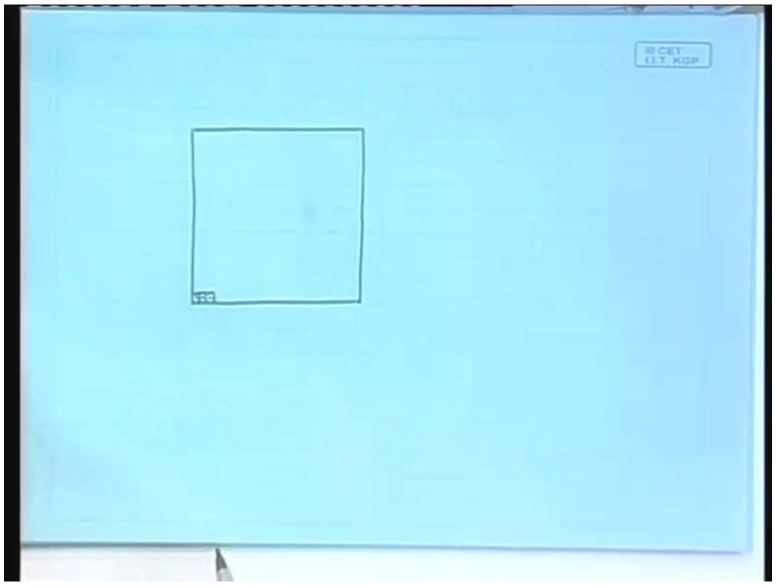
Chaos Fractals and Dynamical Systems

Prof. S. Banerjee
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

Lecture No # 18
IFS Algorithms

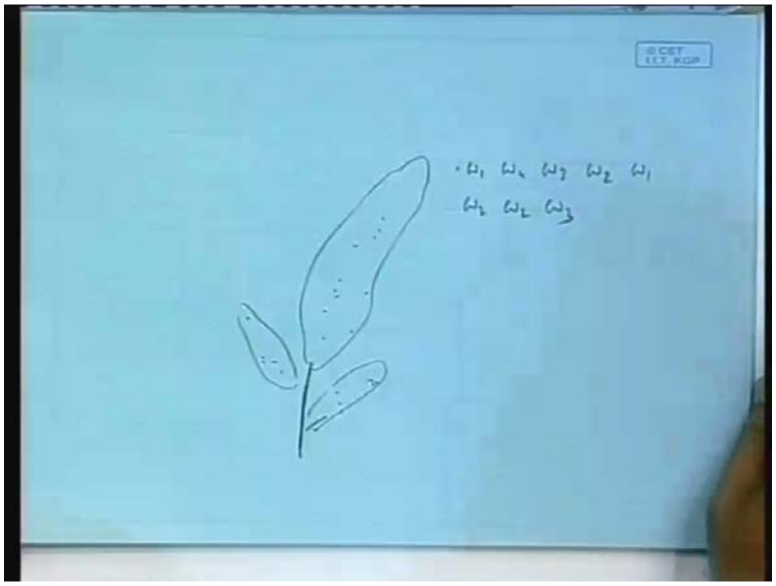
In the last class we have seen the basic mathematical frame work with which we can generate fractals and in the end of last class I gave you the IFS that means, a collection of affined transformations which when iteratively applied, will generate some of the well known fractals for example, Sierpinski triangle the fund and like that and then obviously the question was in your mind. I told you that the simple way to actually generate the fractals would be to take a collection of pixels.

(Refer Slide Time: 00:01:37 min)



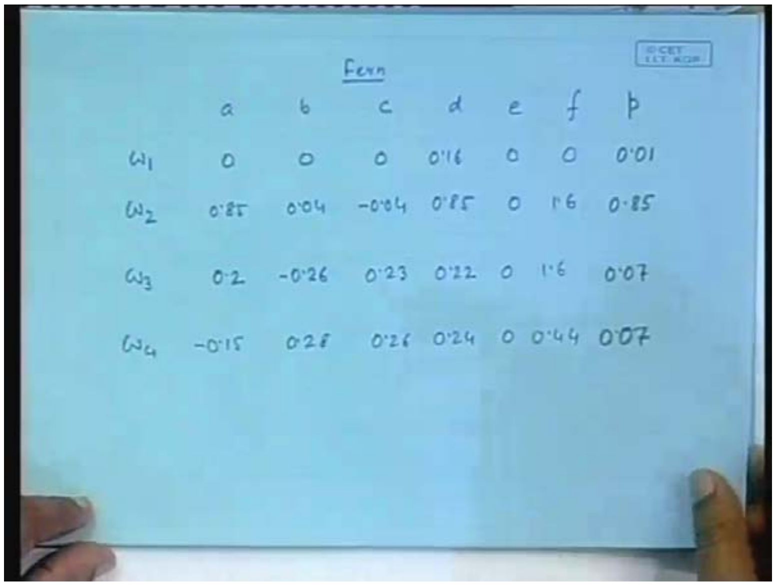
You might start from any image because the initial condition does not really matter, it will always converge on to that particular fractal object. So that collection of pixels would be some number of pixels this way and some number of pixels that way and naturally each pixels will be spanning from a particular real number to particular real number, so this will be in effect like boxes. So each pixels is actually like a box so start from the center of the box and see where that mapping takes you and identify which box will be filled and again repeat the procedure starting from the center of that box. Then it will be a simple procedure by which you can generate the fractals. Some of you may feel somewhat uncomfortable to work with pixels, I found that many students feel more comfortable with working with numbers so in that case there are few ways of doing it. One you might start from just one number means one point then if you apply suppose there are 4 of the w's four of the affine transformation available like in the case of fern if you apply each of them how many points you will get? 4, so in the next iterate you will get some what 4 points, in the next iterate you can apply the same 4 affine transformations on each of the 4 points as a result of which you get how many points? 16 and so on and so forth and if you do this procedure within 3 or 4 iterations you will find that you got the fractal object, all you need to do is to plot it. It will take atmost 10 lines of comman in Metlab very simple but the problem is that, the more you advance in terms of the iterate of this, the more will be the number of points and after some time you will find that the memory of Metlab will crash, you will not be able to go further. That is why this problem though very simple but doesn't really work very effectively after some time it will see 1 to 4, 4 to 16 to within 3, 4 iterates you'll find that the RAM of the computer will be formed. One alternative way of doing it is, so this is called deterministic algorithm, where you have deterministically applying and seeing the sintex shape. So I have given you one algorithm in which the object changes shape and finally deform and deform and deform and finally you will get the final fractal object. That means before your eyes you will be able to see how the action of the IFS the iterated function system is transforming the object finally converging on to that object on to the fractals, but in this way if you are working with the points you will not be able to see the transformation that means how the whole object is changing and transforming and finally converging on to that you will not be able to see. You will be able to see that points randomly falling and finally a shape emerging out of it. If you progressively eliminate or delete erase from the computer screen the earlier iterates and keep only the last one you more or less able to see the final form factor, but how the shape is changing you will not be able to see and it is often instructive to see how the shape is changing and converging onto the final one that is why I would propose you to go by this pixel algorithm. There is another way of doing. The third way of doing it is for example in the fern image.

(Refer Slide Time: 00:06:15 min)


© CET I.I.T. KGP
$\cdot\,\omega_1\ \omega_4\ \omega_3\ \omega_2\ \omega_1$
$\omega_4\ \omega_2\ \omega_3$

Just recall how it was, there was one leaf here, there was something here which resemble the whole thing. That is how it was, the 4 w's four affine transformations actually create the four parts, but the number of points generated by each part need not be the same means that if you start from a point and generate four points from that generate 16 points, the points generated by each W will be the same in number that means the number of points falling in this part will be equal to some number falling in this part so on and so forth. That is not always desire because it might be so that you want more number of points here than this point because of the particular shape of the object which means that you might also assign a probability of an iterate falling here, probability of an iterate falling here and so on and so forth. If you do that then the algorithm would be something like this, start from any point and keep on applying those iterates W_1 or W_2 or W_3 or W_4 depending on their own respective probability. So if W_2 has a twice probability than W_1, you will choose W_2 with a probability double than the probability of W_1. So what will happen is that you will get a sequence which will looks something like that start from a point and then apply W_1 apply W_4 then apply W_3 then apply W_2 then apply W_1 then apply W_2 then again apply W_2 then apply W_3. You might wonder what is generating the sequence that is being generated this sequence in which sequence you will be applying, every time you have been choosing from these 4 W's depending on their respective probabilities. In that way the point will keep on falling like this and finally slowly we will find the figure in merging order. So the earlier one is called deterministic algorithm and this one is called the random iteration algorithm where the iteration are being applied randomly depending on some kind of probability assigned to it. Yes I will come to that.

(Refer Slide Time: 00:09:24 min)

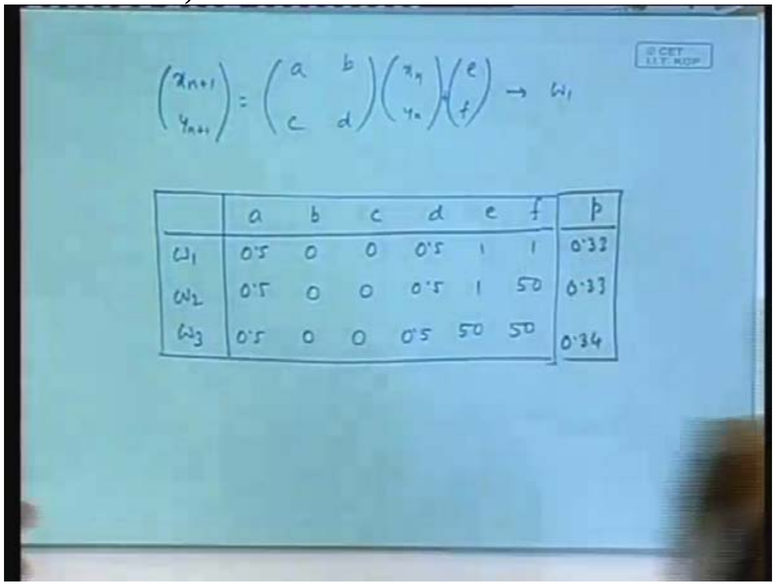


Fern

	a	b	c	d	e	f	p
ω_1	0	0	0	0.16	0	0	0.01
ω_2	0.85	0.04	-0.04	0.85	0	1.6	0.85
ω_3	0.2	-0.26	0.23	0.22	0	1.6	0.07
ω_4	-0.15	0.28	0.26	0.24	0	0.44	0.07

So I had given these yesterday and there should be the one for the fern. We will also lead to assign the probabilities to these four then that means they have to be the p another column. Now this column I will now give you for the fern, you write down if you want to write the random iteration algorithm you will need two assign probabilities say 0.01 very low probability of this one. Explanation why I will come to a little later 0.85 the last 0.070.

(Refer Slide Time: 00:10:19 min)



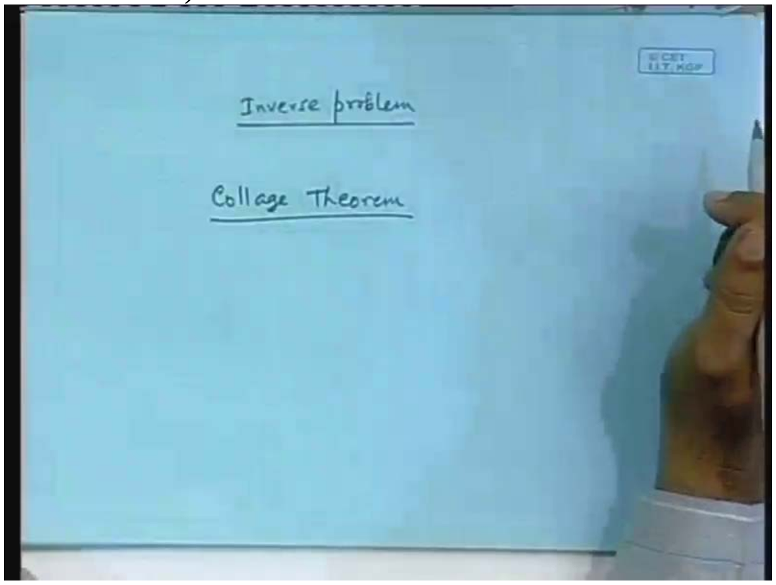
$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} \rightarrow w_1$$

	a	b	c	d	e	f	p
w_1	0.5	0	0	0.5	1	1	0.33
w_2	0.5	0	0	0.5	1	50	0.33
w_3	0.5	0	0	0.5	50	50	0.34

For the Sierpinski triangle the p's are 0.33 3.30.33 and these will also be the same actually but in order to make it one you need to make it 0.34. They have their equal probability because they have three parts in that progressing. So essentially I had told you about three possible algorithms by which you can generate this. The first one is called the deterministic algorithm while where you are starting with a shape not a point and the shape is changing and finally converging on to the fractal and in the other you are actually starting from a point and keeping on applying the iterates one after the other, either just in proper sequence in which case you have not applying the probabilistic algorithm or you have choosing from which W to apply depending on their probabilities. You can easily write the programs for that. So before you come to the next class at least I want one of these routines to be written that's an assignment. One of these routines to be written and you should be able to generate the fractals.

Now let us come to the interesting point of how to generate these numbers that is the inverse problem is the most interesting problem, all the time i am giving you the numbers and you have been thinking where the hell does he gets these numbers. yes the question is quiet obvious. So let us attack the inverse problem.

(Refer Slide Time: 00:12:09 min)

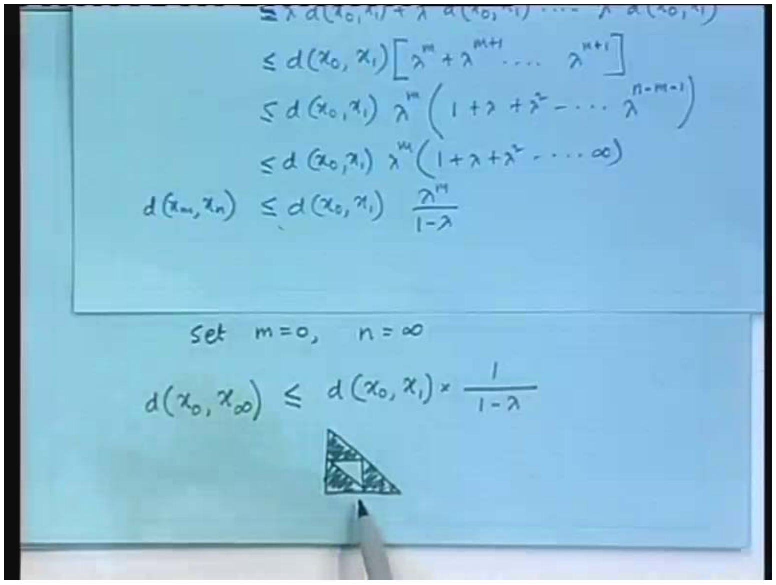


Inverse problem

Collage Theorem

The inverse problem is essentially that given a particular fractal object, how I generate the iterated function system? First was given an iterated function system how can I generate the fractal. Now the problem is given a fractal object how do I generate the iterative system. for that this central theorem is called collage theorem. And the collage theorem is essentially a corollary of the Banach's contraction mapping theorem which I proved in the last class. The reason I went to the length of proving it was that I will need to use a part of that proof. So let us get back there this is what we did in the last class

(Refer Slide Time: 00:13:06 min)



$$\leq \lambda\, d(x_0, x_1) + \lambda\, d(x_0, x_1) \cdots \lambda\, d(x_0, x_1)$$

$$\leq d(x_0, x_1)\left[\lambda^m + \lambda^{m+1} \cdots \lambda^{n+1}\right]$$

$$\leq d(x_0, x_1)\, \lambda^m \left(1 + \lambda + \lambda^2 - \cdots \lambda^{n-m-1}\right)$$

$$\leq d(x_0, x_1)\, \lambda^m \left(1 + \lambda + \lambda^2 \cdots \infty\right)$$

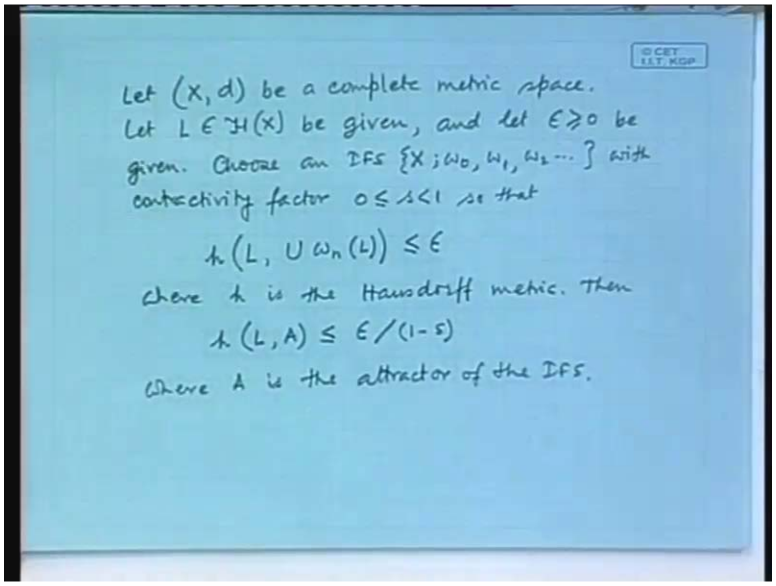
$$d(x_m, x_n) \leq d(x_0, x_1)\, \frac{\lambda^m}{1-\lambda}$$

Set $m = 0$, $n = \infty$

$$d(x_0, x_\infty) \leq d(x_0, x_1) \times \frac{1}{1-\lambda}$$

We started with a distance between two points two elements x_m and x_n and we showed that must be less than this. What is this in the right hand side? You have got the distance between the first two, this is the starting point, this is after the first iterate x_1 the distance between them and then in terms of the contractivity fractal we got the number and we have argued that since lambda is less than one therefore this will go to zero now let us start from there. If you start from there and if you now set m to be zero and n to be infinity. What you get? In this here set m=0, n equal to infinity, the statement immediately leads, lambda to the power of 0 is 1. See what it says, it says that if I start from any particular object that means suppose I have got Sierpinski triangle or something like that so on and so forth we have already done that then, suppose this is my object then i have to choose the transformations in such a way that after the first iterate you get more or less the same object. How can you do that? By generating, by writing each of the w's to make parts of the original object. again if you start from a particular given geometrical object here it says that you choose your w's in such a way that this whole affine transformation, whole IF is applied get the second one X_1 that will be very close to the original one. If that is so that means if the distance between the two objects starting point and after the first iterate is very small and if the transformation are contractive, then you can always say what is this fellow? X infinity is the attractor, start from anywhere, the distance will be smaller than this one. Now let us state this theorem in mathematical terms.

(Refer Slide Time: 00:16:43 min)


© CET I.I.T. KGP

Let (X, d) be a complete metric space.
Let $L \in \mathcal{H}(X)$ be given, and let $\epsilon \geq 0$ be given. Choose an IFS $\{X; \omega_0, \omega_1, \omega_2 \cdots\}$ with contractivity factor $0 \leq s < 1$ so that

$$h(L, \cup \omega_n(L)) \leq \epsilon$$

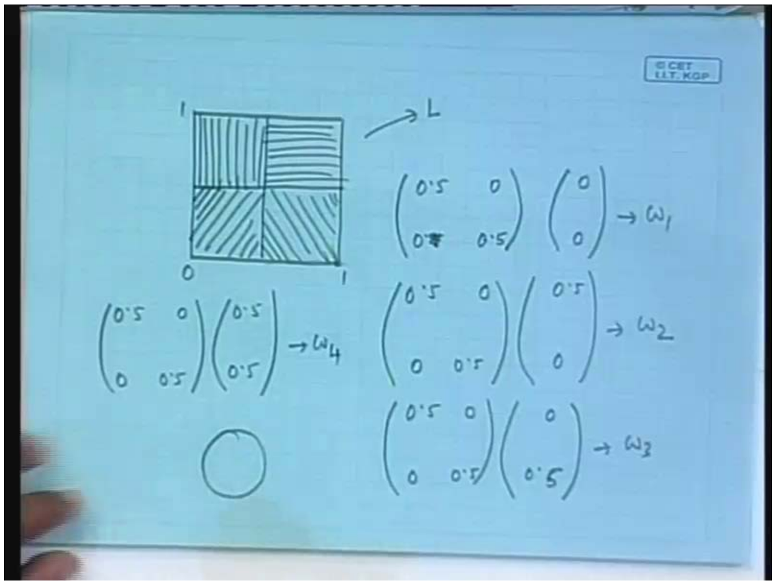
where h is the Hausdroff metric. Then

$$h(L, A) \leq \epsilon/(1-s)$$

Where A is the attractor of the IFS.

It says that this is the collage theorem, let (x, d) be a, L is the object that we are starting with, be given and let epsilon choose an IFS with contactivity factor. So that the Hausdroff distance between the object L and the union of W_n of L is less than epsilon where h is, then this is the theorem called the collage theorem. See what it says. Suppose you have given a fractal object like L say the fern, say the Sierpinski triangle, say the square be given and now you have to choose an IFS choose. We will do that with contactivity factor s so that the distance between L the original object and the union of all these applied to the same object L is less than epsilon. Then this theorem states that ultimately the distance between Hausdroff distance between the original set and the attractor of IFS will be smaller than this number. Now if you can make epsilon zero ultimately you will also be 0 let’s see how to apply this.

(Refer Slide Time: 00:20:18 min)



$$\begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \end{pmatrix} \to \omega_1$$

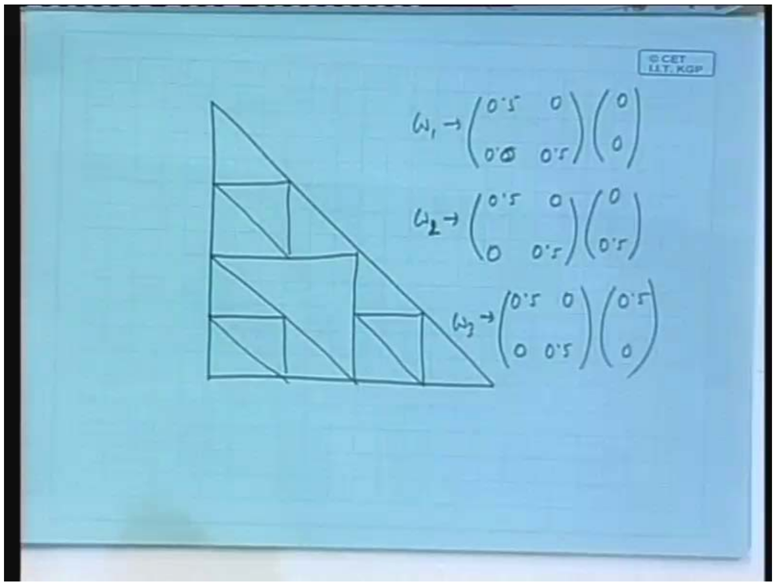
$$\begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} 0.5 \\ 0 \end{pmatrix} \to \omega_2$$

$$\begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} 0 \\ 0.5 \end{pmatrix} \to \omega_3$$

$$\begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} \to \omega_4$$

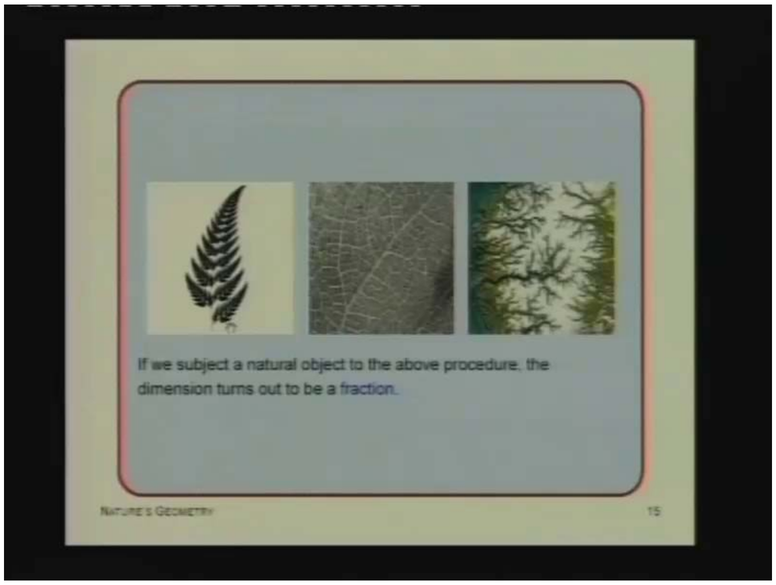
Suppose I want to generate a square which is not a fractal but nevertheless this can generate any image. So let’s say I want to generate a square. So this is my L, now I have to choose the IFS that means the w_0 w_1 w_2 so on and so forth in such a way that the contractivity fractal is less than one but at the same time after the first iterate that means after having applied these w’s on the same set L this should be smaller than epsilon. How can you do that? Simple, you break this square into 4 and design the affine transformation such that if first affine transformation generates these. The second one generates that, third one generates this and the fourth one generate this. Can you do that? Very trivial say this is 0101, from the whole one from the whole square, how to write down the affine transformation that will generate this 0.5 0.5 and here could be 0 for this one. When you want to generate this one it will be same thing shift that by 0.5 so 0.5 0 0 0.5 shift it along the x axis. Now you want to generate this, shifted along the y axis this is and let me write here. So abcdef this is w_4 so all these taken together will define my IFS and then this theorem will assort that even if you start with a circle and keep on applying this it will ultimately converge on to this square. How to generate the Sierpinski triangle?

(Refer Slide Time: 00:23:38 min)


© CET I.I.T. KGP
$\omega_1 \rightarrow \begin{pmatrix} 0.5 & 0 \\ 0.0 & 0.5 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \end{pmatrix}$
$\omega_2 \rightarrow \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} 0 \\ 0.5 \end{pmatrix}$
$\omega_3 \rightarrow \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} 0.5 \\ 0 \end{pmatrix}$

How to generate this? This will of course be further sub divided but nevertheless point is that you notice that the whole we are trying to obtain a particular transformation, affine transformation which generates only this part from the whole. So which transformation when applied on the whole with generate only this part, the last point. Say one actually so your W_1 would be 0.5 0 0 0.5 and how we will generate this one? What we did actually but the fourth one will not be there and that is what makes a difference between a square and Seripinski triangle this is what we had now to check. This is what we had given. Now you notice this do you understand why numbers? game very simple all that we are doing is that we are asking ourselves, which transformation when apply on the whole image will give me part of it and then we generate this part and make a collage of it if that collage becomes very close to the original image, than we have through. That is what we have doing, how to generate the fern. Let us look at the fern image first. Let us look at it.

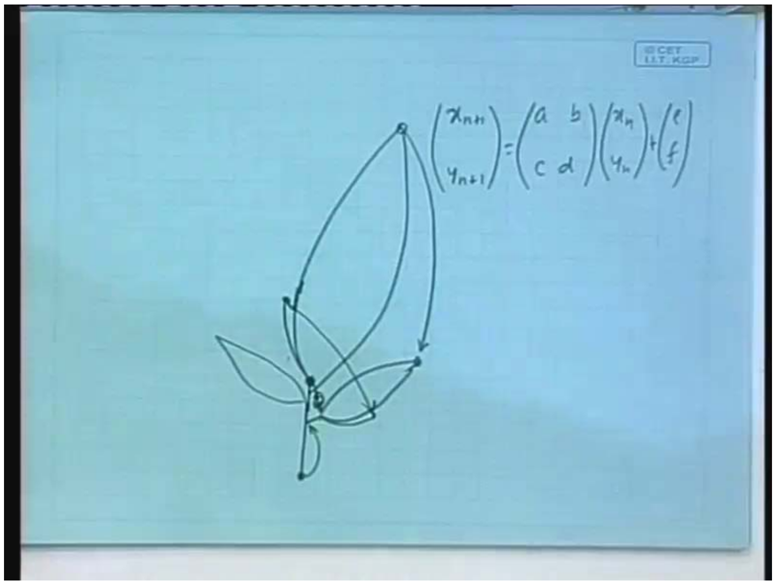
(Refer Slide Time: 00:27:22 min)


If we subject a natural object to the above procedure, the dimension turns out to be a fraction.

NATURE'S GEOMETRY
15

Look at this image of the font can you see that? Here you notice that in this whole image of this fern there is a stem, can you see that? there is a part to the right here, there is a part to the left and there are many of this parts but it’s not difficult to see that this whole is very self similar to this part, is very self similar to the whole fractal. So if we can somehow define the affine transformation that will generate this right part, this left part this stem and this whole thing than we have through. Now let us come back to this screen here.

(Refer Slide Time: 00:28:08 min)

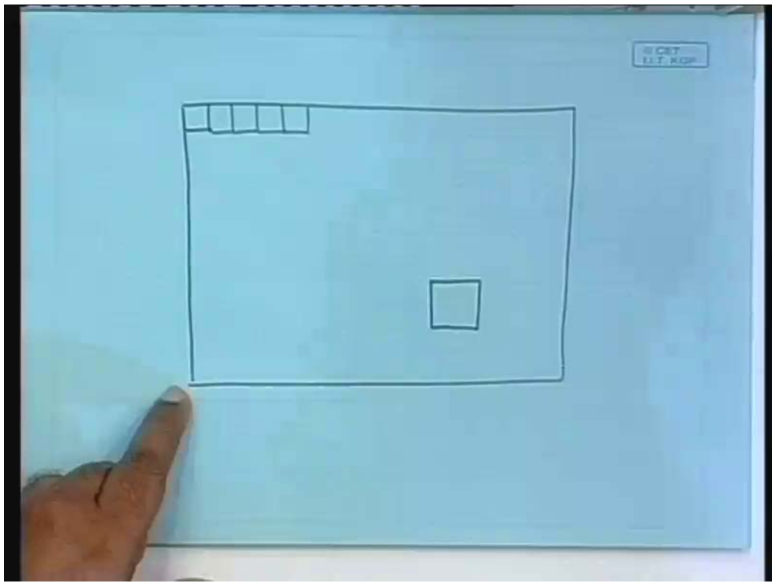

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

Here we have a stem, we are something to the left part, we have something right part and we have something to… So we will need to find this. let us tackle the first problem, that means the whole how can you transform in such a way that means how can it define one affine transform that will bring the whole into this. For this since the structure is like this X_{n+1} Y_{n+1}= (abcd) (X_n) (Y_n) + (e f) this structure is like this. So you notice that if I define two points which one maps to which point then it defines two equations but there are 6 unknown and therefore we actually need to find out 3 points and say that this point should map to this one, that point should map to this one and third point should map to third one. If you can define this in three points then we have got 6 equation 6 unknown case and we can solve it right, so now let us come back to the to the screen. Let me increase the size then it will be clear. Now we are looking at something else. No leave it. It is not difficult see that this point should map to this point, the way the fractal is oriented there is one end here which is actually mapping to the end here, so you get another estimating and the third one is the starting point.

This is starting point of the whole fractal that should map to this one. So this way for this particular part, we are able to identify 3 points of the original full fractal object which when affinely transform will give the smaller one. Then we get this number. How? all we need to do to put that fractal object on the paper and put a graph paper on to that, identify those point as numbers and put here then we can solve this that ah IFS will come on similarly we have to do this one, this one, this one, this one and is not difficult to see notice you just get back to the fractal that we had. the IFS that I gave you last time for the fern, you notice that this is the first one can you identify what it means a 0, b 0, c 0 and d only has value which means only one y coordinate will be there and that to the whole height will be shrunk by a fractal 0.16 and put here right, here zero there is no translation. What does it do? it generate the stem you notice that this one is a big one because it has large probability, big one is which one the big part and this two are the two smaller one's, that is how it is been generated and you could also generate. all you need to do is to simply put the fern image, graph paper on top of that, identify the points simply find which point map to which points, put into this equations solve them and that is how this point have got. That is not god gift somebody obtained it clear.

this idea has been extrapolated, the idea is that see the fern itself is the very complicated structure very complicated image, is not easy to draw is very complicated structure but still it is represented by only this few number something like 24 numbers, if we add this it would be 28 numbers. So 28 numbers only this much of information necessary in order to codify that complicated object. Can we then generalized this idea? that means can we than codify any image in terms of such simple numbers that is the basic idea of what is known as fractal image computation. So that has been applied to image computation where the basic idea is that how to obtain these numbers which when iteratively apply will automatically generate that. How to do that? Notice when we talked about the collage theorem, did we ever really need the concept that this individual affine transforms should actually apply on the whole image. We really didn't mean, we could as well apply that to the part of the images to generate part of the image ultimately we are make it to collage that's all and the collage can also be generated from a part of the image. if we use the whole image and generate a part then we do not need the information regarding which part is it coming from but if we affinely transform a part to a part than, we need the information additionally to which part I am talking about but nevertheless it is not necessary to transform the whole image. The idea is suppose you are trying to codify my face. We can, it is actually possible to find out the number that will be the IFS for my face, your face and then we can start from somebody else face and then apply that you will see the more, it will come to your face. It actually happens I mean you can also do that coding the point is that of course I cannot use the whole image of my face to generate a part, this part is not definitely the whole image transform to be a affined way no is not but then my left cheek is the affined transformation my right cheek. A part of the sky is affined transformation of another part of the sky, a part of a bush is can be seen as a affined transformation of another part of the bush that means normally in any image there is a lot of affined redundance. In any image there is a lot of affined redundance and that affined redundance can be made the use of to obtain the ideas. How would you do that? Suppose you have got an image.

(Refer Slide Time: 00:37:34 min)

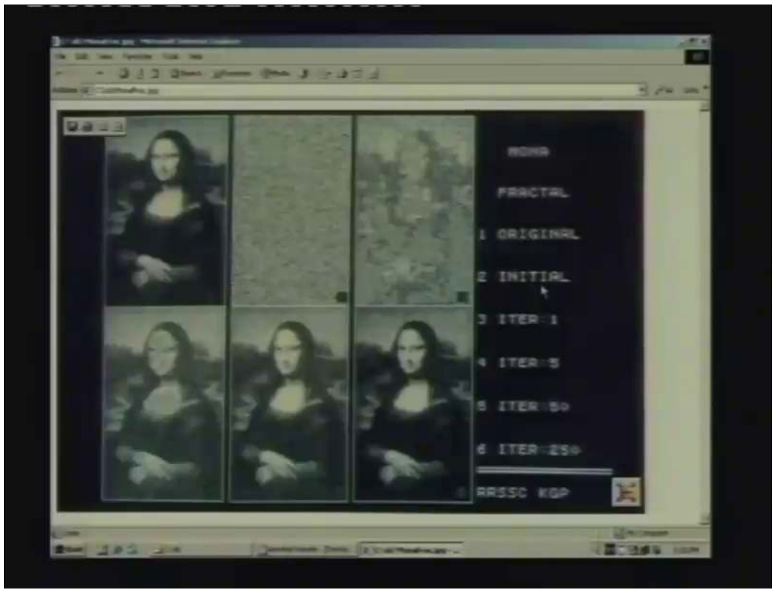


Pretty large image and it has various things, there would be a part of the sky, there would be trees there would be bushes, there would be grass, there would be cows, there would be people and than our object if than is to look for those affined redundancies. How we can we do that? We will divide this image into blocks, these are all blocks. Suppose I start from this block then my question is that can this block be obtain from any other part of the image through an affined transformation. This when we look for the affine transformation it has to have a contractivity fractal less than one and therefore what we look for is a larger block, has to be a larger block. So say we search the image and find that okay when this particular part is affinely transform it gives this particular collection of this pixels. Again we come here search the whole image to find out which part it comes from. normally this routines is written which particular size of this blocks say 8 pixels by 8 pixels block or 16 pixels by 16 pixels blocks and then if it is 8 pixels by 8 pixels blocks then we are looking for 16 by 16 pixels blocks in the whole image from where it gives the best possible fit. Notice the theorems says about the epsilon it does not require to be exact fit, if it is approximate its fine. So you really want to try to get the fit so for each one ah these are the domain blocks for each domain blocks you are looking for the rest of the image to

find out which larger blocks when affinely transform gives this block. If these are square boxes there are only a few affined transformation that are possible.
You can shrink bring it here, you can turn it, twist it, flip it, these are the possibilities. So it will be only a small number of collection that can be seen from a lookup table which one is the best you find out and then assign there. So for this particular block, you locate the particular source and locate that particular transformation that gives here, enough. go to the next one, go to the next, go to the next one, ultimately you have codify the image has a collection of numbers which represent which affine transformation is to be applied and where is to be applied. Yes, that will take a lot of time. That is way that's take a lot of time because for each one you are searching the whole image. That is way this is very competition intensive, the compression stage is very competition intensive, what people have done is that since searching for this one and searching for this one and searching for this one they can proceed parallel. Normally this is done using parallel processing.

So initially they used to be done by parallel processing now parallelized chips are available, specially designed chips to do this work. There is a company called iterated function, the fellow whose theorem these are Michel Bernsley he opened a company called iterated systems incorporated. So they have made those chips so they are now available and you can codify any image just like that, it's not difficult. Now when you want to retreat the image from the information that has been codified what we have to do? Start for any image what so ever and then again start from those bigger blocks, you know the transformer that is to be apply you get the smaller box and then you get the collage. After you have done so you got the first iterate go on applying those iterates, the theorem says that it must converge on to the image that you want. for example right here in IIT Kharagpur we had, this is a horrible slow machine so it may take hours but nevertheless. Just wait to see some of the results. Sir what about the colors? Color will come later. So far we were talking about binary images. Let us first talk in terms of the gray scale then will come to color. Now the gray scale can be treated as one depth level, the gray scale is actually this is (Not Audible) (00:43:39).

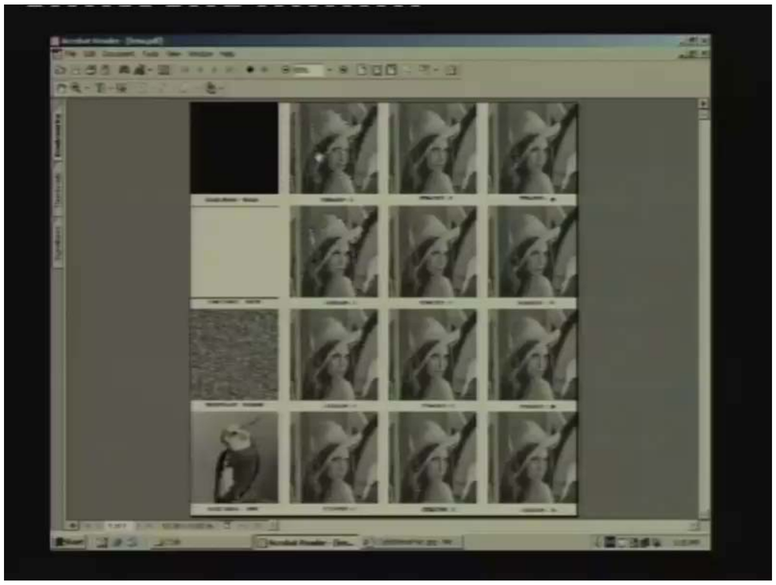
(Refer Slide Time: 00:43:41 min)



See here is the image of you know who Monalisa and that image was codified here in IIT Kharagpur and you start from any arbitrary image in this case a random image and this is after the first iteration, this is after 5 iteration, this is after 50 iteration, this is after 250 iteration, you get the image back. So this is essentially on the application of the fractal image compression on the Monalisa image, done here in IIT Kharagpur. Similarly it can be done in all possible images very interesting but it's a bit computation intensive as I told you, need parallel processing in order to do it in a reasonable time clear. Now here you can see that the contains gray scale. Gray scale means earlier we have considered each pixel is either 0 or 1, either filled or empty but now it will have a gray scale value assigned. then gray scale value can be imagine as either as a depth level that means the whole problem because not a 2 D problem but a 3 D problem that's all no further difference really, can be done.

The second way of doing it is that for each pixel we have got a number, earlier it may binary zero and one now we have sixty four numbers assigned you to it one of those sixty four numbers. Ultimately we have got a 8 by 8 block that we are trying to obtain by affinely transforming from a the lager place but here there are sixteen by sixteen pixels, each with a gray scale value. You can apply the affine transformation, when you apply the affine transformation you get gray scale values of those obtained 8 by 8 blocks and you can still find out the distance between the two, distance between not this distance but the actual a gray scale value the metrics of gray scale values in this block and the one that you get by transformation. Thereby you can find out the distance is possible to find that is another way. You scan the whole thing so that you get a minimum distance is fine. Then it will yield the same thing start from any arbitrary image it will converge on to the ultimate thing. There are a few orders that we have provided this computer is horribly slow. So it's taking a long time but nevertheless ultimately it will come you see that.

(Refer Slide Time: 00:46:59 min)



So can you see those images see it’s a standard image that is used for in a processing the Lena image and here it started from a square and you see the transformation and it ultimately converge on to this, convergence actually converge on to the Lena image starting from a white still does a same thing, starting from a green and starting by parrot, still does the same thing. So actually you can you can see the transformation (Not Audible) (00:47:39 min).

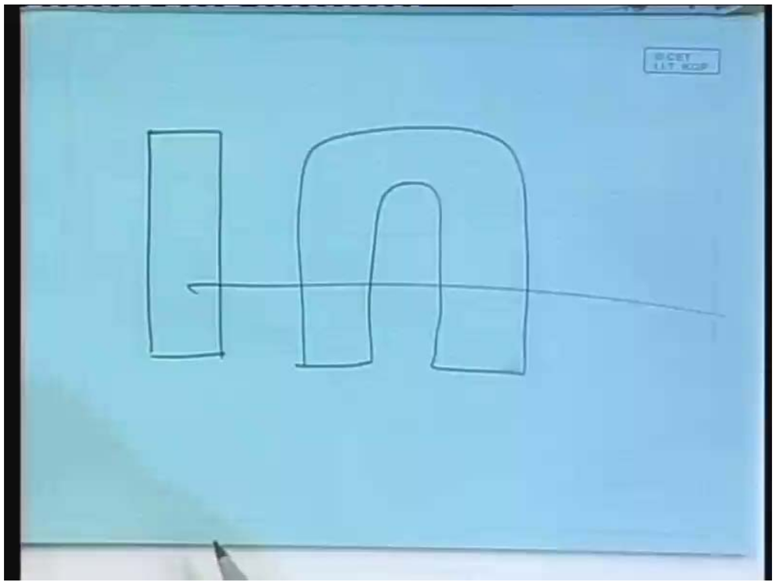
[Conversation between Student and Professor – Not audible ((00:47:40 min))] no the decompression is very fast [Conversation between Student and Professor – Not audible ((00:47:45 min))] that depends on the speed of the computer. what [Conversation between Student and Professor – Not audible ((00:47:50 min))]

Like four hours, doable time. it was some years back when the computers are slow I don’t know how much time it will take to this computers but say four linux machines clustered together that will do in the short time but once you understand the logic, it’s not difficult to write the code at all. it’s very simple to write the code and do it your selves but since the logic is elegant the mathematics is elegant but nevertheless this has not become the standard in image competition, yet the reason is that the discreet cosine transformed that is standard in image competition today that is faster when it comes to compression stage and that is why still this is in an experiment stage it requires really good processing power in order to do the coding part that means the first if I give you the IFS what will you do? simply start from a arbitrary image, take larger blocks and since the information that is available to will have, will say that this particular black will transform to there from where to where that should be available in the code and then which transform to apply that should also be available in the code so that you transform it, the next one you transform it, the next one you transform, ultimately you got a whole image. Again apply the same logic, keep on applying the same logic.

So today with this we are coming to a close of the chapter of the factors and then we will go back to the study of the nonlinear. essential think that we have to remember, these are all fun i should say this are very nice things to do but essentially these are fun and one should do a lot of get a lot of fun out of learning. So this is also lighter side but the relationship between what we were studying the nonlinear dynamics and factors should be clear. Firstly while we are talking about the Mandelbrot set and the Julia set we had made some points. the point was that the Mandelbrot set essentially is a fractal in the parameter space and the Julia set is essentially factor in the state space, so when we see a fractal object in the state space while we are studying non linear dynamics and chaos series and stuff like that, essentially you will remember that it follows the same logic as is followed to generative Julia set and the Mandelbrot set idea also tells you that if a system is given by a few parameters for example electrical circuits are given by the inductances, capacitances, voltage sources applied that these are the parameters the variables are the voltages across capacitor and the currents the inductances. Now you might also you imagine a space of those parameters and based on this knowledge we should expect the state of parameters for which you get a particular type of dynamical behavior to be a fractal object. A fractal object means or therefore the point is that we should expect fractals both in the parameters space as well as in the state space.

We will later us see that the basin of attraction in the state space can be fractals that means there is an attractor and there is a basin of attraction. The set of initial conditions for which the system holds on to that attractor that state of initial condition that can be a fractal and if it is fractal there can be fractal fingers from one type of attractor getting in to the basin of attraction of another type of fractal. Do you understand the kind of complexity that it can lead to? that means if we are in a particular initial condition they can be in the neighborhood a fractal finger of another attractor coming in and those things do happen and the idea about those things come from the understanding fractals. Most importantly when we talked about that the process of generations of chaos, we talked about the horseshoe mapping, snail horseshoe mapping imagine, so we said that in the state space of the system there is a snail horseshoe mapping in action. The state space is like layered there is a continuous stretching and folding and that could be mimic by this snail horseshoe mapping.

(Refer Slide Time: 00:53:28 min)



Imagine that you have got the snail horseshoe mapping that means you take stretch it and fold it you then again stretch it and fold it and stretch it and fold it stretch it and fold it ultimately if you get a cut here. What do you see on this line, just try to thing this this particular point canter set. Ultimately on this line if you slice it then you see canter set and this is how all those ideas about fractals become really when in the study of non linear dynamics, why we need to understand fractals. We made of need to understand the IFS and all that but the essential idea we do need to understand. So that is how the study of fractals becomes important. The other thing is that if a system is chaotic, we do expect a fractal object like the canter set to appear in it’s a state space and the fractals are given by fractal dimensions. Therefore in the state space of a chaotic system we should be able to obtain the fractal dimension to characterize that object. Yes, there is one of the ways of characterizing chaotic behavior to obtain the fractal dimension. So that’s all today.