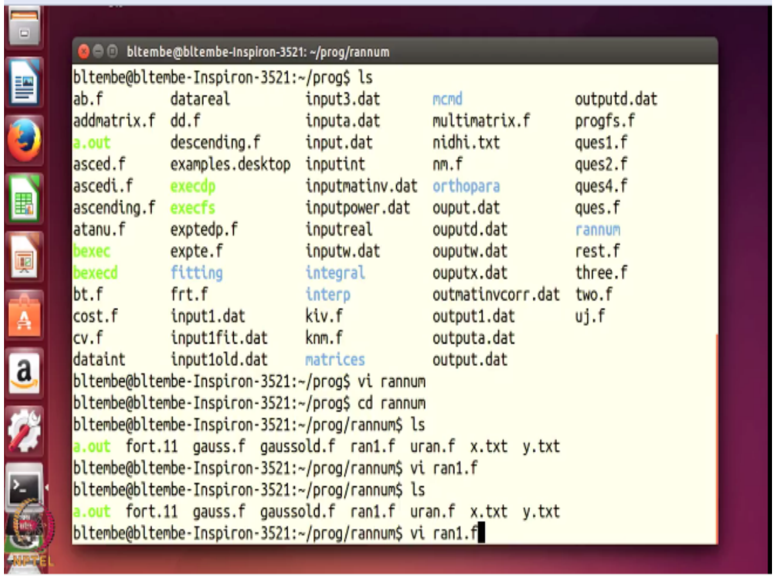
Lecture - 23
Practical Session on Programming 3: Random Numbers, Simpson's Rule and Introduction to Scilab

What we will do today, we will execute the random number programs and the Simpson's rule programs and then start introduction to Scilab. So in the last class we did Matrix inversion and we use the matrix inversion to compute the coefficients of a polynomial fit and there we use the matrix inversion is a subroutine. So in this lecture also we will have the use of several subroutines and it will also acquaint you with a further use subroutines.

In particular, in a subroutine the dimensions have to be variable. Because one main program may have one dimension and another main program may have a different dimension and you want to link the same subroutine to different programs. So different main programs will have different dimensions so the subroutine should be flexible enough to adjust two different dimensions, so that is what I will show in the Simpson's rule program.

First, we will consider the random number program. So I will move to the random number directory.

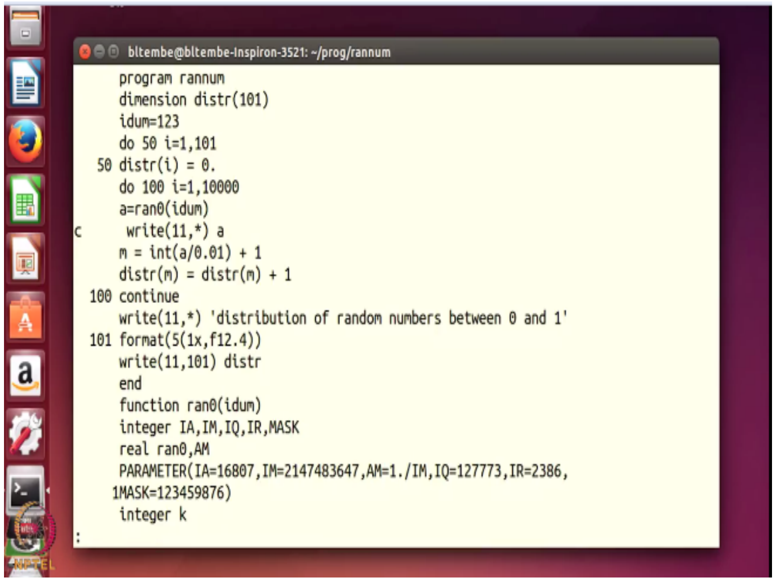
(Refer Slide Time: 01:20)



```
bltembe@bltembe-Inspiron-3521:~/prog$ ls
ab.f         datareal        input3.dat      mcmd              outputd.dat
addmatrix.f  dd.f            inputa.dat      multimatrix.f     progfs.f
a.out        descending.f    input.dat       nidhi.txt         ques1.f
asced.f      examples.desktop inputint       nm.f              ques2.f
ascedi.f     execdp          inputmatinv.dat orthopara         ques4.f
ascending.f  execfs          inputpower.dat  ouput.dat         ques.f
atanu.f      exptedp.f       inputreal       ouputd.dat        rannum
bexec        expte.f         inputw.dat      ouputw.dat        rest.f
bexecd       fitting         integral        ouputx.dat        three.f
bt.f         frt.f           interp          outmatinvcorr.dat two.f
cost.f       input1.dat      kiv.f           output1.dat       uj.f
cv.f         input1fit.dat   knm.f           outputa.dat
dataint      input1old.dat   matrices        output.dat
bltembe@bltembe-Inspiron-3521:~/prog$ vi rannum
bltembe@bltembe-Inspiron-3521:~/prog$ cd rannum
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ ls
a.out  fort.11  gauss.f  gaussold.f  ran1.f  uran.f  x.txt  y.txt
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ vi ran1.f
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ ls
a.out  fort.11  gauss.f  gaussold.f  ran1.f  uran.f  x.txt  y.txt
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ vi ran1.f
```

So I have moved to the random number directory this is rannum, so I will list all the files, okay. So there are these several program here gausse.f that is the Gaussian random program and ran1.f that is the uniform random number program. So let us just first execute the uniform random number program. So I will edit the program vi ran1.f okay.

(Refer Slide Time: 01:48)



```
      program rannum
      dimension distr(101)
      idum=123
      do 50 i=1,101
   50 distr(i) = 0.
      do 100 i=1,10000
      a=ran0(idum)
c      write(11,*) a
      m = int(a/0.01) + 1
      distr(m) = distr(m) + 1
  100 continue
      write(11,*) 'distribution of random numbers between 0 and 1'
  101 format(5(1x,f12.4))
      write(11,101) distr
      end
      function ran0(idum)
      integer IA,IM,IQ,IR,MASK
      real ran0,AM
      PARAMETER(IA=16807,IM=2147483647,AM=1./IM,IQ=127773,IR=2386,
     1MASK=123459876)
      integer k
:
```

So this is my random number program. Let us look through the lines again. So first line is a program random number then dimension distribution 101, so what I want to do I want to generate lots of numbers between 0 and 1 and then see how they are distributed between 0 and 1, okay. So this random number program has to have initials seed idum. Remember in my earlier class I told you that this variable should be other than 0, so this is some starting seed idum.

Then this particular array distribution 101, I will set all those values to 0. So this particular loop do 50 i going from 1 to 101, 50 distribution i0 so all the initial are set to 0. So then what I do I will generate now random numbers, do 100 i going from 1 to 10000, so 100 is this loop. So do 100 means it will do all these lines up to 10000 so I want to generate 10000 random numbers.

So each time to get a random number a=ran0 idum this is my function ran0 is my function, okay. So each time I call this function it gives me a random number. So I want I can write all those random numbers in some file unit 11 but right now I am not interested in writing because I have verified that they are all between 0 and 1 you can verify that yourself. Now once I have a random number between 0 and 1 I want to decide at what region between 0 and 1 it lies.

So what are you do, I create an integer so this integer is a/0.01, so I know that between 0 and 1 I will make 100 wins and when I divide by 0.01 the number I get will be between 0 and 100. For example, suppose a is 0.5; if a is 2.5 when I divide by 0.1 I will 50. So that 0.5 will lie in the 50th box. Similarly, suppose a is exactly 1 which is unlikely so 1/0.0 is 100 so I will add 1 so that random number 1 when will go in the 101st box.

If the random number is 0 so 0/0.1 is 0, so that will go in the first box. The reason I am adding 1 I do not want this integer m to be 0 because if it is 0 I will have a problem of that accessing that particular variable, because usually all my variables will go from 1, 2, 3 up to 101. So in old compilers any variable with 0 as the array variable will not work, so all my arrays will start from 1 to 101.

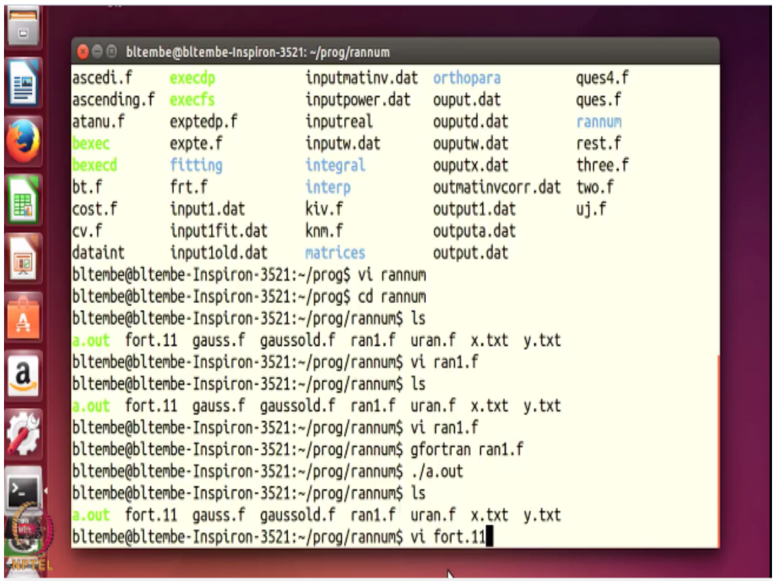
All my values will distribution 1, distribution 2 they will all go between 1 and 101 therefore I am adding this 1. So once I know where that random number lies so that particular box; now this is really like a box, so when I generate this 10,000 Random numbers each one will be between 0 and 1 so they will all be distributed and put in this boxes; it is a histogram so that histogram will be updated. So the total number of random numbers is 10,000.

So I expect that all the Random numbers will be distributed and if they are uniformly distributed each one of these distribution should be; since there are 100 bins so each one should give me about 100 because 100*100 is 10000. So I call the random number through this function, okay and it gets distributed and I will write all the distribution in random number in this file 11, I will write all that in file 11 and then I have given a format also.

I have given a format so that in each line; this is my format, okay. So I will write in file 11, okay. I will write, this term means no format whereas I want to also write on the screen, see this is, okay it writes; there is to write statements here, one distribution of random numbers; okay so let us execute. So there are two write statements, okay. So; okay first write statement it just writes the distribution of random number.

This is it writes whatever in this quotes, okay. First quote and second quote it writes in that file distribution of random numbers. The second write, writes me all the variables, okay. So I shall execute it, I shall execute this program. So let me come out.

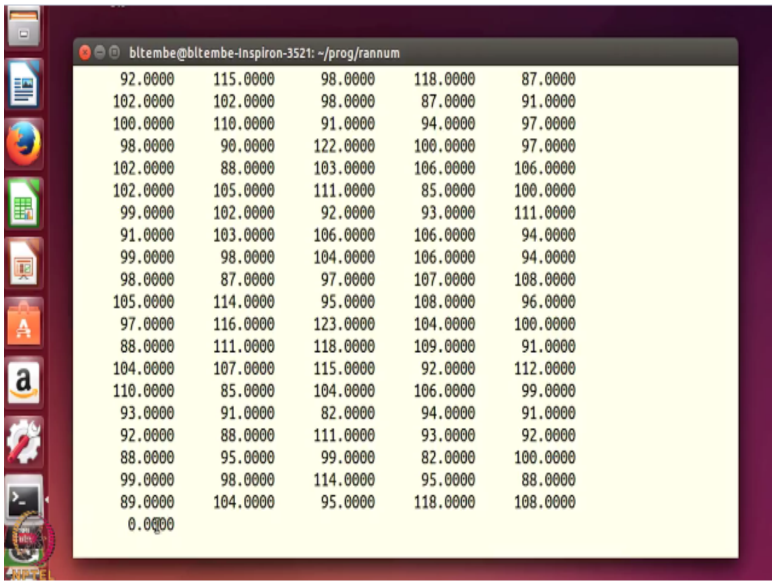
(Refer Slide Time: 07:14)



```
bltembe@bltembe-Inspiron-3521: ~/prog/rannum
ascedi.f     execdp          inputmatinv.dat  orthopara         ques4.f
ascending.f  execfs          inputpower.dat   ouput.dat         ques.f
atanu.f      exptedp.f       inputreal        ouputd.dat        rannum
bexec        expte.f         inputw.dat       ouputw.dat        rest.f
bexecd       fitting         integral         ouputx.dat        three.f
bt.f         frt.f           interp           outmatinvcorr.dat two.f
cost.f       input1.dat      kiv.f            output1.dat       uj.f
cv.f         input1fit.dat   knm.f            outputa.dat
dataint      input1old.dat   matrices         output.dat
bltembe@bltembe-Inspiron-3521:~/prog$ vi rannum
bltembe@bltembe-Inspiron-3521:~/prog$ cd rannum
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ ls
a.out  fort.11  gauss.f  gaussold.f  ran1.f  uran.f  x.txt  y.txt
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ vi ran1.f
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ ls
a.out  fort.11  gauss.f  gaussold.f  ran1.f  uran.f  x.txt  y.txt
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ vi ran1.f
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ gfortran ran1.f
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ ./a.out
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ ls
a.out  fort.11  gauss.f  gaussold.f  ran1.f  uran.f  x.txt  y.txt
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ vi fort.11
```

So I shall compile this ran1.f so now ./a.out, so this is executed. So now let us see where all the data is. So since I had not opened any file with unit 11 it creates now a file called 4.11. When you do not say anything the Fortran compiler itself create this file fort.11 because 11 there was nothing in the program for unit 11 so let us edit fort.11.

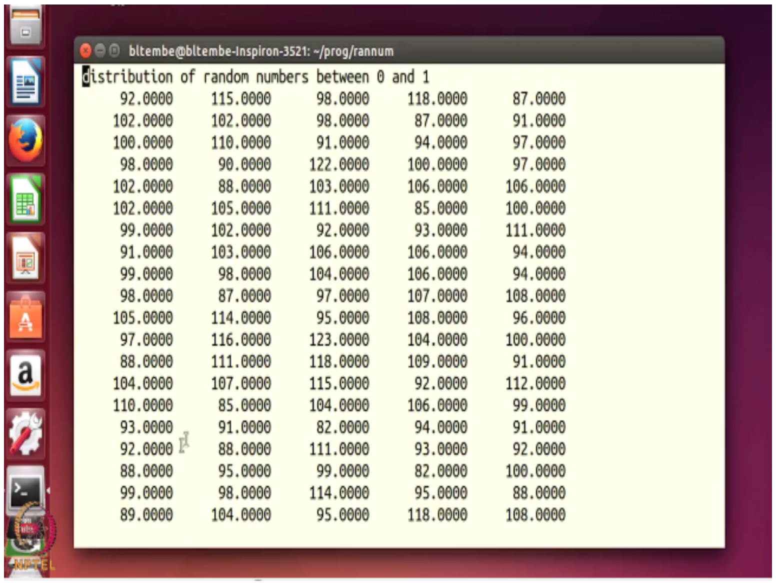
(Refer Slide Time: 07:56)



```
bltembe@bltembe-Inspiron-3521: ~/prog/rannum
   92.0000    115.0000     98.0000    118.0000     87.0000
  102.0000    102.0000     98.0000     87.0000     91.0000
  100.0000    110.0000     91.0000     94.0000     97.0000
   98.0000     90.0000    122.0000    100.0000     97.0000
  102.0000     88.0000    103.0000    106.0000    106.0000
  102.0000    105.0000    111.0000     85.0000    100.0000
   99.0000    102.0000     92.0000     93.0000    111.0000
   91.0000    103.0000    106.0000    106.0000     94.0000
   99.0000     98.0000    104.0000    106.0000     94.0000
   98.0000     87.0000     97.0000    107.0000    108.0000
  105.0000    114.0000     95.0000    108.0000     96.0000
   97.0000    116.0000    123.0000    104.0000    100.0000
   88.0000    111.0000    118.0000    109.0000     91.0000
  104.0000    107.0000    115.0000     92.0000    112.0000
  110.0000     85.0000    104.0000    106.0000     99.0000
   93.0000     91.0000     82.0000     94.0000     91.0000
   92.0000     88.0000    111.0000     93.0000     92.0000
   88.0000     95.0000     99.0000     82.0000    100.0000
   99.0000     98.0000    114.0000     95.0000     88.0000
   89.0000    104.0000     95.0000    118.0000    108.0000
    0.0000
```

So now you will see that the first writement was the distribution of random numbers between 0 and 1. So now it is writing it is writing those 101 values of the distribution. So you will see that, so the last one is 0 so that means there is no random number which was exactly 1, okay. This is the 101st point in that variable distr.

(Refer Slide Time: 08:23)



```
bltembe@bltembe-Inspiron-3521: ~/prog/rannum
distribution of random numbers between 0 and 1
   92.0000    115.0000     98.0000    118.0000     87.0000
  102.0000    102.0000     98.0000     87.0000     91.0000
  100.0000    110.0000     91.0000     94.0000     97.0000
   98.0000     90.0000    122.0000    100.0000     97.0000
  102.0000     88.0000    103.0000    106.0000    106.0000
  102.0000    105.0000    111.0000     85.0000    100.0000
   99.0000    102.0000     92.0000     93.0000    111.0000
   91.0000    103.0000    106.0000    106.0000     94.0000
   99.0000     98.0000    104.0000    106.0000     94.0000
   98.0000     87.0000     97.0000    107.0000    108.0000
  105.0000    114.0000     95.0000    108.0000     96.0000
   97.0000    116.0000    123.0000    104.0000    100.0000
   88.0000    111.0000    118.0000    109.0000     91.0000
  104.0000    107.0000    115.0000     92.0000    112.0000
  110.0000     85.0000    104.0000    106.0000     99.0000
   93.0000     91.0000     82.0000     94.0000     91.0000
   92.0000     88.0000    111.0000     93.0000     92.0000
   88.0000     95.0000     99.0000     82.0000    100.0000
   99.0000     98.0000    114.0000     95.0000     88.0000
   89.0000    104.0000     95.0000    118.0000    108.0000
```

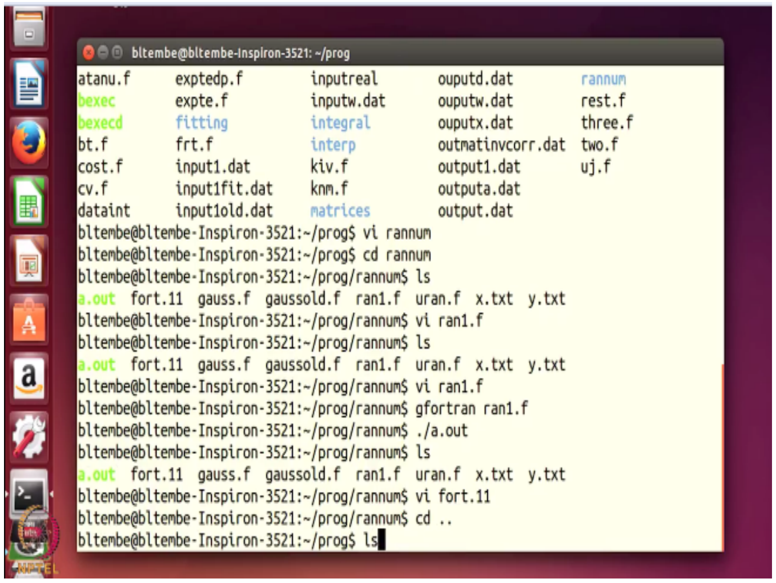
Now if you see from beginning so these are all so 92 is the number of random numbers between 0 and 0.01. 115 is the number of random numbers between 0.01 and 0.02. So you will see that most of these boxes most of these random numbers in this region they are all between around 80 to about the maximum value is; let us see which is the maximum value 106, 110, okay.

So the maximum value is around 123 and the lowest value will be somewhere around 87 or there could be smaller value. So which means these random numbers are more or less uniformly distributed. If it is the perfect random number system every box should contain 100 random number, but no random number is going to be completely perfect. So you will see that these random numbers are distributed between 0 and 1 fairly uniformly.

Lowest value is in the 80s 82, highest value around 120 so they are all uniformly distributed; so that means random number generator is a fairly good random number generator. So I would urge you that remember I said I took this from numerical recipes one of the first algorithm; there are many other random number programs in that book, I would urge you to try some different random number generator and execute the same program.

So next I will go to a Simpson's rule subroutine, okay. I shall go to another directory now.

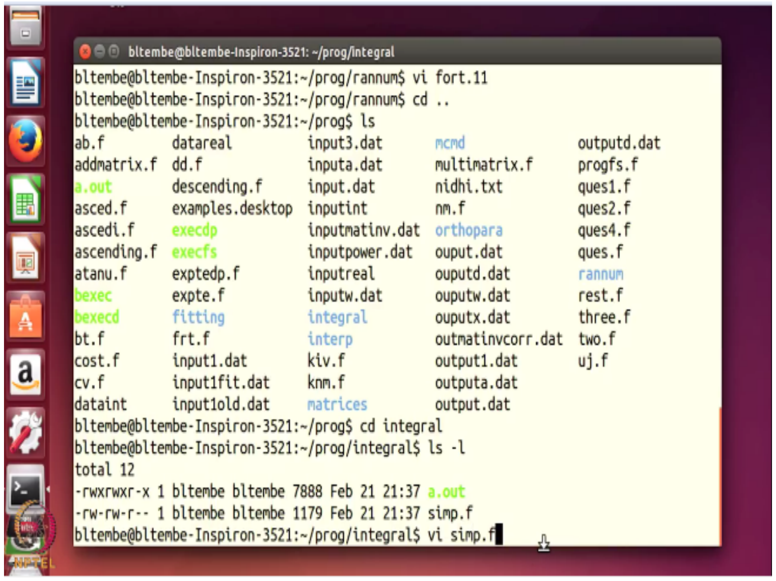
(Refer Slide Time: 09:58)



```
bltembe@bltembe-Inspiron-3521: ~/prog
atanu.f      exptedp.f      inputreal       ouputd.dat         rannum
bexec        expte.f        inputw.dat      ouputw.dat         rest.f
bexecd       fitting        integral        ouputx.dat         three.f
bt.f         frt.f          interp          outmatinvcorr.dat  two.f
cost.f       input1.dat     kiv.f           output1.dat        uj.f
cv.f         input1fit.dat  knm.f           outputa.dat
dataint      input1old.dat  matrices        output.dat
bltembe@bltembe-Inspiron-3521:~/prog$ vi rannum
bltembe@bltembe-Inspiron-3521:~/prog$ cd rannum
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ ls
a.out  fort.11  gauss.f  gaussold.f  ran1.f  uran.f  x.txt  y.txt
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ vi ran1.f
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ ls
a.out  fort.11  gauss.f  gaussold.f  ran1.f  uran.f  x.txt  y.txt
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ vi ran1.f
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ gfortran ran1.f
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ ./a.out
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ ls
a.out  fort.11  gauss.f  gaussold.f  ran1.f  uran.f  x.txt  y.txt
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ vi fort.11
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ cd ..
bltembe@bltembe-Inspiron-3521:~/prog$ ls
```

So I shall go back to my original directory.

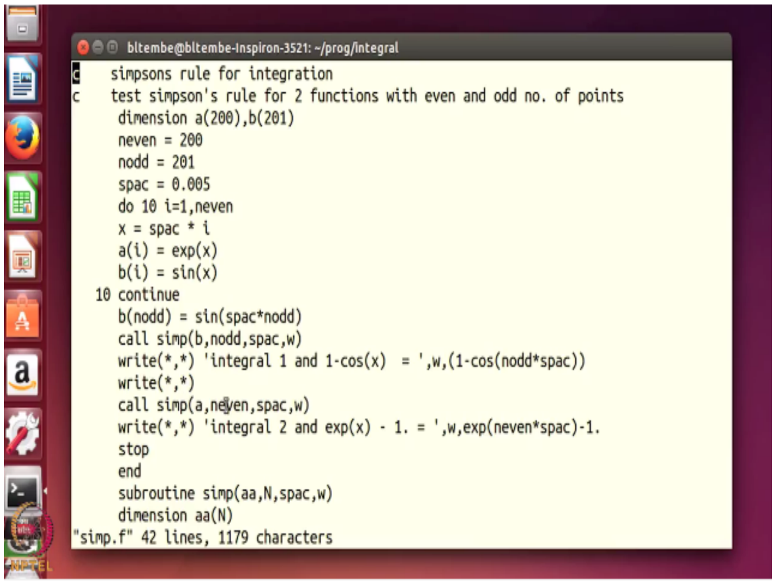
(Refer Slide Time: 10:02)



```
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ vi fort.11
bltembe@bltembe-Inspiron-3521:~/prog/rannum$ cd ..
bltembe@bltembe-Inspiron-3521:~/prog$ ls
ab.f          datareal         input3.dat       mcmd              outputd.dat
addmatrix.f   dd.f             inputa.dat       multimatrix.f     progfs.f
a.out         descending.f     input.dat        nidhi.txt         ques1.f
asced.f       examples.desktop inputint         nm.f              ques2.f
ascedi.f      execdp           inputmatinv.dat  orthopara         ques4.f
ascending.f   execfs           inputpower.dat   ouput.dat         ques.f
atanu.f       exptedp.f        inputreal        ouputd.dat        rannum
bexec         expte.f          inputw.dat       ouputw.dat        rest.f
bexecd        fitting          integral         ouputx.dat        three.f
bt.f          frt.f            interp           outmatinvcorr.dat two.f
cost.f        input1.dat       kiv.f            output1.dat       uj.f
cv.f          input1fit.dat    knm.f            outputa.dat
dataint       input1old.dat    matrices         output.dat
bltembe@bltembe-Inspiron-3521:~/prog$ cd integral
bltembe@bltembe-Inspiron-3521:~/prog/integral$ ls -l
total 12
-rwxrwxr-x 1 bltembe bltembe 7888 Feb 21 21:37 a.out
-rw-rw-r-- 1 bltembe bltembe 1179 Feb 21 21:37 simp.f
bltembe@bltembe-Inspiron-3521:~/prog/integral$ vi simp.f
```

So you will see in my main directory program directly there are many, many subdirectories so I went to the random numbers subdirectory and executed. Now we will go to the integral subdirectory, okay. So in this directory there are two files one is simp.f and one is a.out, okay. Let us edit the simp.f.

(Refer Slide Time: 10:30)



```
c     simpsons rule for integration
c     test simpson's rule for 2 functions with even and odd no. of points
      dimension a(200),b(201)
      neven = 200
      nodd = 201
      spac = 0.005
      do 10 i=1,neven
      x = spac * i
      a(i) = exp(x)
      b(i) = sin(x)
   10 continue
      b(nodd) = sin(spac*nodd)
      call simp(b,nodd,spac,w)
      write(*,*) 'integral 1 and 1-cos(x)  = ',w,(1-cos(nodd*spac))
      write(*,*)
      call simp(a,neven,spac,w)
      write(*,*) 'integral 2 and exp(x) - 1. = ',w,exp(neven*spac)-1.
      stop
      end
      subroutine simp(aa,N,spac,w)
      dimension aa(N)
"simp.f" 42 lines, 1179 characters
```

So what we have done in this particular program so this is a program for Simpson's rule integration and I am using two functions for the Simpsons rule integration. Remember the normal Simpson's rule program is for odd number of points. So therefore, this b 201 I will generate the entire function values in this particular array, okay and I will also have array with even number of point because I want to show how to use the Simpson's rule both for even number of points and odd number of data points, okay.

So first I shall generate my data points, okay. So 1 is neven= 200; nodd= 201. For both these functions I will leave a spacing 0.005 because I have 200 points 0.005*200 is 1. So I want to generate two functions in these arrays. So for the even array I am going to use exponential function. If there is a 200 I will use an exponential function for my even array and for an odd array I will use sine function, okay.

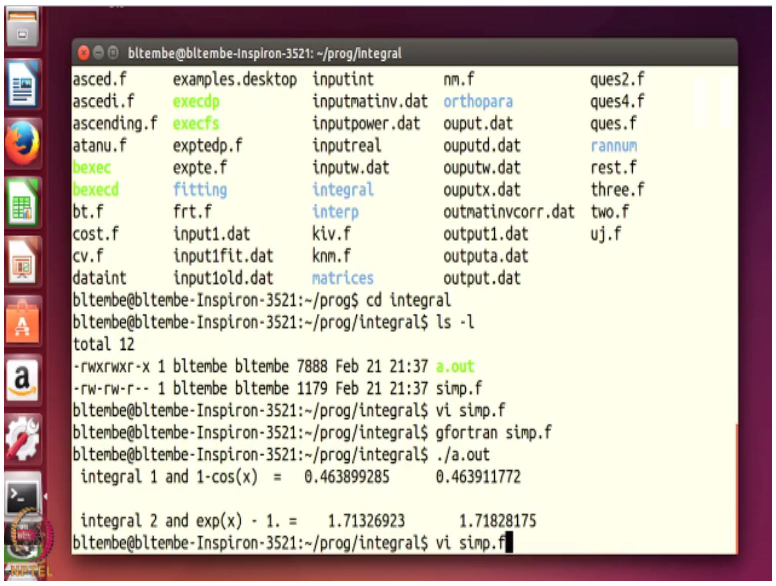
I will use a sine function for odd array, okay. So for odd arrays there was extra points 200+ points. So I generate that b(bnodd)= sin of spac*nodd. This spacing is 0.005 for both variables and so I have generate an exponential function and put that in array a and have put a sine function and put in array b; both of them are generated for 200 points. And since the sine function as an extra dimension that is it has 201 so have generated that last point for the b array, okay.

So now I am going to call the Simpson's rule twice. One for the odd number of points, see this is call simp b, nodd, spac, w. So this call statement has four variables, b is my array variable, n odd is the number of data points in that particular array; spacing is the spacing in the integration process and w is the value of the integral. So once it call this subroutine it has got my answer.

So I will write on the screen write the value of the integral first integral and 1 - cos x, okay. So this is the value of the integral. So I want to come the numerical value of the integral with the analytical value of integral, okay. So I am writing the value of the integral as well as the analytical value of the integral. So once this is done in the next call statement I am using the same subroutine and now I am calling, so next time call simp a, neven, space, w.

So in the earlier case I have called an odd number of data points; in the second case the array is different now. See in the second case array is a; the number of points is neven, spac and w. These are the same. So what this illustrates, I can you the same subroutine with a different number of dimensions of that array, okay. And second one is an exponential, so it will write the integral the value of the integral. Both the numerical value and analytical value stop and end.

(Refer Slide Time: 14:07)



```
bltembe@bltembe-Inspiron-3521: ~/prog/integral
asced.f      examples.desktop  inputint         nm.f               ques2.f
ascedi.f     execdp            inputmatinv.dat  orthopara          ques4.f
ascending.f  execfs            inputpower.dat   ouput.dat          ques.f
atanu.f      exptedp.f         inputreal        ouputd.dat         rannum
bexec        expte.f           inputw.dat       ouputw.dat         rest.f
bexecd       fitting           integral         ouputx.dat         three.f
bt.f         frt.f             interp           outmatinvcorr.dat  two.f
cost.f       input1.dat        kiv.f            output1.dat        uj.f
cv.f         input1fit.dat     knm.f            outputa.dat
dataint      input1old.dat     matrices         output.dat
bltembe@bltembe-Inspiron-3521:~/prog$ cd integral
bltembe@bltembe-Inspiron-3521:~/prog/integral$ ls -l
total 12
-rwxrwxr-x 1 bltembe bltembe 7888 Feb 21 21:37 a.out
-rw-rw-r-- 1 bltembe bltembe 1179 Feb 21 21:37 simp.f
bltembe@bltembe-Inspiron-3521:~/prog/integral$ vi simp.f
bltembe@bltembe-Inspiron-3521:~/prog/integral$ gfortran simp.f
bltembe@bltembe-Inspiron-3521:~/prog/integral$ ./a.out
 integral 1 and 1-cos(x)  =   0.463899285      0.463911772

 integral 2 and exp(x) - 1. =    1.71326923       1.71828175
bltembe@bltembe-Inspiron-3521:~/prog/integral$ vi simp.f
```

So now let us briefly see how that subroutine works. In the subroutine remember we have already studied the Simpson's rule program. So if the number of data is odd I will keep on multiplying alternate values with 4 and 2 remember. The way I will do the Simpsons integration each time 4 times 1 value of the array and two times the other value of the array, okay.

So I would urge you to look at this in great detail. So this is for; this is both for Even and Odd number. Now let us see if n/2*2=n okay, so that means n is an integer. If integer/2; suppose this is an even integers say it is 100; 100/2 will be 50; 50*2 will be 100, so 100=100 then that is an even number it will go to 100. But suppose n is an odd number, okay so this is an integer division if n is 1; 1/2.

So 1/2 is 0.5 in your real arithmetic but in integer arithmetic there is no such thing as 1/2 .so this will be 0. So 1/2 will be 0 if it is an integer but 2/2 will be 1, so anything other than a whole number this will be 0 that is an then integer division. So whenever n is odd so n/2*2 will not be equal to N. So whenever this is odd number it will do the first loop here and whenever it is an even number it goes to 100 and it will do the calculation for the even number.

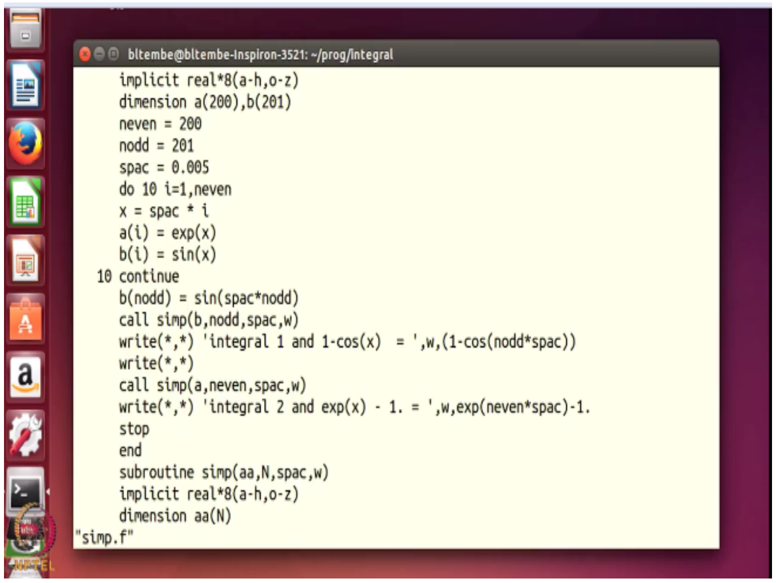
So what I do for the even number is that the last two points I will integrate using a Trapezoidal rule. And first n-1 which is odd they will do exactly as I did for the Simpson's rule with the odd number of points. So this particular subroutine illustrates that the variable aa could have dimensions of N. When it is even number it will go to 100 and do the calculation; when it is an odd number it will do the calculations here.

And this illustrate that the array variables in a subroutine can be variable and this is very important because you do know which main program will have how many points for that integration. So this illustrates how to give a variable number of dimensions in the subroutine. So I have done it using all the things that are in the subroutine are given in that subroutine statements aan space w.

And the array aa has been assigned dimension n and this n is a variable; it takes whatever values it get from the main program. So I have now this particular subroutine simp so I shall execute it; simp.f, so the results are already given here. When I execute the first integral was a sine function so when I integrate 0.4638 is the value of the integral done by the Simpsons subroutine and 0.4639 is the value of the integral using the analytical integration.

Similarly, for the exponential function the value of the integral is 1.7132 using my subroutine, and it is 1.718 using the analytic function, so you will see that these are fairly accurate, these are fairly accurate here the accuracy is only up to the 3rd decimal place. But if I want much greater accuracy I shall the way I should do it is to use double precision, I will just illustrate how to do that.

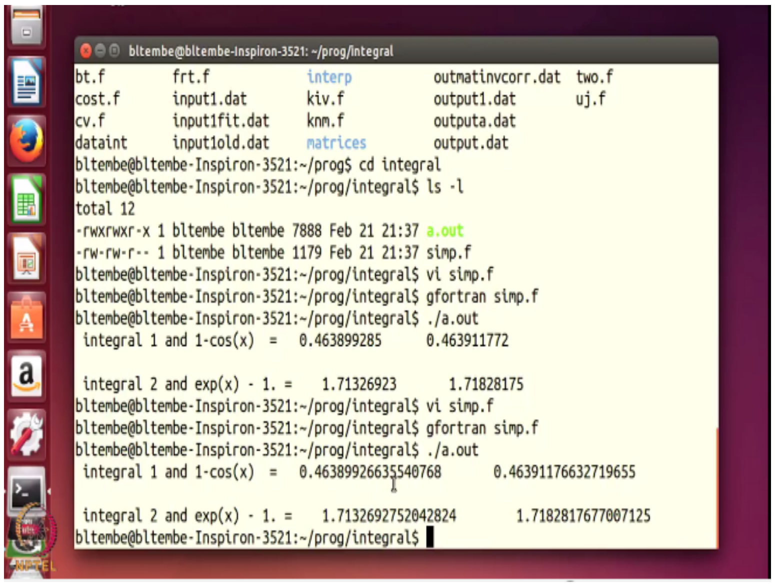
(Refer Slide Time: 18:27)



```
bltembe@bltembe-Inspiron-3521: ~/prog/integral
      implicit real*8(a-h,o-z)
      dimension a(200),b(201)
      neven = 200
      nodd = 201
      spac = 0.005
      do 10 i=1,neven
      x = spac * i
      a(i) = exp(x)
      b(i) = sin(x)
   10 continue
      b(nodd) = sin(spac*nodd)
      call simp(b,nodd,spac,w)
      write(*,*) 'integral 1 and 1-cos(x)  = ',w,(1-cos(nodd*spac))
      write(*,*)
      call simp(a,neven,spac,w)
      write(*,*) 'integral 2 and exp(x) - 1. = ',w,exp(neven*spac)-1.
      stop
      end
      subroutine simp(aa,N,spac,w)
      implicit real*8(a-h,o-z)
      dimension aa(N)
"simp.f"
```

So what I do if I want higher accuracy, okay. So what I will do; so I have declared all the variables in the sub program; in the main program to be double precision and so I should do that exactly in the subroutine also, okay. So this implicit, what it does is it makes all variable which are starting with a and h and o and g to be double precision variables. So let us see.

(Refer Slide Time: 19:38)



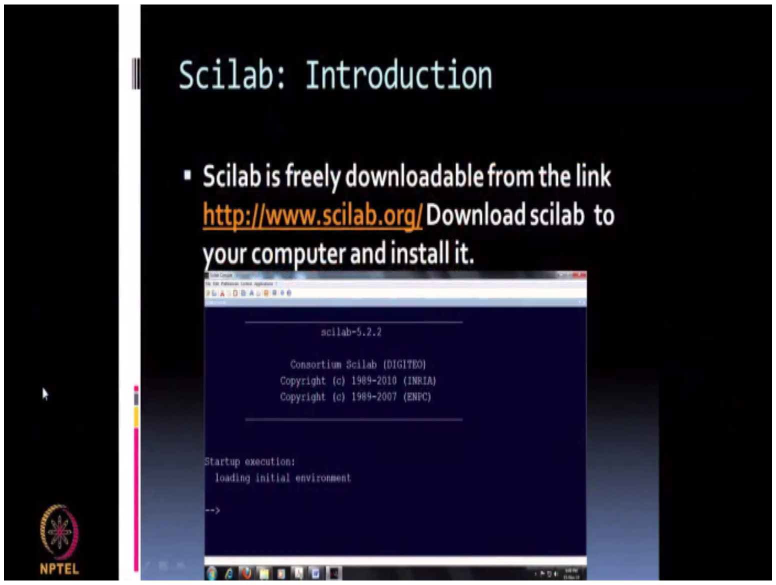
```
bltembe@bltembe-Inspiron-3521: ~/prog/integral
bt.f         frt.f          interp        outmatinvcorr.dat  two.f
cost.f       input1.dat     kiv.f         output1.dat        uj.f
cv.f         input1fit.dat  knm.f         outputa.dat
dataint      input1old.dat  matrices      output.dat
bltembe@bltembe-Inspiron-3521:~/prog$ cd integral
bltembe@bltembe-Inspiron-3521:~/prog/integral$ ls -l
total 12
-rwxrwxr-x 1 bltembe bltembe 7888 Feb 21 21:37 a.out
-rw-rw-r-- 1 bltembe bltembe 1179 Feb 21 21:37 simp.f
bltembe@bltembe-Inspiron-3521:~/prog/integral$ vi simp.f
bltembe@bltembe-Inspiron-3521:~/prog/integral$ gfortran simp.f
bltembe@bltembe-Inspiron-3521:~/prog/integral$ ./a.out
 integral 1 and 1-cos(x)  =   0.463899285      0.463911772

 integral 2 and exp(x) - 1. =    1.71326923       1.71828175
bltembe@bltembe-Inspiron-3521:~/prog/integral$ vi simp.f
bltembe@bltembe-Inspiron-3521:~/prog/integral$ gfortran simp.f
bltembe@bltembe-Inspiron-3521:~/prog/integral$ ./a.out
 integral 1 and 1-cos(x)  =   0.46389926635540768       0.46391176632719655

 integral 2 and exp(x) - 1. =    1.7132692752042824       1.7182817677007125
bltembe@bltembe-Inspiron-3521:~/prog/integral$ 
```

I will compile again. So we see that; first thing that we see is that the integration has been done up to a very large number of digits so 43 46389, 4639 show the accuracy did not change very much this is the 4th decimal place. So second one 1.732 1.718 although accuracy did not change as far as the numbers are concerned still the precision is much higher in a Real*8 rate calculation, so we did both at Double Precision calculation as well as Single Precision calculation.

So what I will do I will conclude my demonstration part here and now I shall start using Scilab, okay. So we shall now switchover to a new software call Scilab and the later part of the lecture we will be using the Scilab. So we will continue now with our discussion on Scilab. As I mentioned several times, so this is a free software which can be downloaded on your Windows operating system as well as on a Linux operating system.

So how to download on a Linux operating system you can see on a YouTube. On my present computer I have downloaded it on the Windows operating system.

(Refer Slide Time: 21:08)



Scilab: Introduction

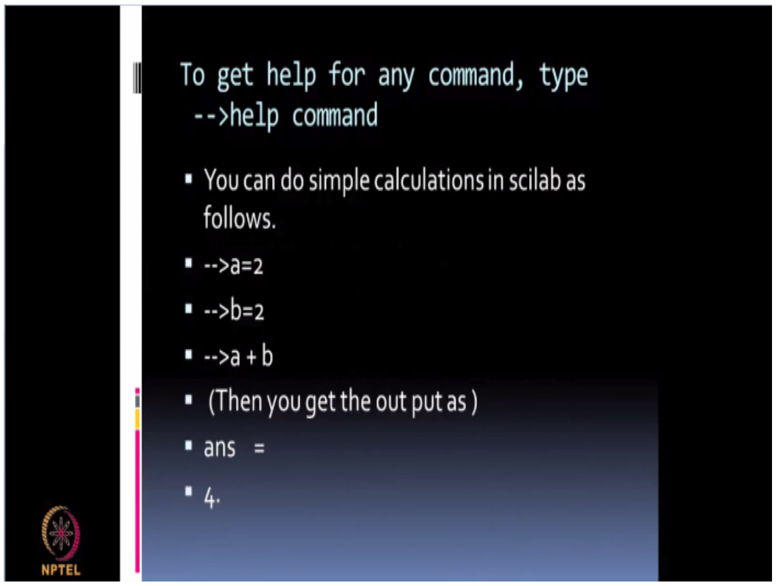
- Scilab is freely downloadable from the link http://www.scilab.org/ Download scilab to your computer and install it.

```
                    scilab-5.2.2

            Consortium Scilab (DIGITEO)
          Copyright (c) 1989-2010 (INRIA)
          Copyright (c) 1989-2007 (ENPC)


Startup execution:
  loading initial environment

-->
```

So see the command to download it. So it is; Scilab is a freely downloadable software; you can use the link; https//:www.scilab.org. So download this on your computer and install this, okay. Once you install this and run the Scilab environment what you get is shown on this blue screen here.

You will have a top line from Scilab version; this is a 5.2. It will give all these copyright commands. And it loading the initial environment this particular arrow /- - > this is the Scilab environment. Once you have this particular arrow that means you are ready to execute all the commands in Scilab. So we will illustrate a very simple command now and then continue with all the more advanced commands, okay.

(Refer Slide Time: 22:05)

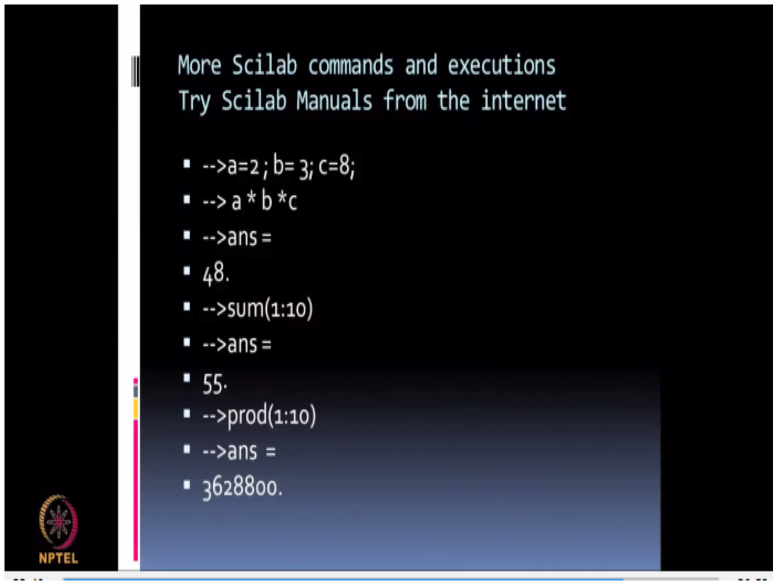


To get help for any command, type
-->help command

- You can do simple calculations in scilab as follows.
- -->a=2
- -->b=2
- -->a + b
- (Then you get the out put as)
- ans =
- 4.

So one good thing about Scilab is you can get help for any command, okay. Suppose you want some command on a matrix operation okay so you just say help that command name, it will be a full detail of how that command works, okay. So now let us start some very simple operation in Scilab. Suppose you want to do calculations a=2; b=2 and a+b so you just a=2 enter, b=2 enter, a+b it will give the output as a+b, okay. So this is just an addition operation in Scilab. Okay.

(Refer Slide Time: 22:46)



More Scilab commands and executions
Try Scilab Manuals from the internet

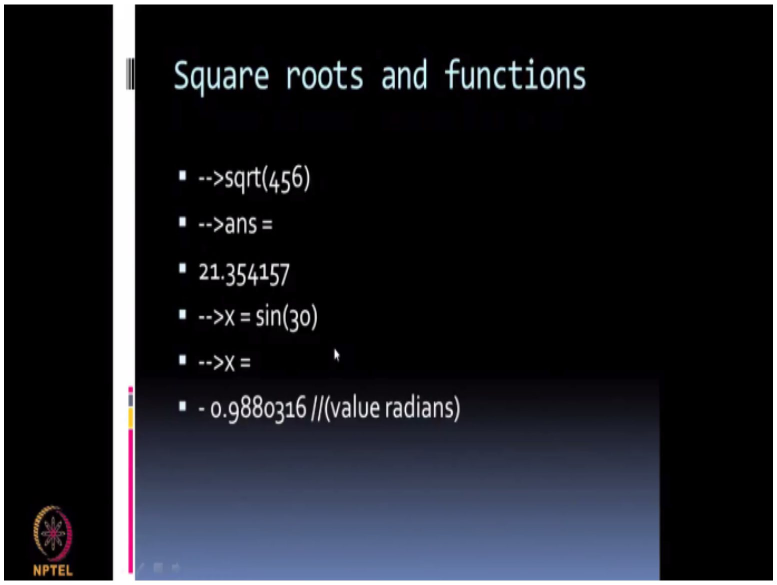
- -->a=2 ; b= 3; c=8;
- --> a * b *c
- -->ans =
- 48.
- -->sum(1:10)
- -->ans =
- 55.
- -->prod(1:10)
- -->ans =
- 3628800.

Then, let us see some other operation, support I want to do multiplication operations a=2; b=3; c=8. So to separate two different lines in Fortran I have to write all these three in different lines. So as semicolon separates two particular commands. So a=2 was one command; b=3 separate command; c=8 separate command; enter, so then once you enter a* b* c. Now it just give the answer as 48.

So this is how the commands are separated and executed. Now Scilab has different forms of Do loops. So now suppose I want to sum numbers from 1 to 10 the way I did it in Fortran I have to start a do loop then initiate the do loop then end the do loop then sum all the variables. Here it is very, very simple; sum*(1:10) that means it will sum all the number between 1 and 10, so answer is 55.

The same way as I sum 1 to 10 I can also take products * bracket; prod*(1:10) that means it will take all the numbers between 1 and 10 and multiply them and this is my answer, okay. So that is a product of N numbers.

(Refer Slide Time: 24:15)



Square roots and functions

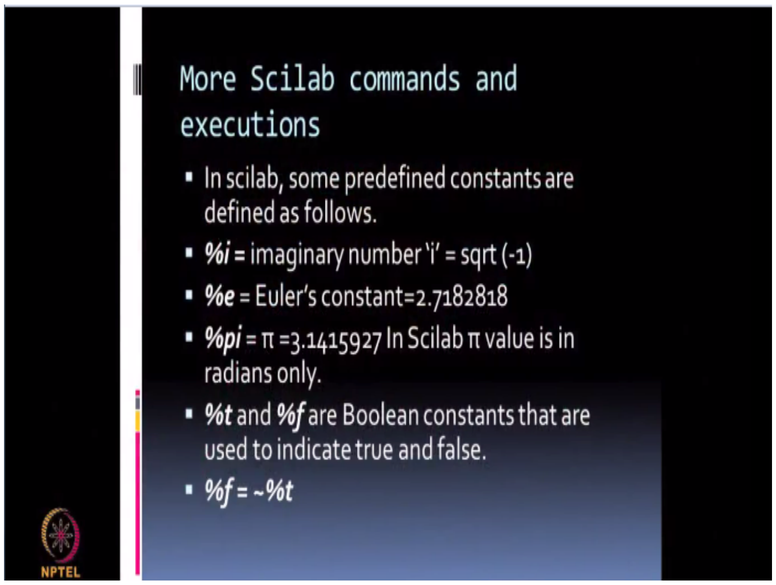
- -->sqrt(456)
- -->ans =
- 21.354157
- -->x = sin(30)
- -->x =
- - 0.9880316 //(value radians)

So now just as Fortran has different kinds of function Scilab also has different functions. For example, suppose I want square root of 456 so it has a square root function into bracket 456 you do that then this is the answer. So what it means is that once you get this arrow you are in an executable environment of the Scilab and everything is a command there, okay.

So in Fortran when you type a.out the commands were executed so that means this arrow is in an executable environment everything that you give here is a command. But suppose I want a comment statement a comment statement can also be given in Scilab and that comment statement is given below. So before I go to that, so square root 456, give me the answer. Now suppose x=sin 30 okay, so when it gave sin 30 so it gave this value -0.9880316 and this value is in radians.

So the comment lines in Scilab are with double slash; when I give a double slash that means whatever is written on the right hand side it is not executed it is a comment, okay. So this says that this sin 30 that is executed in radians, okay. So now what I want to do, Scilab has certain predefined constant, okay. So I will give a list of these constants and conclude this lecture. So % i is an imaginary number, imaginary number you know that your; you have real numbers and complex numbers,

(Refer Slide Time: 26:03)



More Scilab commands and executions

- In scilab, some predefined constants are defined as follows.
- ***%i*** = imaginary number 'i' = sqrt (-1)
- ***%e*** = Euler's constant=2.7182818
- ***%pi*** = π =3.1415927 In Scilab π value is in radians only.
- ***%t*** and ***%f*** are Boolean constants that are used to indicate true and false.
- ***%f = ~%t***

So % i would mean it is x+iy so this is imaginary number; % e means that is an Euler constant 2.7182828. So what I will do I will stop this lecture here in the next one I will conclude, I will start with the same particular slide. Thank you.