2004

Acoustic signature for chaos and fractal geometries

Obadiah M. Kilonzo

ACOUSTIC SIGNATURE FOR CHAOS AND FRACTAL GEOMETRIES

OBADIAH M. KILONZO

This is a thesis submitted in partial fulfillment of the requirements for the degree of **Master of Science** in **Mechanical Engineering**

Department of Mechanical Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology

Approved by:

J.S Török — J. S. Török
Rochester Institute of Technology (Thesis Advisor)

A. Crassidis — Agamemnon Crassidis
Rochester Institute of Technology

B. Varela — Benjamin Varela
Rochester Institute of Technology

E. Hensel — Edward Hensel
Rochester Institute of Technology (Department Chairman)

November 2004

Permission Grant

I, Obadiah M. Kilonzo, hereby grant permission to the Wallace Memorial Library of the Rochester Institute of Technology to reproduce my thesis in whole or part, as long as no reproductions will be used for commercial use or profit.

Permission granted by: Obadiah M. Kilonzo

Obadiah M.Kilonzo (Author of thesis)

Date: 12/07/04

ACKNOWLEDGEMENTS

I would like to thank my parents for encouraging me during my stay at RIT.

I would like to thank Dr. Steve Mutunga, Dr. Lawrence Agbezuge, Geoffrey Musau and Charles Mirango and Ivan Delgado for their encouragement through my stay at RIT.

I would like also to thank Dr. Josef Török for all that he has taught me in and out of class. I thank him for his guidance and direction in pursuing my degree. I want to thank him for the encouragement to have one of the important building blocks for engineers - an open mind. Thanks too for the help in editing my thesis.

I would also like to thank my classmates who I have worked with. Thanks for your encouragement.

Thanks to the faculty of the Department of Mechanical Engineering and Kate Gleason College of Engineering for the effort to provide a conducive environment for learning. I would also like to thank the staff of Mechanical Engineering Department and Kate Gleason College of Engineering for keeping the learning environment running.

ABSTRACT

One discipline of study that has been coming up in the recent years is the study in the behavior of nonlinear systems. Most of these exhibit chaos traits and this has spurred much interest. Fractal geometries, which are as a result of chaos behavior, have been more feasible to research on, with the recent computer technology.

Most of these fractal behaviors can be mapped into the sound domain. This 'sound domain' is referred to as acoustic signature. This thesis majors on a way to map out the fractals to the sound domain without much change in the parameters that define the fractal. Some of these parameters include position of the individual points on the drawing axis and the way a fractal appears in form of a color map.

Due to their dependence on initial conditions sometimes they may look similar and hence a method is needed that can distinguish them. The different types of fractals are mapped in different ways. Some of these ways involve producing an audio wave (known as wav file) that is further converted to MIDI.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF CHARTS

INTRODUCTION

Computers are one our most important tools for creational reasoning. The beauty and importance of computers lies mainly in their usefulness as a tool for reasoning, creating and discovering. All around us we are surrounded by creatures and things that have motion or states of behavior .The study of these behaviors that leads to evolving of states is what makes systems. Some of the behaviors can be predicted but some cannot. The reason why most of these cannot be predicted is due to the fact that they are sensitive to initial conditions. The behavior they project is what is known as chaotic behavior or simply 'chaos'.

Chaotic behaviors are bounded, and it has been discovered that some display patterns in and within themselves;-such that a behavioral picture is made up of different parts which are similar to each other upon magnification and it's these pictures that are called fractals. These can be further mapped into the sound domain to get an acoustic signature. There is a broader term that is used for mapping fractals into sound domain and its 'fractal music.' Basically there are two definitions of fractal music. One is where upon scrutiny of a piece of music one is able to see self-similarity or the 'fractal' behavior, or one can create a piece of music that embeds self similarity in it. This has no relation to the fractal images that are as a result of behavior displayed by chaos.

The other definition of fractal music is where a property of a fractal image is obtained from a fractal algorithm and it's mapped to music such that every particular image has its own version of sound. This is like labeling every fractal .This is what is referred to as 'acoustic signature'.

Just like the way every human is expected to have a particular signature or way of identifying oneself, each fractal will have its own identification.

This thesis is dealing with the latter definition of fractal music. The objective is to obtain an acoustic signature for chaos and attractors.

The algorithms used are in created in MATLAB .The program maps out the fractals into the sound domain in MIDI form.

1 LITERATURE SURVEY

1.1 DYNAMICAL SYSTEMS

Different systems around us involve dynamics and hence create the interest to study them. [15] Some include:

- population growth
- variations on exponential growth e.g. radioactive decay
- competition and predators in nature
- Motion of living creatures, e.g. the flashing of fireflies.

The are basically two broad divisions of dynamical systems

- Discrete –time dynamical systems
- Continuous –time dynamical systems

A *discrete –time dynamical system* consists of a space X and a one parameter family of maps of $\{f^t : X \to X\}, t \in R$ or $t \in R^+$ that forms a one parameter group. [16]
The system is called a flow if the time t ranges over R and a semi flow if t ranges over R^{+}.
An example of a discrete time dynamical system is a map, which can be written in vector form as

$$X_{n+1} = M(X_n)$$

X_n has N components:

$$X_n = (x^{(1)}, x^{(2)} \ldots x^{(N)})$$

Hence the different values are obtained at different components.

An example of a *continuous-time dynamical system* is a system of N first order autonomous, ordinary differential equations; [23]

$$\frac{dx^{(N)}}{dt} = F_N(x^{(1)}, x^{(2)} \ldots x^{(N)})$$

This is often written in vector form as

$$\frac{dx(t)}{dt} = F[x(t)]$$

x is an N-dimensional vector.

Hence for an initial state of the system, we can in principle solve the equations to obtain the future system state, $x(t)$.

The other broader division of systems in terms of motion is *linear* and *nonlinear* systems. Linear systems can be used to approximate the behavior and study how nonlinear systems behave. The key ingredient is differentiability i.e. the existence of a good linear approximation near any given point.

1.2 CHAOS

The simplest nonlinear chaotic dynamical systems in dimension one are the quadratic maps [16], $q_n(x) = rx(1-x), r \geq 4$

Liapunov exponents:

Liapunov exponents are a generalization of the eigenvalues of a dynamical system at an equilibrium point. They are used to determine the stability of any type of steady state behavior including chaotic solutions. Liapunov exponents measure the infinitesimal exponential rate at which nearby orbits are moving apart, or the measure of exponential growth rate of tangent vectors along orbits. [16,17].

Considering the case of a discrete time system e.g. the cantor set dimension one , for each point $x0$ the liapunov exponent $\lambda(x0)$ is defined as follows:

$$\lambda(\mathrm{x0}) = \lim_{\mathrm{n}\to\infty} \frac{1}{n}\left(\log(|f_{(n)}'(x0)|)\right)$$
$$= \lim_{n\to\infty} \frac{1}{n}\sum_{j=0}^{n-1} \log\left(|f'(x_j)|\right) \;, x_j = f^{j}(x0).$$

For continuous systems in the form of first order differential equations like

$\frac{dx}{dt} = Fx$, the liapunov exponent is defined as

$$\lambda(x0,u0) = \lim_{t\to\infty} \frac{1}{t}\left(\ln\left[\frac{|y(t)|}{|y(0)|}\right]\right)$$

These liapunov exponents can be used to calculate an approximate fractal dimension of an attractor. [44].

One of the differences between chaos and fractals is that chaos is a condition of extreme unpredictability occurring in a dynamical system and fractality is a condition of extreme irregularity or ruggedness in a geometric configuration. [5]
Let (X,d) be a metric space, and let $f_n : X \to X$ be a function. The map is chaotic provided that

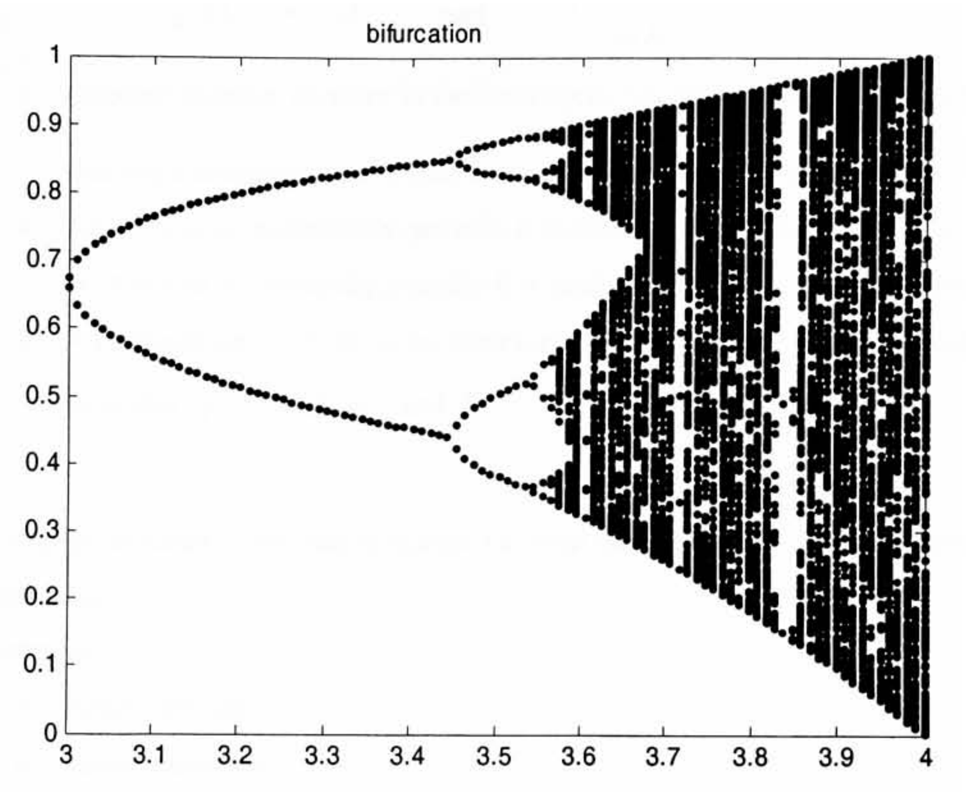
- f has sensitive dependence on initial conditions
- f is transitive(For two open sets U and V, $f_{(n)}$ U is an area that intersects U andV
- Periodic points of f are dense in X.

Topological entropy is defined as a quantitative measurement of how chaotic a map is by determining how many "different orbits" there are for a given map (or flow)[17].The human mind, when aided by numbers and symbols is capable of expressing and understanding concepts of great complexity.[1]
An example of chaos in a logistic map:

BIFUR_MAP.m

```
% Bifurcation Diagram for the Logistic Map
%
r0 = 3;
r = linspace(3,4,100);
y = zeros(250,100);
for nn = 1:100

x(1)=.45;
%
for k = 1:1000
  x(k+1)=r(nn)*x(k)*(1-x(k));
end;
```

```
y(:,nn) = [x(751:1000)]';

end
plot(r,y','k.')  %  The columns of y , or the rows of y' are plotted vs. r
```



Bifurcation seen on its way to chaos!

1.3 ATTRACTORS

Attractors by observation in nature: [21]

- Moths to a light/ bees to a flower.
- Magnetic attraction/ gravitation
- Cauchy sequence/ limit sequence
- Convergence /divergence

An attracting fixed point which is a simple attractor can be defined below:-

- A fixed point p of a map $f_n : X \to X$ of a metric space is said to be an attracting fixed point if there is a neighborhood U of p such that

$$f(U) \subset U \quad \text{and} \quad \cap_{n \in \bar{N}} f^n(U) = \{p\}$$

- Another familiar attractor is the limit cycle, which is defined as a periodic point p that has a neighborhood whose every point is positively asymptotic to $\mathcal{O}$p.[1]
- An orbit is asymptotically periodic if it converges to a periodic orbit as $n \to \infty$ and is eventually periodic if it lands precisely on a periodic orbit.[25]
- A compact set $C \subset U$ is an attractor if there is an open set U containing C, such that $f(U) \subset U$ and $C = \cap_{n \geq 0} f^n(U)$

A Strange attractor is one that is chaotic i.e. with sensitive dependence on initial conditions.

Examples –

- Henon attractor
- Lorenz attractor
- Rossler attractor
- Ueda attractor

Another definition of an attractor is a set that contains a dense orbit of a function f. The existence of such an orbit ensures that the attracting set behaves as a single unit that cannot be decomposed into a collection of disjoint subsets that are themselves attractors. [17, 18, 19]

ROSSLER ATTRACTOR

Credited to Otto Rossler

The equations that govern the attractor are basically:

$$\dot{x} = -y - z$$

$$\dot{y} = x + a * y$$

$$\dot{z} = b + z * (x - c)$$

Where a = 0.2, b = 0.2, c = 5.7

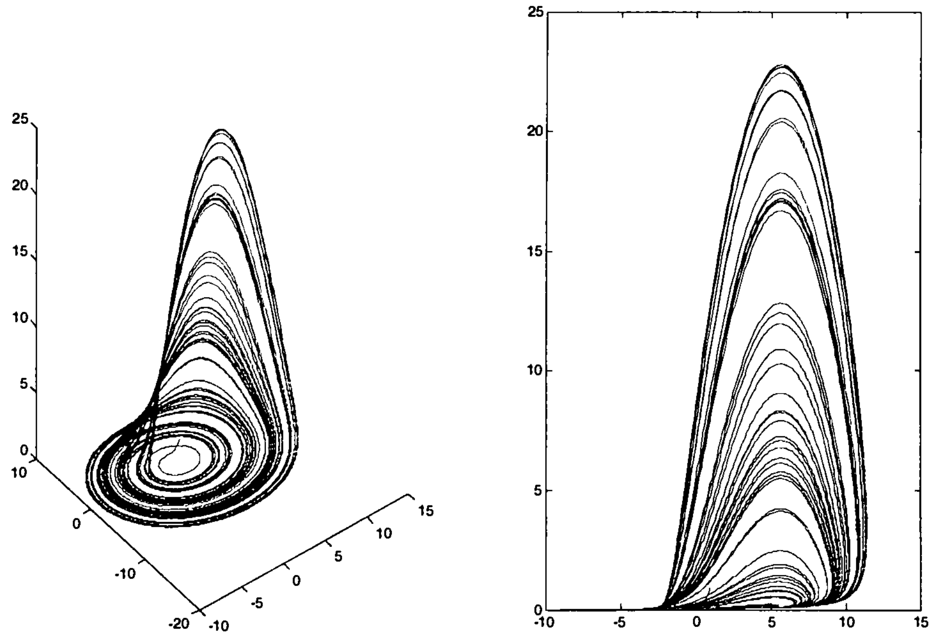
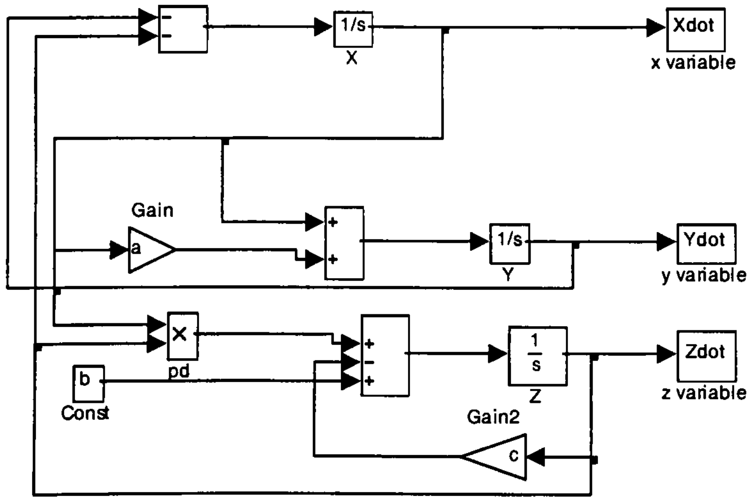


fig 1.



UEDA ATTRACTOR

$$\dot{x} = y$$
$$\dot{y} = -x^3 - k * y + B * \sin z$$
$$\dot{z} = 1$$

B = 7.5, k = 0.05

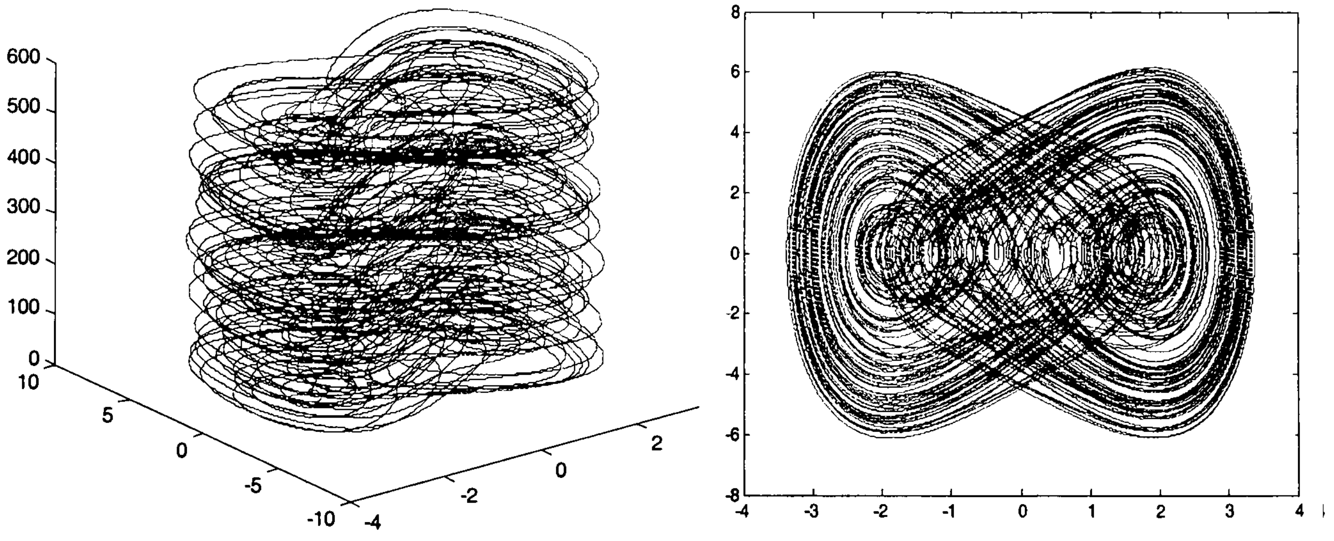
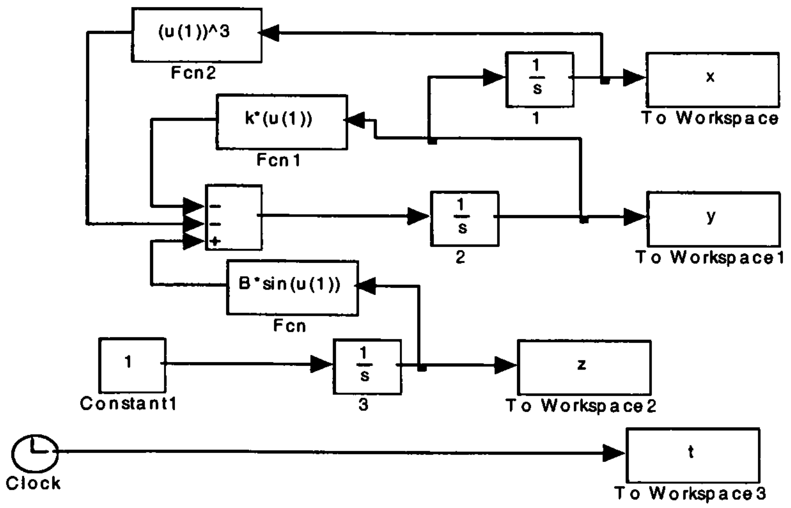


fig 2.



LORENZ ATTRACTOR

$$\dot{x} = \sigma * (y - x)$$
$$\dot{y} = -x * z + r * x - y$$
$$\dot{z} = -b * z + x * y$$

$\sigma = 10$; b = (8/3); r = 25.3;

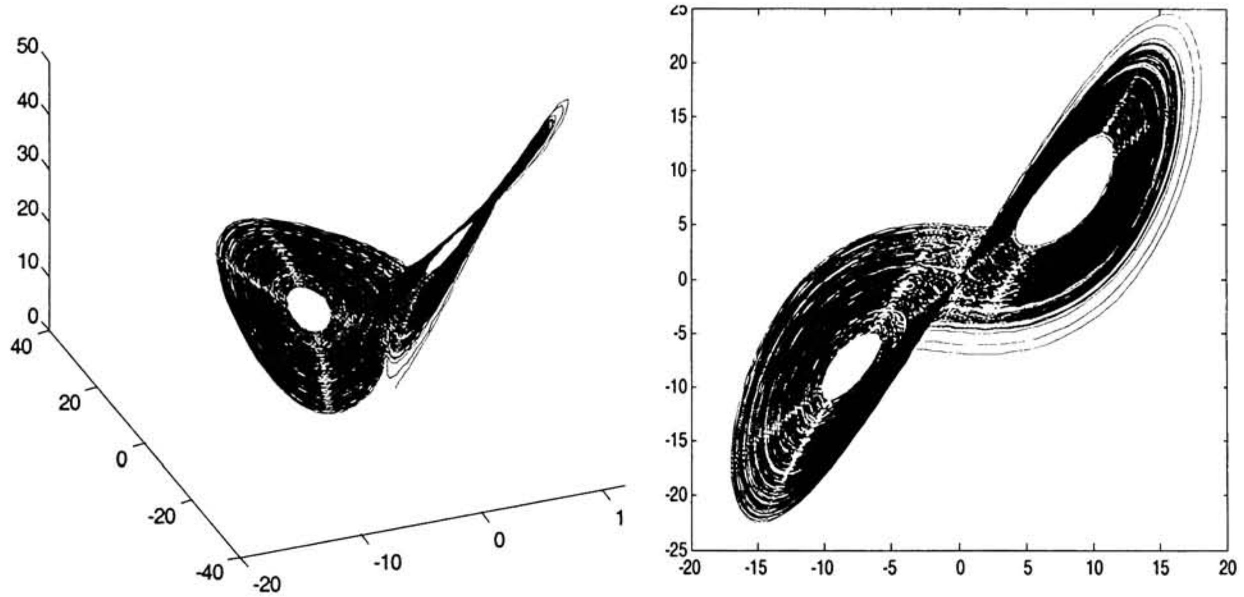
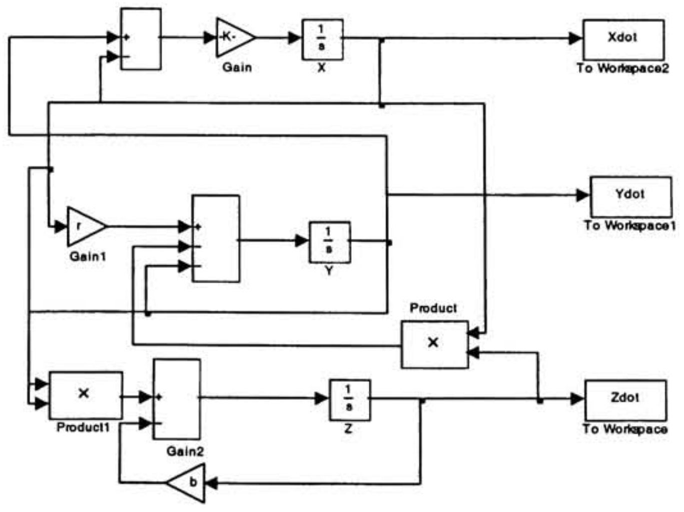


fig 3.



HENON ATTRACTOR

$$\dot{x} = 1 - A * x^2 + y$$

$$\dot{y} = B * x$$

A = 1.4 , B = 0.3

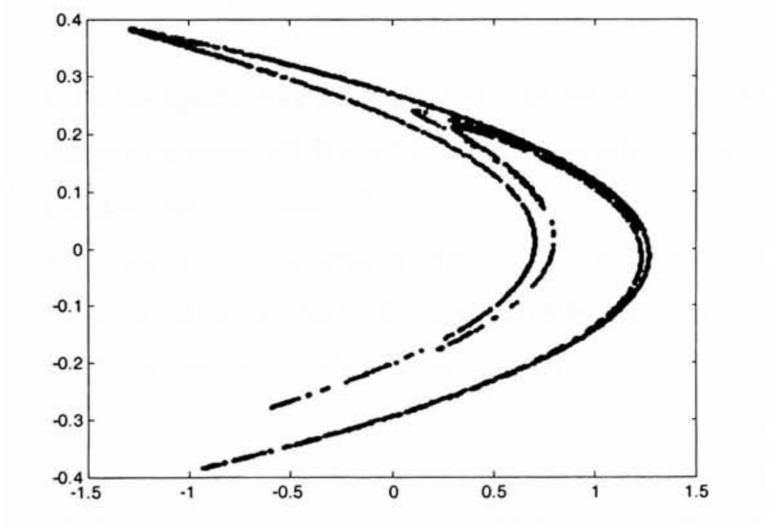


fig 4.

The Henon model is given by:

$$X_{n+1} = a * x_n - b(y_n - x_n^{\ 2})$$

$$Y_{n+1} = b * x_n + a(y_n - x_n^{\ 2})$$

Here $a = \cos\alpha$ and $b = \sin\alpha$.

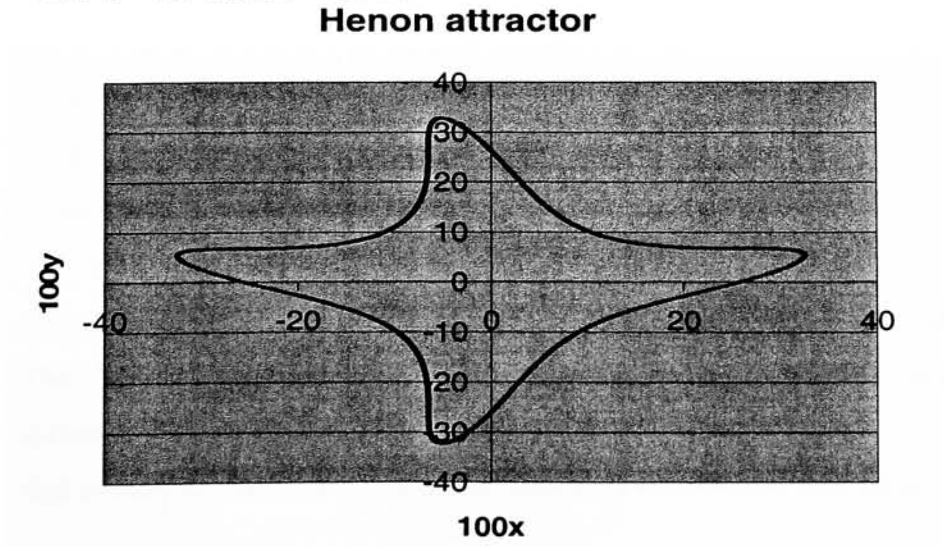


fig 5

1.4 FRACTALS

DEFINITIONS & FRACTAL BASICS:

A fractal is a geometric shape that has 2 special properties: [3]

- The object is self similar
- The object has fractional dimensions.

A fractal applies to a particular static geometry configuration such as a freeze – frame image of a waterfall. It can be described as self similar if it can be decomposed into smaller copies of itself. [5, 11]
The smaller units are similar to the whole and share the same units of measurement. An ideal fractal surface looks the same at all magnifications. They can also be defined as sets that have non integer dimensions. [14,15].

Fractal geometry builds complex objects by applying processes to complex building blocks , for example - recursion , which involves the echoing a simple rule over and over again.
A geometric object is called self-affine if it may be written as a union of rescaled copies of itself where the rescaling may be dependent on the direction.

Examples of self similarity: [21]

- The head of the cauliflower,
- peeling an onion,
- head of lettuce ,
- The distance between clouds.

A self similar image has smaller pieces that are similar to each other and to the whole. The parts must scale equally in all directions while for a self –affine image it scales differently in one direction than another.
Self affinity can be considered as self similarity with a skew or a shear.

Types of transforms:

- Scaling – this changes the size uniformly .More than 1 magnifies the size.
- Shear (known as affine transformation) is scaling that is uneven in different directions. In music it can be considered as a motive changed in part such as changing intervals in size while others remain the same.
- Translation (called glide or displacement) moves the image. In music it s compared to transposition.
- Reflection turns the image over, right or left, up or down. In music vertical reflection is called inversion and horizontal reflection is known as retrograde.
- Rotation turns the image around an axis. In music it refers to a melody that is changed into a chord or vice versa.

TYPES OF FRACTALS

There are basically two types of fractals:

- Regular
- Random

Regular fractals are those which display exact self similarity. These include simple objects such as line integrals, solid squares, solid cubes, and some complex shapes are like the Cantor set and Koch snowflake.
Regular fractals have their origin in the formulas for the area of a square and the volume of a cube.

A random fractal is an element of a set S which is closed under application of a renormalization formula or a group of renormalization formula.
Random fractals display a weaker, statistical version of self-similarity or self affinity.

1.5 APPLICATIONS

- Speech production uses generating functions which simulate the air pressure waves produced by the lungs. These pressure wave forms are then shaped by the filter functions in order to produce a more complicated speech like signal.
- The power spectral analysis of instrument sounds are sensitive to the periodicities in the melodic pitch sequence .The computer is not analyzing time waveforms but frequencies of the progression of musical score(which themselves represent fundamental frequencies of notes on a keyboard).The sounds are not analyzed. Parameters such as loudness, attack and timbre are left out.
- Study of carbon black particles, respiratory dust, cosmic particles.
- Study of metal grain shapes and size and also special metal crystals.

- Study of diesel soot
- Study of the structure of some types of sand grain.

2 FRACTAL GEOMETRIES

Fractal characterization

A fractal object has a shape with increasingly detailed features with increasing magnification, and examples include mountains and coastlines.

Shapes of nature can be characterized by a single number, the "fractal dimension" D. Mandelbrot suggested that the relationship between the measuring stick length (ε) and the apparent total length (L) of a coastline could be expressed by the parameter D, the fractal dimension. Let N be the number of measuring sticks, ε be the length of the measuring stick.

For a smooth curve such as a circle:-

$$N(\varepsilon) = \frac{c}{\varepsilon} \qquad \text{where } c \text{ is a constant.}$$

For a fractal curve the relationship is:-

$$N(\varepsilon) = \frac{c}{\varepsilon^D} \quad \text{and multiplying both sides by } \varepsilon \quad \text{the relation becomes}$$

$$L(\varepsilon) = \frac{\varepsilon}{\varepsilon^D} \quad \text{and } D \text{ can be a fraction.}$$

Since the coastline has bumps upon bumps as it is magnified, it tends to fill space and its dimension lies somewhere between a plane and a line. Fractals objects are not always self- similar at all scales.

Types of self similarity

An object is considered self-similar if its features (i.e. the general nature of its irregular bumpy surface) remain constant through successive magnifications.

Self similarity implies scaling similarity i.e. shapes are invariant under magnification. The edges of circles and lines are self similar since they look the same at different magnifications; however they are smooth hence not fractals. They possess standard scaling symmetry.

Objects with "bumpiness" but with scaling symmetry possess non standard scaling symmetry. The main focus in this thesis is to deal with objects that have non standard scaling symmetry.

Though irregularity continues with each magnification, the degree of roughness seems to fluctuate slightly when looking at isolated magnifications.

To estimate the values of D using the above equation, plots of log N vs. log ε are calculated and the slope determined by the least squares line fit to the data. For a speech signal, D seems to be unaffected by the pitch (fundamental frequency).

Self-similarity and its branches:

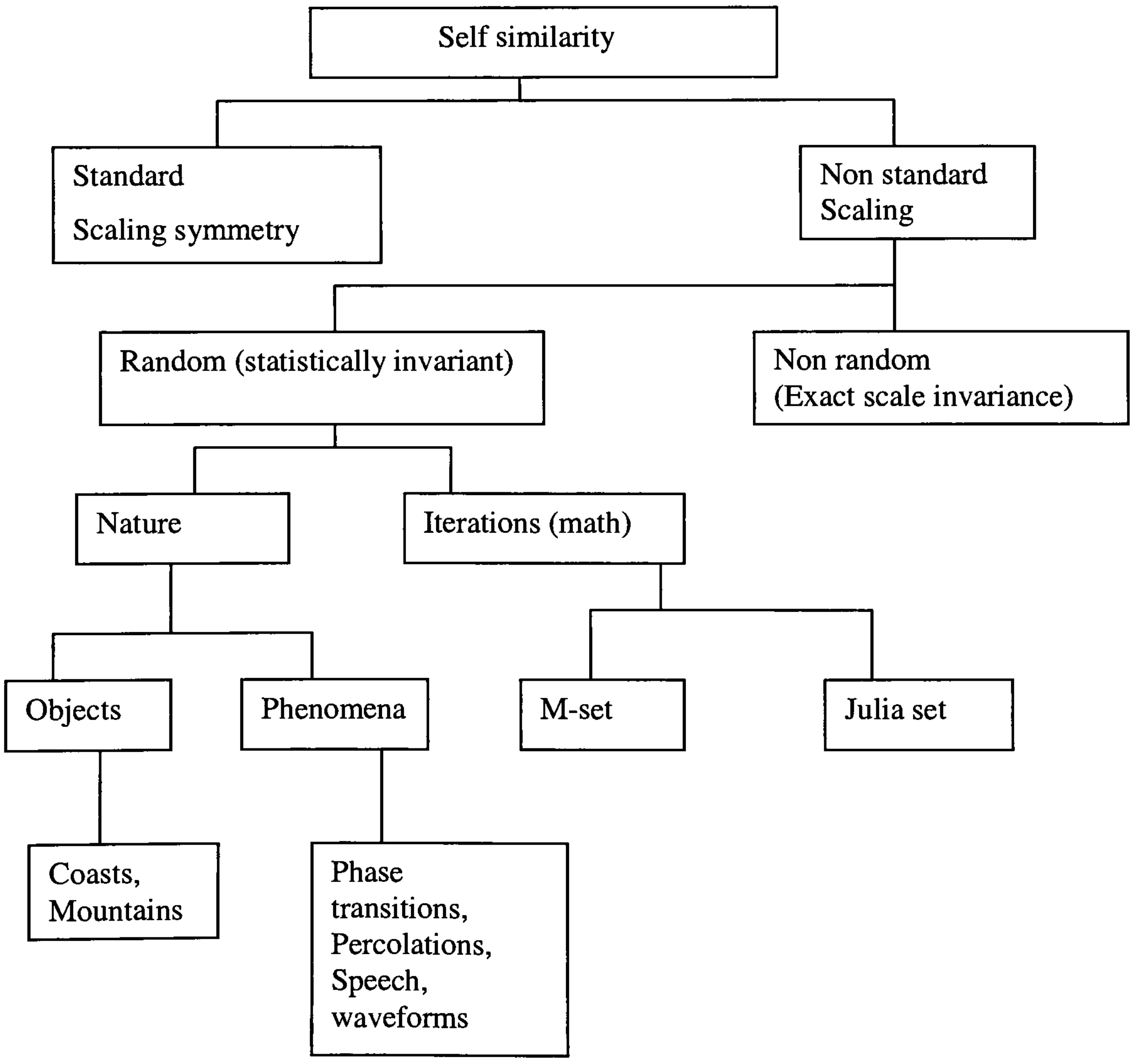

Self similarity
Standard Scaling symmetry
Non standard Scaling
Random (statistically invariant)
Non random (Exact scale invariance)
Nature
Iterations (math)
Objects
Phenomena
M-set
Julia set
Coasts, Mountains
Phase transitions, Percolations, Speech, waveforms

CHART 1

2.1 FRACTAL DIMENSIONS

One of the definitions of dimension is given below:-[19]

Let S be a set in R^n and let $N(S,\in)$ be the smallest number of closed balls of diameter$\in$, required to cover the set S. Then the dimension D of S is given by

$$D = \lim_{\in \to 0}\left(\frac{-\ln(N(S,\in)}{\ln(\in)}\right)$$

<u>Example</u>

For the Cantor set, $\hat{C} = \bigcap_{n=1}^{\infty} I_n$ with

$I_1 = [0,1], I_2 = [0, \tfrac{1}{3}] \cup [0, \tfrac{2}{3}], I_3 = [0, \tfrac{1}{9}] \cup [\tfrac{2}{9}, \tfrac{1}{3}] \cup [\tfrac{2}{3}, \tfrac{7}{9}] \cup [\tfrac{8}{9}, 1] \ldots$

$N(\hat{C}, \tfrac{1}{3}) = 2,$

$N(\hat{C}, \tfrac{1}{9}) = 4\ldots$ where N is the number.

$N(\hat{C}, \tfrac{1}{3^n}) = 2^n,$

$$D = \lim_{\in \to 0}\left(\frac{-\ln(N(\hat{C},\in)}{\ln(\in)}\right)$$

$$D = \lim_{n \to \infty}\left(\frac{-\ln(2^n)}{\ln(3^{-n})}\right) = \frac{\ln 2}{\ln 3}$$

Hence $\hat{C}$ has a fractal dimension less than one. (D=0.63)

Another method of characterizing the fractal dimension is given below by considering an island [13]

Let:

λ be the side of a polygon normalized with respect to maximum projected length of the profile.

P is the polygon perimeter of side λ normalized with respect to maximum projected length of the profile.

δ be the fractal dimension of the boundary .

L be the maximum projected length.

Given the island

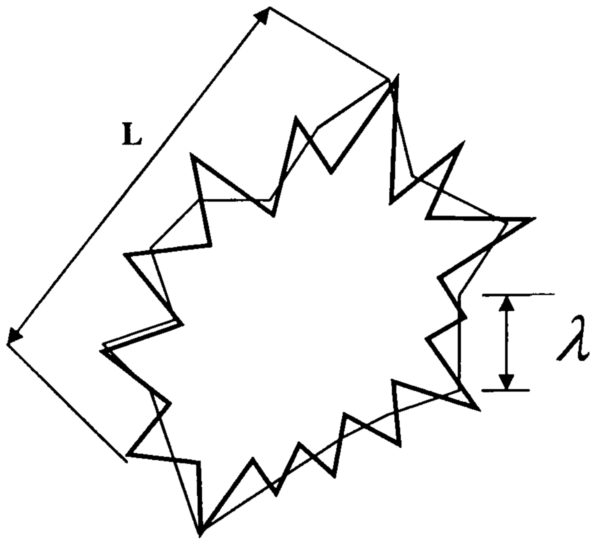
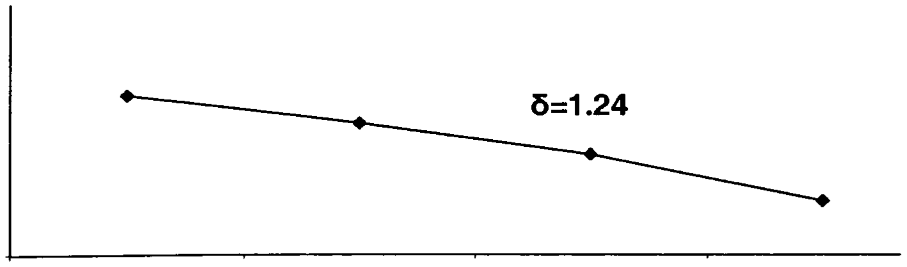


fig 6

λ is varied a plot of P against λ yields data with a slope *m* where

$$\delta = 1 + |m|$$

δ is the fractal dimension.



Fractal dimension does not tell us anything about the overall gross shape of the profile. An example of the application of fractal dimension is a robot programmed to use the structured walk exploration technique, can only "see" what it "feels" at any one time with its "fingers" set at λ.

Koch Island	1.1291
Quadratic Koch island	1.5
Koch island and archipelago	1.6131
Peano curve	2

Fractal dimensions [2]

2.2 ITERATED FUNCTION SYSTEMS (IFS)

Iterated functions scheme is a new type of dynamical system that employs a collection of contracting maps to create chaotic attractors that have a fractal structure. [20].

IFS can be thought of as a collection of functions whose domain and range are in the same space. It can be considered as a set of rotations, translations, scalings and reflections. Iteration will mostly result in a fractal attractor that is the union of all limit points of the composition of the functions.

There are generally two ways of creating IFS:

- Deterministic- where all the transformations are applied simultaneously.
- Random – where the transformations are applied in a random order.

IFS has been useful to help produce images of clouds, smoke, seascapes, flames, horizons and plant branching to name a few.

The simplest IFS can be described as a set of two dimensional affine maps:

$$x_{n+1} = a_1 x_n + a_2 y_n + a_5$$
$$y_{n+1} = a_3 x_n + a_4 y_n + a_6$$

The Sierpinski triangle uses the same iteration principle but the results lie in the real plane, using only real numbers. An example of this can be generated using the following affine transformations:

$$w_1(x,y) = (0.5x, 0.5y)$$
$$w_2(x,y) = (0.5x + 0.5, 0.5y)$$
$$w_3(x,y) = (0.5x, 0.5y + 0.5)$$

Where w represents the transformation maps.

The Henon model is given by:

$$X_{n+1} = a * x_n - b(y_n - x_n{}^2)$$
$$Y_{n+1} = b * x_n + a(y_n - x_n{}^2)$$

Here $a = \cos\alpha$ and $b = \sin\alpha$.

The linear map used later in the Matlab codes as (Test2.m and Test3.m) use the basic function

$$x_{n+1} = a_1 x_n + b$$

The code Test2.m uses 2 functions:

$$f_1(x) = a_1 x + b_1$$
$$f_2(x) = a_2 x + b_2$$

And the range of a and b is in $0 < a_i + b_i < 1$.

The code Test3.m uses 3 functions.

2.3 COMPLEX FRACTALS

Gaston Julia (1893-1978) in 1919 founded the mathematical theory of a type of fractal generated by a so called iterative conformal transformation .A conformal transformation is one that leaves the angle unchanged.

$$X' = x^2 - y^2 + a$$
$$Y' = 2xy + b$$

where b and a are arbitrary numbers. For every value of b and a, we obtain a fractal (Julia fractals).

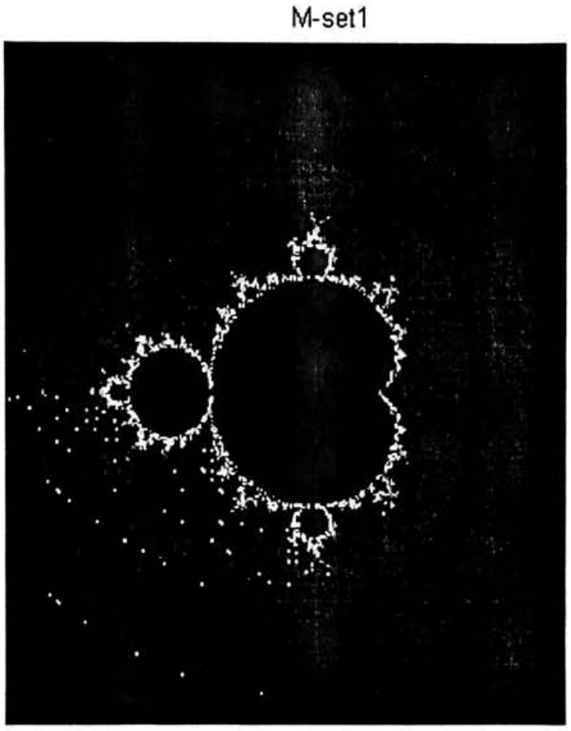
In the complex notation we have:-

$$Z' = Z^2 + c$$

Where $Z = X + i * Y$ and $c = a + i * b$.

The set of points, whose orbits are bounded under the iteration of, $q(Z) = Z^2 + c$ is called the filled Julia set of q. A Julia set is the boundary of a filled Julia set [24].To create a Julia set, the value of c is fixed and the values of Z is varied .Then iterations are performed.

To create a Mandelbrot set the value of Z is fixed initially at zero and the value of c is varied.



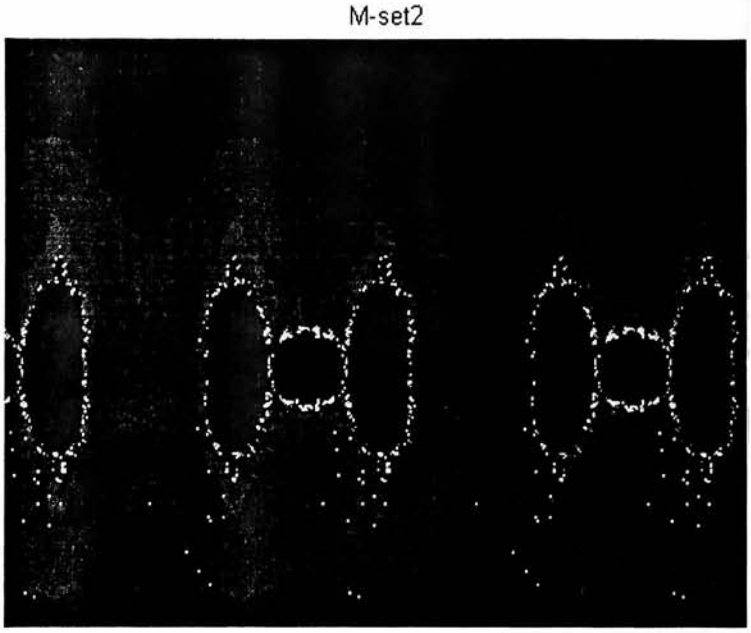
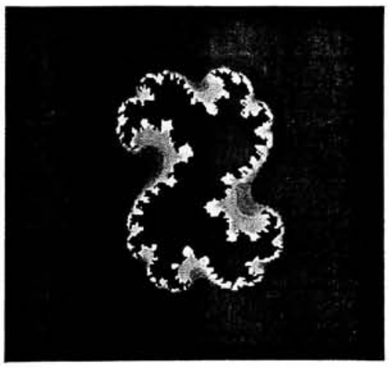


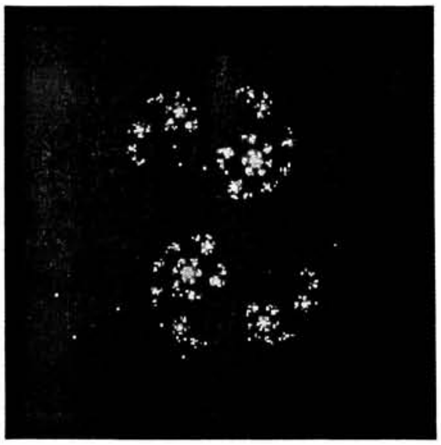
fig 7

Julia set



(Code: Julia_5)
with c = 0.36 + 0.1i

Julia set



(Code: Julia_6)
with c = 0.46 + 0.2i

fig 8

3.0 SYNTHESIS OF SOUND

The basic process involved in this chapter is mapping mathematical algorithms into sound. The first method discussed is mathematical synthesis which involves mapping sine waves to sound and obtaining a pleasable sound. The next method, MIDI, involves mapping instructions into sound. In this thesis the instructions are created from the fractals.

3.1 MATHEMATICAL SYSNTHESIS

One of the well known links between mathematics and music is the theory of Fourier series. This theory states that every periodic function can be represented by an infinite series containing only sine and cosine terms where the frequencies are integer multiples of one fundamental frequency, the frequency of the tone. We are able to distinguish a pleasant sound by the way the tones are arranged. Therefore if we can come up with a formula recipe that arranges the tones mathematically then we can make a pleasant sound mathematically.

The series is given by:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{2\pi nx}{c}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2\pi nx}{c}\right)$$

Considering a special case with the wavelength, c=1. When all a_n 's are zero the resulting function is symmetric around the origin f (-x) = f (x) and this is equivalent to the condition that f(0) = f (1) , which in physical terms , corresponds to the waveform representing a vibration string; secured at the endpoints.
Hence the resulting function used here to get a waveform is

$$f(x) = \sum_{n=1}^{\infty} b_n \sin(2\pi nx)$$

According to some research done Erich Neuwirth [9], they found out that to get a pleasing sound from a sine wave, the value of the coefficients in the series above has to be $b_{n+1} = \frac{b_n}{2}$ or $b_n = \left(\frac{1}{2}\right)^{n-1}$ which means that the overtone amplitude used is half the previous one.

For generalization purposes suppose the value $\left(\frac{1}{2}\right)$ is considered a factor q in the range -1<q<1. Then the term becomes

$$f(x) = \sum_{n=1}^{\infty} q^{n-1} \sin(2\pi nx)$$

It would be helpful to consider this as part of a complex function

$$g_q(x) = \sum_{n=1}^{\infty} q^{n-1} \cos(2\pi nx) + i\sum_{n=1}^{\infty} q^{n-1} \sin(2\pi nx)$$

$$= \sum_{n=1}^{\infty} q^{n-1} e^{inx}$$

$$= e^{ix} \sum_{n=1}^{\infty} (qe^{ix})^n$$

From the convergence test of a geometric sum [44] we get

$$g_q(x) = \frac{e^{ix}}{1 - qe^{ix}}$$

This leads to

$$= \frac{e^{ix}(1 - qe^{-ix})}{(1 - qe^{ix})(1 - qe^{-ix})} = \frac{e^{ix} - q}{1 + q^2 - 2q\cos(2\pi x)}$$

Expanding e^{ix} leads to

$$g_q(x) = \frac{(\cos(2\pi x) - q) + i\sin(2\pi x)}{1 + q^2 - 2q\cos(2\pi x)}$$

The imaginary part then is:

$$f_q(x) = \frac{\sin(2\pi x)}{1 + q^2 - 2q\cos(2\pi x)}$$

From further calculations [36] and considering the local extrema the value of x is found to be

$$x = 1 - \frac{1}{2\pi}\arccos\left(\frac{2q}{1+q^2}\right)$$

A matlab code for the synthesis of sound can be created

Matsynthesis.m

```
% Mathematical synthesis of sound
clear
q = -1:0.01:1;            % when using a fractal the value of q will come from the
                          % fractal image
x = (1-(1/2*pi).* acos(2.*q./(1+ q.^2)));
for n=1:201
   f(n)= sum(q.^(n-1).*sin(2*pi.*n.*x));
end
wavwrite (f, 'synsound')  % this creates a wav file in the directory called
                                %'synsound.wav'.
%the method above sets the sampling rate at the default value in Matlab. The
value is 8192 Hz. It is important to have in mind the sampling frequency because
this helps put a check on aliasing. Hence to create a sinusoid to make sound we
can specify the sampling rate. It has to be more than twice the highest frequency
in the signal. Suppose we want to play for 1 second and we have 201 points then
it will be comfortable to use the sampling frequency as 512.
Fs=512;
N=201;
t=( [1:N]-1)/Fs;
y1=sin(2*pi.*x.*t);
plot(t,y1);
sound(y1,Fs)  % this creates a sound vector and plays at the same time
wavwrite (f,512, 'synsound')  % this creates a wav file
```

3.2 MIDI BASICS

Definition: MIDI stands for Musical Instrument Digital Interface.

MIDI FILE

A MIDI file is a data file. It stores information, just like a text (i.e., ASCII) file may store the text of a newspaper article, but a MIDI file contains musical information.
Specifically, a MIDI file stores MIDI data -- the data (i.e., commands) that musical instruments transmit between each other to control such things as playing notes and adjusting an instrument's sound in various ways.
You can't load a MIDI file into a text editor such as Notepad and view it. That is why a Midi Disassembler is needed to view the text file.

Sequencers are software used to play back a MIDI file. Windows has a MIDI sequencer built right in. It allows MIDI data sequences to be captured, stored, edited, combined, and replayed. The MIDI data output from a MIDI sequencer is transmitted via the devices 'MIDI OUT' connector.
A MIDI message is made up of an eight-bit status byte, which is generally followed by one or two data bytes. At the highest level, MIDI messages are classified as being either Channel Messages or System Messages.[11] .(More details on MIDI is in Appendix 3.0)

HOW IT WORKS

MIDI manages to remain compact by capturing the essential characteristics of a piece of music in a shorthand format. During the play back the MIDI information is decoded in real time, including details such as what instrument voices are playing, what notes and for what duration. The frequency of a sound wave determines the pitch of a sound -what we perceive as high or low an audible wave appears.

Advantages of using MIDI composers [3]

One is able to:

- Write down notes

- Transpose the music
- Change clefs
- Cut and paste notes
- Change tempo/keys.

Advantages of using MIDI files

- One can easily edit and manipulate data.
- They occupy a small storage space.

A good reason for using MIDI: Because of this relation between instruments, which we humans hear in music, we can distinguish the separate instruments (or instrument groups). Therefore in some way we are able to 'translate' a piece of music into a MIDI file by listening to it. A computer (program) does not have that ability, that sense. It cannot distinguish music from noise.

Difference between MIDI and other audio files:

MIDI files store MIDI messages, which are commands that tell a musical device what to do in order to make music. For example, messages to play a particular note or to change to another instrument. These files do not contain actual sampled sound files like WAVE files.

WAVE files store digital audio waveforms. This data must be played back through a device with a Digital to Analog Converter (i.e., DAC) such as a MIDI sampler or a computer sound card's DAC. There is no information concerning musical rhythms or tempo stored in a WAVE file.

The primary difference between a WAVE and MP3 file is that the latter uses compression to squeeze the data down in size, resulting in a typically much smaller file size.

An example of musical attractors

Keys, scales and chords contain tendencies that are established by the internal organization of the key. In C major, C is the tone toward which all motion tends. Other supporting tones such as E and G help stabilize C and establish its definition. B is known as a ‘leading tone’ tending upward to C; while F tends downward toward E if no other motion requires it to rise. [11]

Here then C is a prominent attractor with support from E and G as less prominent but useful attractors.
Other musical attractors might be the meter and the barline.

A small copy of a Midi text file:

```
MFile 0 1 24
MTrk
0 TimeSig 4/4 24 8
0 Tempo 500000
 60 On ch=10 n=36 v= 77
 62 On ch=10 n=36 v= 0
TrkEnd
```

Where:

‘MFile 0 1 24’	–Describes the header of the file. The file name, file format method, number of tracks and the number of clock ticks.
‘MTrk’	– Shows the beginning of the track.
‘0 TimeSig 4/4 24 8’	–This specifies the way the music is going to be played.
‘0 Tempo 500000’	– This specifies the speed of the music piece. The higher the number the slower the speed.
‘60 On ch=10 n=36 v= 77’	- Describes the number of the note, that it is to be ‘ON’, its to be played on Channel 10, the key that is to be played (in midi numbers), the volume at which it will be sounded. (Maximum volume is 127).
‘62 On ch=10 n=36 v= 0’	- Describes the number of the note. This is the same key

note that was played before but this is 'OFF' because the volume 'v' is 0.

'TrkEnd' - Denotes the end of the track.

NB: When making a midi text there has to be a new line after 'TrkEnd'. I guess this is because of how the Text to midi conversion application file was made.

4.0 ACOUSTIC SIGNATURE

4.1 BASICS

Definitions of fractal music

Hugh McDowell "All music may be thought of as consisting of patterns. In nature, music and fractals there is a happy blend of form and irregularity, structure and surprise, or, if you prefer, theme and variation".

Lawrence Ball

"Music is fractal in many senses, one of which is that there are patterns occurring at different time scales. Music which has these same patterns is called self-similar since levels of basic skeletal structures within it are similar to its surface patterns."

Fractal music

The term 'Fractal Music' typically refers to music composed wholly or in part using the same types of iterative or recursive processes used to create fractal images. This is different from the term 'acoustic signature' where music is made directly from the fractal image as a way of identifying the fractal.

Algorithmic Composition

Creating music from fractals is an example of 'algorithmic' music composition. Algorithmic composition refers to the process of using a formula or recipe (an algorithm) for creating music. Musical 'recipes' can be formal or informal depending upon how precisely the compositional steps are specified.

Though fractal music can be composed without a computer, currently most is composed with the aid of a computer. Computer composed fractal music always uses a formal 'recipe' - the software program - to produce music.

During computer-based fractal music composition inputs from the fractal image or fractal generating process are converted to musical parameters to create melodies, harmonies, rhythms, textures, etc.

a) Voss (1/f) Music In 1975, Voss and Clark discovered music often follows a '1/f (frequency)' rule like many other natural phenomena. 1/f noise is an example of 'scaling noise' and therefore has certain fractal characteristics.

A '1/f' distribution is used to generate note pitches and volumes. Note that with this distribution small pitch changes are more probable than large changes. With the random algorithms all pitch interval changes are equally possible.

b) Chaotic Attractor Music

Many attractor-based composition algorithms work by assigning a starting pitch to particular x, y coordinates on the attractor. They then follow the orbit of successive **iterations** along the attractor and map either the absolute values of the coordinates or the change in coordinate to a given pitch range.

c) IFS Attractor Music

Like the Chaotic Attractor example this algorithm works by following the orbit of the attractor for **successive iterations** and mapping the normalized values of the coordinates along that path to an index into the pitch array.

d) Mandelbrot Music

This method generates melodies by following the orbit of an attractor and mapping the coordinate values to musical parameter values. An example is by using the RGB color values from a fractal image to generate fractal music.

4.2 DSP METHODS

There are two methods particularly specified in this thesis on how to get an acoustic signature of fractals. Concentrating on the IFS fractals, the methods used are:

- Making a wave from x and y coordinates. These values obtained from the coordinates are 'modified' using signal processing tools (DSP methods) to create a signal. The signal is converted to a WAV file and then to a MIDI file.
- Mapping the 'modified' x, y coordinates to the music key notation .From this point it is converted to a midi text file or one can manually enter them into the bar chords in a music software and the result can be heard through a MIDI instrument like a keyboard.

This thesis is not dealing with manually drawing notes into the music bar chords.

DSP literally Digital Signal Processing is a way of processing signals. A physical signal such as sound pressure or voltage from an amplifier can be represented as a continuous function of time.

To get an acoustic signature for fractals the data obtained from the fractals like the x, y coordinates have to be analyzed in the form of a continuous signal. One of the ways of obtaining a waveform of a map with respect to time is by getting the time series .By doing a Fourier transform on the continuous signal we are able to get the frequency domain representation. This is convenient for filter design and analysis among other applications.

Some important definitions associated with DSP are:

- Nyquist rate –to obey the Nýquist sampling theorem, a band limited waveform has to be sampled at a rate which is at least twice the Nyquist frequency. This minimum sampling rate is known as the Nyquist rate. Failure to follow this restriction results in aliasing.
- Aliasing -refers to an often detrimental phenomenon associated with the sampling a continuous time waveform at a rate below the Nyquist rate. Hence the frequencies greater than one-half the sampling rate becomes indistinguishable from frequencies between DC and one-half the sampling rate (the fundamental band width).

  To avoid aliasing, the sampling frequency has to be at least twice the highest frequency occurring in the signal.
- Nyquist frequency – For a band limited waveform, the width of the band of frequencies contained within the waveform is described by the upper limit known as the Nyquist frequency.
- Sampling rate -refers to the frequency at which a continuous time waveform is sampled to obtain a corresponding discrete – time waveform. Values are given in Hz. [43].
- Zero-Padding- This is the method of extending a signal with zeros to extend its time (or frequency band) limits. This enables the creation of a wave file with a reasonable amount of sampling rate.

- Discrete Fourier Transform (DFT) - This is Fourier transform in a discrete form. The infinite sum in the Fourier transform is replaced with a finite sum. The result when applied to a discrete signal is a set of sine and cosine coefficients which when multiplied to sine and cosine waves of appropriate frequencies the original waveform is reconstructed.
- Fast Fourier Transform (FFT) – This allows the DFT to be obtained rapidly and efficiently. It reduces the number of computations needed for N points from $2N^2$ to $2N \log_2 N$, where $\log_2$ is the base -2 logarithm.

In Matlab,

```
F1=abs(fft(y1)); % the fft of a signal y1
```

The formula for DFT is

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\left(\frac{2\pi}{N}kn\right)}$$

$$k = 0,1,\ldots\ldots N-1$$

The transform in the frequency domain is:

$$X(jw) = \sum_{n=0}^{N-1} x(nT) e^{-jwnT}$$

5.0 INSTRUMENTATION AND TESTING

Equipment used for this thesis: -

- Computer
- Music Keyboard
- Midi connectors

Software employed:

- Matlab / C++
- Music Time deluxe

Advantages of using Matlab:

- Storing capabilities for visualization, analysis and algorithm development.
- Compatibility across multiple hardware
- Open and extensible software architecture.

To create an acoustic signature the main programs are run in Matlab .The application files, - t2mf.exe and mf2t.exe, that are used to convert a text file to a midi file and vice versa respectively, have to be loaded into the folder with the Matlab algorithms.
To convert for example the text file to midi file, in Matlab we type:

```
! t2mf    'textfile'.txt   'midifile'.mid
```

Where 'textfile' is the name of the text file that had been created from the midi text-creation programs (Midi programs) and 'midifile' is the name of the midi sound file to be created.
It is possible to open the window media player from Matlab in order to play a midi file. The file 'mplayer2.exe' located in the Windows media folder has to be copied into the folder containing the Matlab and midi files. Then to execute this from Matlab we type:

```
! mplayer2 'midifile'.mid
```

If we are using the other version of the windows media, we will do the same with 'wmplayer.exe' and type:

```
! wmplayer 'midifile'.mid
```

1. MAIN PROGRAMS

Caller programs

These programs bring together different programs hence the linking one program to another. They link up the program making the fractal, the program processing the fractal, the program making the midi file and they initialize the opening of the midi file. Some of the fractal programs are designed to create video files (*.avi). The avifile can be played in the media windows when executed from Matlab. The two options from Matlab platform are:

```
! mplayer2 'avifile'.avi
! wmplayer 'avifile'.avi
```

Hence the stages from a fractal creation to a midi file in text format can be achieved from running one file.

- Caller 1 is the logical map that requires an input of either Test2 or Test3. These are logical maps involving a 2x2 and a 3x2 matrix respectively. Other programs linked in this one are:
  - Signal_IFS
  - Shownotes
  - Midi programs- Notelabels, Pianolabels, and Drumlabels.

- Caller 2- Uses the same format as in Caller 1 but it first creates a wav file from the map and then uses a C++ program to create a midi text file. To create a wav file, DSP tools are utilized. These are zero-padding, fft and magnitude sampling. Other programs linked are:
  - Signal_IFS
  - wavconverter.m

        - mFormat.c

- Caller3- Creates an acoustic signature for random-point fractal system. It requires the specific file as input together with the number of points desired. Other programs linked are: -
    - Random_IFS
    - Midi programs

- Caller4- Uses the same files in Caller3 but it creates a wav file first and transforms this to get the midi text file. Other programs linked are:
    - Random_IFS
    - wavconverter.m
    - mFormat.c

- Caller5 – Gives the option to select a Julia set.
- Caller6 - Starts the process to get a Julia set by giving an option to choose the value of c (a constant) and links up to Caller5.
- Caller7 – Just like Caller 6 but gives other values of c to choose from.
- Caller8 – This program combines the option to choose a value of c and also to choose a Julia set.
- Caller9- Creates the Mandelbrot set
- Caller10- creates fractals from iterated function systems.

2. DSP PROGRAMS

These are called DSP programs because they use DSP tools such as

- Fast Fourier Transform
- Sampling
- Normalization
- Zero-padding

to get the desired output. Not all fractals can be treated in the same way so by using the appropriate DSP tools we are able to get an acoustic signature in consistency with the uniqueness of the fractal. The programs in this category are:

- Signal_IFS- the main focus in this is to use the X coordinate points from the fractal. These are
- Random_IFS -this program makes use of both the X and Y coordinate points. Both are mapped to the parts of a complex number where the X represents the real part and the Y the imaginary part. Then the absolute is obtained and after normalizing the points are mapped into midi by either creating a wav file or mapping the numbers into 'midi' numbers.
- Frac_IFS –This program deals with the attractor-like fractals.
- ComplexIFS- this program deals with the fractals that are made from the complex fractals. These involve the Julia sets and Mandelbrot fractals.

3. MIDI PROGRAMS

These are the programs that induce the mapping into the acoustic domain. The main focus is to get a sound file and particularly MIDI because of the benefits of MIDI as discussed before. The programs involved are:

- Shownotes -This operates with the logical map and maps the X coordinate points onto the musical scale ranging from 0-12. There are 12 keys on the musical scale namely A, A+, B, C, C+, D, D+, E, F, F+, G, G+. So every number obtained is changed into these keys and is further mapped into midi numbers (recognized by the text-to-midi converter).
- Notelabels- this program embarks on changing the numbers obtained from the DSP programs into the musical scale keys. The first thing is to change the numbers into a string. The main function involved is 'strrep' which replaces the numbers with the keys as specified.

- Drumlabels- this program further maps the keys into midi numbers. The numbers for drums in midi are specifically from 34 to 58. So the keys are replaced with these numbers. A useful tool has been the Chart 2 in appendix.
- Pianolabels- this program like Drumlabels maps the keys from 'Notelabels' to piano numbers. These range from 0-127. Middle C is 60. Since middle C is comfortable to listen to, not high or low, this program makes use of 12 numbers from 60.
- Drumtext.m and Piantext.m – these both operate with the numbers obtained to make a midi text file. This file is later changed into a midi file one can listen to. Piantext is modified in a way that one can listen to a specific instrument by changing the value of "p". The instrument numbers are given in the appendix.

RECONSTRUCTION

This is an attempt to reconstruct a fractal from a midi file. The steps are:-

- Record the piece of music. This is done with the help of music keyboard and music deluxe midi software.
- Save as a midi file.
- Move the midi file to the folder containing the midi-to-text converter application file (mf2t.exe) i.e. to the current Matlab directory.

```
! mf2t   'midifile'.mid   'textfile'.txt
```

Where 'midifile' is the name of the midi file recorded and 'textfile' is the name of the text file being created.

- In Matlab we load the text file and run 'midiback.m'. This program picks out the keys from the text file and these can be used to create a fractal. Hence a color map can be created.

6.0 Matlab Programs

The Matlab programs are divided into different sections:

1. Main programs
2. Chaos and fractal programs
3. Signal processing programs
4. Midi programs

1. Main Programs

The main programs are programs to demonstrate how to make an acoustic signature of fractals. The basic programs used here are the versions of 'Test.m' and 'fern.m'.

The programs are called 'Callers' because they call different files together to get a result. The final output is a midi text file that has to be changed to a midi file using an executable file outside of matlab.

==

Caller1.m

```
% This program involves the logistic/linear map
% The notes obtained from shownotes file are further mapped
% into midi drum /piano numbers in a midi text file .
%The file is later turned to a midifile using an application.
% When running again and changing the logical map , all other files will be
% changed accordingly.


clf
% select logical map
 while 1    clc;
```

```
    v = input('Test2(1), Test3(2) ? ');
    if v >= 1 & v <= 2;       break
    end
 end

switch v
   case 1
      disp('Test2- every file made will be for "Test2" ');
      Test2;
   case 2
      disp('Test3- every file made will be for "Test3"');
      Test3;

end

%
shownotes;
%
notelabels;
%
drumlabels;
%
pianolabels;
%
% Run the stand alone application to change the midi text to a midi file
%within Matlab.

! t2mf  drumtext.txt  caller1a.mid
%
disp( 'a midi file called "caller1a.mid" has been created ')
% to create an additional wav file
[filename,path]=uigetfile('*.mid','Choose CALLER1a.MID Input Sound File');
habari = [path,filename];

Midi2wav;   %Creates an additional wav file.
%
movefile('mzuri.wav','caller1a.wav')% to change the name back
%
! t2mf  piantext.txt  caller1b.mid
%
disp( 'a midi file called "caller1b.mid" has been created')
% to create an additional wav file
[filename,path]=uigetfile('*.mid','Choose CALLER1b.MID Input Sound File');
habari = [path,filename];

Midi2wav;   %Creates an additional wav file.
```

```
%
movefile('mzuri.wav','caller1b.wav')% to change the name back

%
%To listen to the midi sounds
pause
%
! mplayer2  caller1a.mid
%
pause
disp('may have to manually close the mplayer2 window to play the next *.mid')
%
 ! mplayer2  caller1b.mid
```

===

Caller2.m

```
% This program involves IFS fractals
% and makes a wav file from the data obtained from the fractal
% The data from the wav file-(obtained using wavread) is compiled using a MEX-
% file in C++ program.
% The end result is a midi text file that is later converted to a midi file.

clf

%
n = input('Enter the number of points for the fractal:(at least 500):\n\n n = ')
disp(' ');
% select fractal
 while 1    clc;
      v = input(' Choose the kind of fractal \n Tree(1), Sierp(2) ? ');
    if v >= 1 & v <= 2;       break
    end
 end

switch v
   case 1
      disp('Tree on figure');
      [xpts,s]=Tree(n);
   case 2
      disp('Sierp on figure');
      [xpts,s]=Sierp(n);

end
%
Allocation;  % this processes points and creates a wave file.
```

```
%
disp(‘ a wave file called “random.wav” has been created for the next step’);
%
pause
%
disp(' Upon request to choose a compiler choose "LCC" ')
%
wavconverter;   % makes a matlab matrix from the wav file
%
mex mFormat.c    % from here we are building a text file
%
mFormat(ans,'tree.txt')   % this text is used to get a midi file.
%
disp(‘ a text called “tree.wav” has been created’)
%
% Run the stand alone application to change the midi text to a midi file
%within Matlab.
if v==1;
! t2mf  tree.txt  Tree.mid
%
disp( 'a midi file called "Tree.mid" has been created ')
[filename,path]=uigetfile('*.mid','Choose Tree.MID Input Sound File');
habari = [path,filename];
Midi2wav;   %Creates an additional wav file.
%
movefile('mzuri.wav','Tree.wav')% to change the name back
%
else
! t2mf  tree.txt  Sierp.mid
%
disp( ‘a midi file called "Sierp.mid" has been created ')
%To create an additional wav file
[filename,path]=uigetfile('*.mid','Choose Sierp.MID Input Sound File');
habari = [path,filename];
Midi2wav;   %Creates an additional wav file.
%
movefile('mzuri.wav','Sierp.wav')% to change the name back
end
pause

% to play the midi sounds.
if v==1;
 !  mplayer2  Tree.mid
else
 !  mplayer2  Sierp.mid
end
```

==

Caller3.m

```
 % This program creates the fern fractal and
 % processes the data to obtain an acoustic signature in the form of
 % midi drums thru 'drumlabels.m' or piano sounds thru' pianolabels.m'
% the signature can be heard thru midi file obtained using an application.
clear

%
n = input('Enter the number of points for the fern:\n\n n = ')
disp(' ');
if n<=300
   error(['use a minimum of 500']);
end
% select fern
 while 1    clc;
      v = input(' fern(1), fern1(2),? ');
    if v >= 1 & v <= 2;       break
    end
 end

switch v
   case 1
      disp('fern');
      [xpts,s]=fern(n);
   case 2
      disp('fern1');
      [xpts,s]=fern1(n);

end

%
Allocation;
%
drumlabels
%
disp(' a text called "drumtext.txt" has been created')
%
pianolabels
%
disp(' a text called "piantext.txt" has been created')
% Run the stand alone application to change the midi text to a midi file
%within Matlab.
```

```
! t2mf  drumtext.txt  fernd.mid
%
disp( 'a midi file called "fernd.mid" has been created ')
%To create an additional wav file
[filename,path]=uigetfile('*.mid','Choose fernd.MID Input Sound File');
habari = [path,filename];
Midi2wav;   %Creates an additional wav file.
%
movefile('mzuri.wav','fernd.wav')% to change the name back
%
! t2mf  piantext.txt  fernp.mid
%
disp( 'a midi file called "fernp.mid" has been created')
%To create an additional wav file
[filename,path]=uigetfile('*.mid','Choose fernp.MID Input Sound File');
habari = [path,filename];

Midi2wav;   %Creates an additional wav file.
%
movefile('mzuri.wav','fernp.wav')% to change the name back
% To listen to the midi sounds
pause
%
! mplayer2 fernd.mid
%
pause
disp(' manually close the mplayer2 window to play the next "*.mid" ')
%
 ! mplayer2 fernp.mid
```

==

Caller4.m

```
 % this program creates a fern fractal and using the location of the data
 % points it creates an acoustic signature. the first step involves
 % creating a wav file and then converting it to a midi file. The wav file
 % file is changed into a matlab matrix and this is then compiled thru a C++
 % program to get a midi text file. This is then later converted to a midi file.
%Changing the type of fern fractal will change the type of miditext
 %displayed and the midi file too.

%
clear
%
```

```
clf

n = input('Enter the number of points for the fern:\n\n n = ')
disp(' ');
% select fern
 while 1    clc;
      v = input(' fern(1), fern1(2) ? ');
    if v >= 1 & v <= 2;        break
    end
 end

switch v
   case 1
      disp('fern');
      [xpts,s]=fern(n);
   case 2
      disp('fern1');
      [xpts,s]=fern1(n);

end

%
Allocation % processes the data points and creates a wav file.
%
disp(‘ a wave file called “random.wav” has been created-for the next step’);
%
pause
%
disp(' Upon request to choose a compiler choose "LCC" ')

wavconverter; % Changes the wav file to a matlab matrix.
%
mex mFormat.c   % from here we are building up the text file.
%
mFormat(ans,'fern.txt')
%
disp(' a text called "fern.txt" has been created')
%
 ! t2mf  fern.txt  caller4.mid
%
disp( 'a midi file called "caller4.mid" has been created ')
%To create an additional wav file
[filename,path]=uigetfile('*.mid','Choose Caller4.MID Input Sound File');
habari = [path,filename];
Midi2wav;   %Creates an additional wav file.
%
```

```
movefile('mzuri.wav','caller4.wav')% to change the name back
%
disp('manually close the mplayer2 window after playing midi ')
pause
% to play the midi sounds.

! mplayer2  caller4.mid
```

==

Caller5.m

```
function  [W] = Caller5(varargin);

global c

% This file helps to choose the type of julia recipe to plot.
% you have to begin from caller6 or Caller7

N = input('Enter the number of iterations:\n\n N = ')
disp(' ');
if N > 50
    error(['Sorry!Takes too long to iterate-use a ' ...
       'maximum of 50']);
end

 while 1    clc;
       v = input('select a julia set\n\n J1(1), J2(2),J3(3),J4(4),J5(5),J6(6),J7(7) ? ');
    if v >= 1 & v <= 7;    break
    end
 end
       switch v
   case 1
      disp('J1');
    [W]=Julia_13(c,N);
   case 2
      disp('J2');
    [W]=Julia_2(c,N);
   case 3
      disp('J3');
   [W]=Julia_3(c,N);
   case 4
      disp('J4');
   [W]=Julia_4(c,N);
   case 5
```

```
      disp('J5');
  [W]= Julia_5(c,N);
   case 6
      disp('J6');
  [W]=Julia_6(c,N);
   case 7
      disp('J7');
   [W]=Julia_7(c,N);
end

complexnotes
```

==

Caller6.m

```
%This program helps to start of making Julia sets in conjunction with
%Caller 5. It gives you the option of choosing a complex number to use.

global c

%select c
 while 1    clc;
         v = input('select a value of c\n\n c1(1), c2(2),c3(3),c4(4),c5(5),c6(6) ? ');
    if v >= 1 & v <= 6;        break
    end
 end

switch v
   case 1
      disp('c1');
     c = 0.5 + i*0.5;
   case 2
      disp('c2');
     c = 0.46 + i*0.2;
   case 3
      disp('c3');
    c = 0.36 + i*0.1;
   case 4
      disp('c4');
     c = 0.29 + 0.54*i;
   case 5
```

```
      disp('c5');
     c=-0.1+0.8*i;
   case 6
      disp('c6');
     c=0.4+0.07*i;
end

Caller5  % link to Caller5.m
```

==

Caller7.m

```
%This program helps to start of making Julia sets in conjunction with
%Caller 5. It gives you the option of choosing a complex number to use.
% Other numbers are available in Caller6.

 %select c
 while 1    clc;
         v = input(' c1(1), c2(2),c3(3),c4(4),c5(5),c6(6),c7(7),c8(8) ? ');
    if v >= 1 & v <= 8;        break
    end
 end

switch v
   case 1
      disp('c1');
      c = -0.194 + i*0.6557;
   case 2
      disp('c2');
      c = 0.27334 + i*0.00742;
   case 3
      disp('c3');
     c = -1.553 + i*0.003;
   case 4
      disp('c4');
      c = -0.765 + i*0.003;
   case 5
      disp('c5');
      c = -0.765 + i*0.11;
   case 6
      disp('c6');
     c = -0.687 + i*0.312;
   case 7
      disp('c7');
      c = 0.402 + i*.1951;
   case 8
      disp('c8');
      c = 0.399 + i*.0999;
end

Caller5  % link to Caller5.m
```

==

Caller8.m

```
% This program involves some Julia sets. the variables are all in one file
% unlike the connection of Caller6 &7 with Caller5

 while 1    clc;
        v = input('select a value of c\n\n  c1(1), c2(2),c3(3),c4(4),c5(5) ? ');
    if v >= 1 & v <= 3;        break
    end
 end

switch v
   case 1
      disp('c1');
      c= -.745429+.11308*i;
   case 2
      disp('c2');
      c = 0.27334 + i*0.00742;
   case 3
      disp('c3');
      c = 0.399 + i*.0999;
   case 4
      disp('c4');
      c = -0.01;
   case 5
      disp('c5');
      c = 0.36 + i*0.1;
end
%

N = input('Enter the number of iterations:\n\n N = ')
disp(' ');
if N > 50
    error(['Takes too long to iterate-use a ' ...
        'maximum of 50']);
end
%
 while 1    clc;
        v = input('select a julia set\n\n J8(1), J9(2), J10(3),J11(4),J12(5) ? ');
    if v >= 1 & v <= 5;        break
    end
 end

switch v
   case 1
       disp('J8');
 [W]= Julia_8(c,N);
   case 2
       disp('J9');
[W]=Julia_9(c,N);
   case 3
       disp('J10');
[W]=Julia_10(c,N);
  case 4
       disp('J11');
[W]=Julia_11(c,N);
  case 5
       disp('J12');
[W]=Julia_12(c,N);
end

complexnotes

%
disp(' a text called "drumtext.txt" has been created')
%
% Run the stand alone application to change the midi text to a midi file
%within Matlab.

! t2mf  drumtext.txt caller8.mid
%
disp( 'a midi file called "caller8.mid" has been created - percussion sounds')
%To create an additional wav file
[filename,path]=uigetfile('*.mid','Choose caller8.MID Input Sound File');
habari = [path,filename];
Midi2wav;   %Creates an additional wav file.
%
movefile('mzuri.wav','caller8.wav')% to change the name back
% To listen to the midi sounds
%
```

```
% To listen to the midi sounds
pause
%
! mplayer2 caller8.mid
```

===

Caller9.m

```
% This program involves some Mandelbrot sets.

N = input('Enter the number of iterations:\n\n N = ')
disp(' ');
if N > 30
    error(['Takes too long to iterate-use a ' ...
       'maximum of 30']);
end
%
 while 1    clc;
v = input('select a mandelbrot set\n\n M1(1),M2(2), M3(3),M4(4),M5(5),M6(6) ? ');
    if v >= 1 & v <= 6;       break
    end
 end

switch v
   case 1
      disp('M1');
 [W]= Mandel_1(N);
   case 2
      disp('M2');
[W]= Mandel_2(N);
   case 3
      disp('M3');
[W]= Mandel_3(N);
  case 4
      disp('M4');
[W]= Mandel_4(N);
  case 5
      disp('M5');
[W]=Mandel_5(N);
case 6
      disp('M6');
[W]=Mandel_6(N);

end
```

```
complexnotes

%
disp(' a text called "drumtext.txt" has been created')
%
% Run the stand alone application to change the midi text to a midi file
%within Matlab.

! t2mf  drumtext.txt caller9.mid
%
disp( 'a midi file called "caller9.mid" has been created - percussion sounds')
[filename,path]=uigetfile('*.mid','Choose Caller9.MID Input Sound File');
habari = [path,filename];
Midi2wav;   %Creates an additional wav file.
%
movefile('mzuri.wav','caller9.wav')% to change the name back
%
% To listen to the midi sounds
pause
%
! mplayer2 caller9.mid
```

==

Caller10.m

```
% This program involves different fractals.

while 1    clc;
 v = input('select a fractal\n\n M1(1), M2(2), M3(3) ? ');
   if v >= 1 & v <= 6;       break
   end
 end

switch v
   case 1
      disp('Mosaic');
     [W] = mosaic;
      complexnotes
  case 2
      disp('gingerman');
       [X1,Y1] =gingerman(N);
   case 3
      disp('dream');
      [X1,Y1] =dream(N);
    end
```

```
%
disp(' a text called "drumtext.txt" has been created')
%
% Run the stand alone application to change the midi text to a midi file
%within Matlab.

! t2mf drumtext.txt caller10.mid
%
disp( 'a midi file called "caller10.mid" has been created - percussion sounds')

[filename,path]=uigetfile('*.mid','Choose CALLER10.MID Input Sound File');
habari = [path,filename];
Midi2wav;   %Creates an additional wav file.
%
movefile('mzuri.wav','caller10.wav')% to change the name back
%
% To listen to the midi sounds
pause
%
! mplayer2 caller10.mid

if v==1
 movefile('caller10.mid','mosaic.mid')
elseif v==2
 movefile('caller10.mid','gingerman.mid')
else
 movefile('caller10.mid','dream.mid')
end
=====================================================
```

2. Chaos and fractal programs

These are the programs that generate the fractals. Some have been designed to create video files(*.avi). The video files are then played using the 'mplayer2.exe' or 'wmplayer.exe' from Matlab platform.

===

TEST2.m

```
% TEST2
%  Iterated Function System using 3 linear maps
%
```

```
clear C
disp(' ')
C = input('Enter a 2x2 matrix in which each row contains the slope and...
intercept:\n\n C = ')
disp(' ')
a1 = C(1,1); b1 = C(1,2); a2 = C(2,1); b2 = C(2,2);
%
x(1) = 0;
%
for kk = 1:1000
   P(kk) = fix(2*rand);
   if P(kk) == 0
      x(kk+1) = a1*x(kk) + b1;
   else
      x(kk+1) = a2*x(kk) + b2;
   end;
end;

subplot(2,1,1), plot(x(800:1000),'.'),title(' Time Series ')
subplot(2,1,2), plot(x(800:1000),x(801:1001),'.','color','r')...
xlabel('x(n)'), ylabel(' x(n+1) ')
%
pause
subplot(111)
stairs((800:1000),x(800:1000)),title(' Stair Plot ')
pause
hist(x(900:1000)),axis([0 1 0 100]),title(' Histogram ')
%
pause
polar(2*pi*x(800:1000),ones(1,201),'.')
xlabel( 'Limit Points on the Circle ')
pause

% From here we go to create a signal and process it.
signal_IFS
```

==

TEST3.m

```
% Iterated Function System using 3 linear maps
%
clear C
```

```
disp(' ')
C = input('Enter a 3x2 matrix in which each row contains the slope and...
%intercept:\n\n C = ')
disp(' ')
%
a1 = C(1,1); b1 = C(1,2); a2 = C(2,1); b2 = C(2,2); a3 = C(3,1); b3 = C(3,2);
%
x(1) = 0;
%
for kk = 1:1000
  P(kk) = fix(3*rand);
  if P(kk) == 0
    x(kk+1) = a1*x(kk) + b1;
  elseif P(kk) == 1
    x(kk+1) = a2*x(kk) + b2;
  else
    x(kk+1) = a3*x(kk) + b3;
  end;

end;
subplot(2,1,1), plot(x(800:1000),'.'),title(' Time Series ')
%
subplot(2,1,2), plot(x(800:1000),x(801:1001),'.','color','r')
xlabel('x(n)'), ylabel('x(n+1)')
%
pause
subplot(111)
stairs((800:1000),x(800:1000)),title(' Stair Plot ')
pause
hist(x(900:1000)),axis([0 1 0 100]),title(' Histogram ')
pause
%
polar(2*pi*x(800:1000),ones(1,201),'.')
xlabel( 'Limit Points on the Circle ')
pause

%
signal_IFS
```

==

TREE.m

```
function [xpts,s] =Tree(n)

% This program creates a fractal tree with
```

```
%different values of the affine transformation.
%

A1 = [0 0; 0 .5];  A2 = [.42 -.42; .42 .42];
A3 = [0.42 .42; -0.42 0.42]; A4 = [0.1 0;0 0.1];
T1 = [0 ; 0];  T2 = [ 0 ; .2];
T3 = [0 ; .2];   T4 = [0; .2];
%
P1 =.05; P2 = .4; P3 = .4; P4 = .15;   % Probabilities
%
clf
%
s=rand(2,1); xpts = s;
plot(s(1),s(2),'.'), hold on,
mov = avifile('Tree.avi','keyframe',50,'fps',10)
%
for j=1:n
   r=rand;
     if r <= P1, s=A1*s+T1;
     elseif r <= P1+P2, s=A2*s+T2;
     elseif r <= P1+P2+P3, s=A3*s+T3;
    else s=A4*s;
      end

plot(s(1),s(2),'.');
xpts = [xpts s];
G=getframe;     % creation of a movie by getting frames
mov=addframe(mov,G);
end
mov=close(mov)
hold off
```

==

SIERP.m

```
% This program creates a Sierpinski triangle

function [xpts,s] = Sierp(n)

%
A1 = [.5 0; 0 .5];  A2 = [0.5 0; 0 .5];
A3 = [.5 0; 0 .5];
T1 = [150 ; 150];  T2 = [ 0 ; 150];
```

```
T3 = [75 ; 0];
%
P1 =.85; P2 = .07; P3 = .08;    % Probabilities
%
clf
%
s=rand(2,1); xpts = s;
plot(s(1),s(2),'.'), hold
mov = avifile('Sierp.avi','keyframe',50,'fps',10) % creates an avifile
%
for j=1:n
   r=rand;
   if r <= P1, s=A1*s+T1;
   elseif r <= P1+P2, s=A2*s+T2;
   else  s=A3*s+T3;
   end
   plot(s(1),s(2),'.')
   xpts = [xpts s];
G=getframe;
mov=addframe(mov,G);
end
mov=close(mov)
hold off
```

==

FERN.m

```
function [xpts,s]=fern(n)

% This program creates a fractal fern with the probabilties as :

A1 = [.85 .04; -.04 .85];  A2 = [-.15 .28; .26 .24];
A3 = [.2 -.26; .23 .22];   A4 = [0 0; 0 .16];
T1 = [0 ; 0.2];  T2 = [ 0 ; .2];
T3 = [0 ;0.2];   T4 = [0; 0];
%
P1 =.85; P2 = .07; P3 = .07; P4 = .01;   % Probabilities
%
clf
%

s=rand(2,1); xpts = s;
plot(s(1),s(2),'.'), hold on;
mov = avifile('fern.avi','keyframe',50,'fps',10)
%
```

```
for j=1:n
   r=rand;
     if r <= P1, s=A1*s+T1;
     elseif r <= P1+P2, s=A2*s+T2;
     elseif r <= P1+P2+P3, s=A3*s+T3;
     else s=A4*s;
     end
plot(s(1),s(2),'.');
xpts = [xpts s];
G=getframe;
mov=addframe(mov,G);
end
mov=close(mov)
hold off
```

===

FERN1.m

```
function [xpts,s] = fern1(n);

% This program creates a fractal fern with different values of T vector.
%
A1 = [.85 .04; -.04 .85];  A2 = [-.15 .28; .26 .24];
A3 = [.2 -.26; .23 .22];   A4 = [0 0; 0 .16];
T1 = [0 ; 1.6];  T2 = [ 0 ; .44];
T3 = [0 ; 1.6];   T4 = [0; 0];
%
P1 =.85; P2 = .07; P3 = .07; P4 = .01;   % Probabilities
%

%
s=rand(2,1); xpts = s;
plot(s(1),s(2),'.'), hold on
mov = avifile('fern1.avi','keyframe',50,'fps',10)
%
for j=1:n
   r=rand;
   if r <= P1, s=A1*s+T1;
   elseif r <= P1+P2, s=A2*s+T2;
   elseif r <= P1+P2+P3, s=A3*s+T3;
   else s=A4*s;
   end
   plot(s(1),s(2),'.')
   xpts = [xpts s];
G=getframe;
```

```
mov=addframe(mov,G);
end
mov=close(mov)
hold off
```

==

MOSAIC.m

```
% Mosaic

%this program creates a mosaic pattern.

function [Z] = mosaic
clf

x=-50:0.5:50;
y=x;
[X,Y]=meshgrid(x,y);
%
Z=(cos(X))^2+(cos(Y))^2;
W=exp(abs(Z));

colormap prism(256)
pcolor(W);shading interp
axis('equal','square','off')
```

==

GINGERMAN.m

```
% this program produces the gingerbread man

function [X1,Y1] =gingerman(n)

n=input('Input the number of points \n n=');

if n>=30
        error(['use a maximum of 20']);
end
mov = avifile('gingerman.avi','keyframe',5,'quality',100)
```

```
%
clf
%
for nn = 2:12
x(1) = 0.1 + (nn-2)/10;
y(1) = 0.1 + (nn-2)/10;
for k = 1:n
x(k+1) = 1 - y(k) + abs(x(k)) ;
y(k+1) = x(k) ;
end;

%
plot(x,y,'*')
F=getframe;
mov=addframe(mov,F);
hold on
end
mov=close(mov)
 X1= x;
 Y1= i.*y;

 Random_IFS
```

==

DREAM.m

```
%This fractal is used to create the Kings dream

clf
%

function [X1,Y1] =dream(N);
if nargin ~= 1
        error(['One input argument is required.']);
elseif N<=100
   error(['use a minimum of 200']);
end

a=-0.966918; b=2.879879; c= 0.765145;
d=0.744728;
x(1)=rand(1);
y(1)=rand(1);
```

```
mov = avifile('dream.avi','quality',100)

for n=1:N
x(n+1) = sin(y(n) * b) + c * sin(x(n) * b);
y(n+1) = sin(x(n) * a) + d * sin(y(n) * a);
x(n)=x(n+1);
y(n)=y(n+1);
plot(x,y,'.')

F=getframe;
mov=addframe(mov,F);
hold on
end
mov=close(mov)

 X1= x;
 Y1= i.*y;

 Random_IFS
```

===

3. DSP PROGRAMS

SIGNAL_IFS.m

```
%   SIGNAL_IFS    Fourier Spectrum of IFS Time Series
%
signl = x(745 :1000);
N=length(signl);
del_t = 1;  %sampling rate set as 1
k = 0:N-1;
%
%
subplot(2,1,1), plot(745 + k*del_t,signl-mean(signl)), xlabel( 'n' )
title(' Signal about the Mean Value ')
%
% Compute and plot the frequency spectrum
XX = fft(signl-mean(signl),500);
magXX = abs(XX);
Hertz  = k*(1/(N*del_t));
subplot(2,1,2), plot(Hertz(1:N/2),magXX(1:N/2))
```

```
   title(' Magnitude of XX(k) '),...
   xlabel( 'Hz' ), ylabel( ' |XX(k)| '), grid
%
subplot(111)
pause
stem(Hertz(1:N/2),magXX(1:N/2)), title(' Magnitude of XX(k) '),...
   xlabel( 'Hz' ), ylabel( ' |XX(k)| '), grid
```

==

RANDOM_IFS.m

```
% Random_IFS
% The objective here is to process the sample points.What is needed is a
% signal as the output.The signal is further mapped into Midi by other
% programs.
% The x-component of the sample points is combined with the y-component.

 %
 S=sqrt(real(X1).^2 + imag(Y1).^2);
 %
 S1=S-mean(S);          % mean value
 S11=abs(S1)/max(S1);   % Normalization

N=length(S1);
del_t = 1;
k = 1:N;
S1=S11(100:length(S11));
S2 =abs(fft(S1,10000)); % 10,000 pnts needed for the wave file
S2=S2/max(S2);

Hertz  = k*(1/(N*del_t));
pause
figure(2)
if S2~=0;
subplot(2,1,1), plot(Hertz(100:(N-1)/2),S2(100:(N-1)/2))
    title(' Magnitude of S2(k) '),...
   xlabel( 'Hz' ), ylabel( ' |S2(k)| '), grid
subplot(212)
stairs(Hertz(100:(N-1)/2),12*S2(100:(N-1)/2)/max(S2),'.'),title(' Stairs of S2(k) '),...
 xlabel( 'Hz' ), ylabel( ' |S2(k)| '), grid
%
% Create a wav file
wavwrite(S2,22050,'random') % this makes a wav file
else
```

```
notes=12*S11(1:N-1)/max(S11);
 stairs(notes),title(' Stairs of notes '),xlabel( 'notes' ), ylabel( ' scale ')
end
%
FF=floor(notes*10);
%
F=num2str(FF);
%
notelabels
%
```

==

COMPLEXNOTES.m

```
% for  complex fractals:This is the Processing fashion for the complex
% fractals. There is much of NaN involved and they are replaced with zero
% for the sake of mapping to sound

disp(' ')
NN = input( '  Enter the number of notes to display NN : ' )
%

s=floor(nansum(W)); % Get the susm ignorig the NaN's
k=num2str(s);       % Convert to a string inorder to get rid of the final NaN
s=strrep(k,'NaN','0'); % replace the NaN with zero for the sake of plotting
s=str2num(s);       % Convert back to  numbers because plot cannot use 'strings'.

if length(s)>1000  % Limit the number of values to 1000
   length(s)=1000;
end
NN1 = length(s)-NN+1;
figure(2)
stairs(NN1:length(s),s(NN1:length(s)))
title('Note Sequence'), ylabel(' Scale Values')
%
s=s/max(s);
notes = round(127*s(NN1:length(s)));  % scale to range of midi numbers
%
F=num2str(notes);
%
notelabels
%
drumlabels
```

==

ALLOCATION.m

```
% this program allocates the points obtained from the fractals and renames
%the axis values
X1=xpts(1,:)';
 Y1= i.*xpts(2,:)';
 %
 Random_IFS
=====================================================================
```

WAVCONVERTER.m

```
%%===================================================================
%********** REF*******************
% [W5]
%====================================================================


% Get the energy of a wave file, piece by piece.
% Analyze the energy to determine the overall tempo,
%  when each note was started and stopped,
%  the pitch of each note
%  and the volume of each note

% k = the actual final output matrix
%    Each entry in k has 8 parts:
%       1) when the note starts (measure subdivision)
%       2) when the note starts (quarter subdivision)
%       3) when the note starts (smaller note subdivision)
%    NB: I have combined the first 3 divisions to suit the application file that
%        converts it to midi file.
%       4) when the note ends (measure subdivision)
%       5) when the note ends (quarter subdivision)
%       6) when the note ends (smaller note subdivision)
%     NB: I have combined these 3 divisions to suit the application file that
%        converts it to midi file.
%       7) whether the note was a kick, snare, high hat hit or a ding
%       8) how loud the note was
%    The very first entry of k is the tempo (clicks per second)

% a = the original sound wav file

% quantizeLevel = what level of notes the MIDI should be quantized to
% For instance:  1/16 = sixteenth notes, 1/32 = thirty-second notes
%               1/8 = eigth notes,     1/24 = swing sixteenth notes
function k = Wavconverter(a, quantizeLevel)
```

```
[filename,path]=uigetfile('*.*','Choose .WAV Input Sound File');
mywav = [path,filename];
a=wavread(mywav);
b = length(a);
countm = (1:b);
c = size(a);
e = zeros(c);          % e = bar graph of when sound occurs
f = zeros(c);          % f = a, when sounds are compressed to the same volume
g = zeros(c);          % g = FFT of each sound in a
h = zeros(c);          % h = avg FFT for each sound in a, given by the equation:
                   %    avgFFT = (Sigma(k = 1 to (N/2))[FFT(a)*k])
                   %          /(Sigma(k = 1 to (N/2))[FFT(a)])
j = zeros(c);          % j = avg FFT for each sound in a, compressed over time
ktemp = zeros(1,8);   % ktemp = k before it's adjusted to take the tempo into
                            % account
k = zeros(1,8);        % k = the actual final output matrix
kIndex = 2;          % kIndex is used to determine how many notes were in the wav
soundEnergyCutoffLevel = 0.03;
                   % If any collective form of energy adds up to being less than
              % soundEnergyCutoffLevel, then it is ignored as being background noise
avgEnergyAmt = 0.005; % Used to compress the wav file so that all the sounds
                   %  will be the same volume, and thus matrix 'f' can be created
collectiveEnergy = 0; % Built up over the time of each sound, and then divided
                   %  by the average energy amount in order to compress the sound
                   % to the same volume as all other sounds; also used to determine
                  % the sound's volume
energyStart = 0;      % Used to record when each note begins
energyEnd = 0;        % Used to record when each note ends
energyOldEnd = 0;   % Used to record when the previous note ends to avoid
                            % blips
energyLevel = 0;       % Used to record when the previous note starts to avoid
                         blips
     % A blip is an unintentional sound, such as collective energy incorrectly
     % gathered  at the end of one sound and treated as a new sound.
energyTime = 0;        % Used to record how long a sound lasted
energyRatio = 0;       % Ratio between the collective energy of a sound and the
                   % average energy; used to compress the sound's volume
soundHit = 0;          % Used to determine if a sound is currently occuring
matlabRatio = 1000;   % Make all output numbers in the same range so that the
              % C program inputs the exact values without any rounding errors

jValue = 0;       % Each value in matrix 'h' divided by the time that value occurred

kickHit = 10000;      % Used to represent that the sound was of a kick
dingHit = 30000;      % ...                              ding
snareHit = 50000;     % ...                              snare
```

```
highHatHit = 60000;   % ...                       high hat
ksDiff = 0.095;       % Used to determine the difference in frequency between  a
                       % kick and a snare
hsDiff = 0.26;        % ...                            a high hat and a snare
dingTime = 3500;      % Used to show if the note was long enough to be a ding

          % These beat variables are used to determine when each note started
          % and ended, in terms of measures, quarters, and any smaller beats,
            % depending on what the quantize level was set to.
beatsPerQuarter = 24;
beatsPerMeasure = 4 * beatsPerQuarter;
quantizeLevel=1/16;
beatsPerQuantize = beatsPerQuarter * 4 * quantizeLevel;
samplingTimeRatio = 100000/22050;
blipDifference = beatsPerQuantize * samplingTimeRatio;
            % blipdiffernce makes sure no blips are recorded starting and
            % ending before  the end of the previous notes

% amount of samples for each block checked... 22050 samples = 1 second
delta_amnt = 20;

% In this for loop, the program does an initial scan of the wav file, 20 samples at
a time.
% Here the program discovers:
%     when each note starts
%     when each note ends
%     the pitch of each note
%     the volume of each note
for i = 1:delta_amnt:b
  if (i + delta_amnt) < b
    % Find energy in the 100-unit block
    eBlock = (mean(abs(a(i:(i + delta_amnt)))));
    if eBlock > soundEnergyCutoffLevel
      soundHit = 1;
    else
      soundHit = 0;
    end
    % Here is where a sound just started, so set the energyStart flag and begin
    % recording the collective energy of the sound.
    if energyLevel - soundHit == -1
      energyStart = i;
      energyLevel = 1;
      collectiveEnergy = eBlock;
      energyOldEnd = energyEnd;
    else
      collectiveEnergy = collectiveEnergy + eBlock;
      % Here is where a sound just ended, so
      if energyLevel - soundHit == 1
         energyEnd = i;
         energyLevel = 0;
         energyTime = energyEnd - energyStart;
% To avoid small blips at the end of a sound, check to make sure that no
%  sounds  are accidentally included too close to the end of the previous sound
         if energyEnd > (energyOldEnd + blipDifference)
            % e simply records when each sound starts and stops
            e(energyStart:energyEnd) = 1;
            energyRatio = (energyTime * avgEnergyAmt) / collectiveEnergy;
            % f takes each sound and compresses it, normalizing the volume
            f(energyStart:energyEnd) = ...
               energyRatio * (a(energyStart:energyEnd));
            % g is the fft of each individual sound
            g(energyStart:energyEnd) = ...
               (abs(fft(f(energyStart:energyEnd))));
            energyTimeHalf = energyTime/2;
            energyMid = energyStart + energyTimeHalf;

      % For some reason, g gets inverted at times,  this code deals with that
            [flipReason, tempNumber] = size(g(energyStart:energyMid));
            % Calculate h using the equation:
            %      h = (Sigma(k = 1 to (N/2))[FFT(a)*k])
            %         /(Sigma(k = 1 to (N/2))[FFT(a)])
            if flipReason == 1
               h(energyStart:energyEnd) = ((countm(1:energyTimeHalf + 1))* ...
                  ((g(energyStart:energyMid))'))/ ...
                  (sum(g(energyStart:energyMid)));
            else
               h(energyStart:energyEnd) = ((countm(1:energyTimeHalf + 1))* ...
                  (g(energyStart:energyMid)))/ ...
                  (sum(g(energyStart:energyMid)));
            end
            jValue = h(energyStart:energyEnd)/energyTime;
            j(energyStart:energyEnd) = jValue;
            % Put when the sound started and ended into ktemp.
      % NB: The start and stop values are temporary and will be quantized
      %     once the tempo is determined and added to k (see next FOR loop)
            ktemp(kIndex,1) = energyStart;
            ktemp(kIndex,4) = energyEnd;
            % Determine the volume of the sound.
            % Quieter sounds will be given a velocity of 75.
            % The louder a sound, the closer its velocity will be to 127.
            k(kIndex,8) = ...
               (floor(75 + (atan(collectiveEnergy) * (57/(pi/2))))/1);
```

```
                if k(kIndex,8) > 127
                   k(kIndex,8) = 127 ;
                end
                k(kIndex,8) = k(kIndex,8) * matlabRatio;
                % Increase the k index.
                kIndex = kIndex + 1;
             else
                collectiveEnergy = 0;
             end
          end
       end
    else
       e(i:b) = 0;
    end
end


% Figure out tempo by taking the difference of when the first two notes started.
%  Then multiply that tempo by the ratio between the recording sampling rate
%  (22050 samples/second) and the MIDI playback rate (100000 micros/second)
tempo = ktemp(3,1) - ktemp(2,1);
k(1,1) =  tempo * samplingTimeRatio;
tempoRatio = beatsPerQuarter/tempo;

% These two variables are used to compare the time the current note started and
stopped
%  to when previous notes started and stopped, and thus discover its rhythm
relative to
%  the previous notes
prevHit = ktemp(4,1);
oldBeats = 0;


for i = 4:(kIndex - 1)

    % Figure out what each hit was, depending on the level of its avg frequency,
    %  discovered taking its fft and averaging it above
    drumHit = j(ktemp(i,1));
    if drumHit < ksDiff
       if ktemp(i,4) - ktemp(i,1) < dingTime
          k(i,7) = kickHit;
       else
          k(i,7) = dingHit;
       end
    else
       if drumHit > hsDiff
```

```
      k(i,7) = highHatHit;
    else
      k(i,7) = snareHit;
    end
  end

  % Quantize note starts to fit into tempo
  % amtTime is the raw data of when the note started, based on the last few hits
  amtTime = ((ktemp(i,1) - prevHit) * tempoRatio) + oldBeats;
  % Take that data and quantize it in terms of the quanitize level, rounding it to
  %  the closest quantized beat subdivision (ie sixteenth note, eighth note, etc)
  if (amtTime - (floor(amtTime/beatsPerQuantize) * beatsPerQuantize)) < ...
      (((floor(amtTime/beatsPerQuantize) + 1) * beatsPerQuantize) - amtTime)
    amtHitQuanTime = (floor(amtTime/beatsPerQuantize) * beatsPerQuantize);
  else
   amtHitQuanTime = ((floor(amtTime/beatsPerQuantize) + 1) *...
beatsPerQuantize);
  end
  % To get the value of the beat
  numQuantBeats = 1 + floor(amtHitQuanTime/beatsPerQuantize) *...
beatsPerQuantize;
  k(i,3) = numQuantBeats * matlabRatio;
  if k(i,3) < 0
    k(i,3) = 0;
  end

  % Update previous beat data every five notes to ensure accuracy
  if mod((i + 1), 5) == 0
    prevHit = ktemp(i,1);
    oldBeats = amtHitQuanTime;
  end

  % Quantize note releases to fit into tempo the same way that note starts were
calculated
  amtTime = ((ktemp(i,4) - prevHit) * tempoRatio) + oldBeats;
  if (amtTime - (floor(amtTime/beatsPerQuantize) * beatsPerQuantize)) < ...
      (((floor(amtTime/beatsPerQuantize) + 1) * beatsPerQuantize) - amtTime)
    amtReleaseQuanTime = (floor(amtTime/beatsPerQuantize) *...
beatsPerQuantize);
  else
    amtReleaseQuanTime = ((floor(amtTime/beatsPerQuantize) + 1) *...
beatsPerQuantize);
  end
  if amtReleaseQuanTime == amtHitQuanTime;
    amtReleaseQuanTime = amtReleaseQuanTime + beatsPerQuantize;
  end
```

```
   amtReleaseQuanTime = amtReleaseQuanTime - 1;
numQuarters=1+floor(amtReleaseQuanTime/beatsPerMeasure)+...
numQuantBeats/1.05;
   k(i,4) = numQuarters * matlabRatio;

end

============================================================
```

4. MIDI PROGRAMS

SHOWNOTES.m

```
% SHOW_NOTES
%
disp(' ')
NN = input( '  Enter the number of notes to display  NN : ' )
%
NN1 = 1000-NN+1;
stairs(NN1:1000,round(12*x(NN1:1000))),axis([NN1 1000 0 12])
title('Note Sequence'), ylabel(' Scale Values')
%
notes = round(12*x(NN1:1000));
cmatrix = C;
prob = P;
F=num2str(notes);
```

==

Notelabels.m

```
% This program converts the notes to the keys and hence we obtain an
%acoustic signature of the  IFS
%
% this program changes the  Midinote numbers to the names

   F= strrep(F,'0','C');
   F= strrep(F,'1','Db');
   F= strrep(F,'2','D');
   F= strrep(F,'3','Eb');
   F= strrep(F,'4','E');
   F= strrep(F,'5','F');
   F= strrep(F,'6','Gb');
   F= strrep(F,'7','G');
   F= strrep(F,'8','Ab');
   F= strrep(F,'9','A');
   F= strrep(F,'DbC','Bb');
   F= strrep(F,'DbDb','B');
   F= strrep(F,'DbD','C');
   F= strrep(F,'DbEb','Db');
   F= strrep(F,'DbE','D');
   F= strrep(F,'DbF','Eb');
   F= strrep(F,'DbGb','E');
   F= strrep(F,'DbG','F');
```

Missing Page

```
F= strrep(F,'GbE','E');
F= strrep(F,'GbF','F');
F= strrep(F,'GbGb','Gb');
F= strrep(F,'GbG','G');
F= strrep(F,'GbAb','Ab');
F= strrep(F,'GbA','A');
F= strrep(F,'GC','Bb');
F= strrep(F,'GDb','B');
F= strrep(F,'GD','C');
F= strrep(F,'GEb','Db');
F= strrep(F,'GE','D');
F= strrep(F,'GF','Eb');
F= strrep(F,'GGb','E');
F= strrep(F,'GG','F');
F= strrep(F,'GAb','Gb');
F= strrep(F,'GA','G');
F= strrep(F,'AbC','Ab');
F= strrep(F,'AbDb','A');
F= strrep(F,'AbD','Bb');
F= strrep(F,'AbEb','B');
F= strrep(F,'AbE','C');
F= strrep(F,'AbF','Db');
F= strrep(F,'AbGb','D');
F= strrep(F,'AbG','Eb');
F= strrep(F,'AbAb','E');
F= strrep(F,'AbA','F');
F= strrep(F,'AC','Gb');
F= strrep(F,'ADb','G');
F= strrep(F,'AD','Ab');
F= strrep(F,'AEb','A');
F= strrep(F,'AE','Bb');
F= strrep(F,'AF','B');
F= strrep(F,'AGb','C');
F= strrep(F,'AG','Db');
F= strrep(F,'AAb','D');
F= strrep(F,'AA','Eb');
F= strrep(F,'BbC','E');
F= strrep(F,'BbDb','F');
F= strrep(F,'BbD','Gb');
F= strrep(F,'BbCEb','G');
F= strrep(F,'BbE','Ab');
F= strrep(F,'BbF','A');
F= strrep(F,'BbGb','Bb');
F= strrep(F,'BbG','B');
F= strrep(F,'BbAb','C');
F= strrep(F,'BbA','Db');
```

```
  F= strrep(F,'BC','D');
  F= strrep(F,'BDb','Eb');
  F= strrep(F,'BD','E');
  F= strrep(F,'BEb','F');
  F= strrep(F,'BE','Gb');
  F= strrep(F,'BF','G');
  F= strrep(F,'BGb','Ab');
  F= strrep(F,'BG','A');
  F= strrep(F,'BAb','Bb');
  F= strrep(F,'BA','B');
  F= strrep(F,'CC','C');
  F= strrep(F,'CDb','Db');
  F= strrep(F,'CD','D');
  F= strrep(F,'CEb','Eb');
  F= strrep(F,'CE','E');
  F= strrep(F,'CF','F');
  F= strrep(F,'CGb','Gb');
  F= strrep(F,'CG','G');
```

===

Drumlabels.m

```
% This program converts the keys to drum midi numbers

Clear Keys

Keys=F;

  Keys= strrep(Keys,'Bb','46');
  Keys= strrep(Keys,'B','35');
  Keys= strrep(Keys,'C','36');
  Keys= strrep(Keys,'Db','37');
  Keys= strrep(Keys,'D','38');
  Keys= strrep(Keys,'Eb','39');
  Keys= strrep(Keys,'E','40');
  Keys= strrep(Keys,'F','41');
  Keys= strrep(Keys,'Gb','42');
  Keys= strrep(Keys,'G','43');
  Keys= strrep(Keys,'Ab','44');
  Keys= strrep(Keys,'A','45');
  Keys= strrep(Keys,'Bb','46');

  Keys=str2num(Keys);
  Keys=abs(Keys);  % to put a check on negative values.
  drumtext
```
===

```
  pKeys= strrep(pKeys,'Bb','71');

  pKeys=str2num(pKeys);

  pKeys=abs(pKeys);  % to put a check on negative values.

  pianotext
```

===

Pianotext.m

```
% This program crerates a Midi Text from the notes obtained from the 'Test'
% program. This is then converted to a midi file using an executable file.
% x and y values become the values of the note number/position and the 'pKeys'
% indicate the type of key/note to be played.

fp = fopen('piantext.txt','w');

% Print head information into the text file.

fprintf(fp, 'MFile 0 1 24\n');
fprintf(fp, 'MTrk\n');
fprintf(fp, '0 TimeSig 4/4 24 8\n');
fprintf(fp, '0 Tempo 450000\n');
fprintf(fp, '0 PrCh ch=1 p=40\n');  % the value of p shows the instrument type
                                     %  (see appendix 2)

x(1)= 1;     % initial values x(1) and y(1) picked at choice.
           %The disparartion between the two is for making them quite audible.
y(1)=6;
%
for m=1:length(pKeys);
  y(m)=x(m)+13;
  x(m+1)=y(m)+4;
 fprintf(fp,'%d On ch=1 n=%2d vol=120\n ',x(m),pKeys(m));     % quite an
                                                             % audible volume.
 fprintf(fp,'%d On ch=1 n=%2d vol=0\n ',y(m),pKeys(m));      % The volume is 0
end                                                   % to ensure the note is turned off
                                                         %and ready for the next one.

%
fprintf(fp, 'TrkEnd\n');        % the end of the track.

status = fclose(fp);            % closing the file.
```

===

Multimix.m

```
% Multimix.m

% This program creates a Midi Text from the notes obtained from the
% fractals. This is one is particularly modified to have multiple
% instruments in the midi file. It is designed to have 3 tracks.

fp = fopen('mixtext.txt','w');
% Print head information into the text file.
fprintf(fp, 'MFile 0 3 24\n');  % format of file is 0, and it has 3 tracks.
fprintf(fp, 'MTrk\n');
fprintf(fp, '0 TimeSig 4/4 24 8\n');
fprintf(fp, '0 Tempo 550000\n');
x(1)= 1;
y(1)=6;
for m=1:length(Keys);  % The instrument here is percussion
  y(m+1)=x(m)+12;        % which is always channel 10.
  x(m+1)=y(m)+4;
 fprintf(fp,'%d On ch=10 n=%2d vol=120\n ',x(m),Keys(m));
 fprintf(fp,'%d On ch=10 n=%2d vol=0\n ',y(m),Keys(m));
end
fprintf(fp, 'TrkEnd\n');
%  begin another track
fprintf(fp, 'MTrk\n');
fprintf(fp, '0 PrCh ch=2 p=35\n'); % The instrument is no. 35 on channel 2
for m=1:length(Keys);
  y(m+1)=x(m)+11;
  x(m+1)=y(m)+4;
 fprintf(fp,'%d On ch=2 n=%2d vol=120\n ',x(m),pKeys(m));
 fprintf(fp,'%d On ch=2 n=%2d vol=0\n ',y(m),pKeys(m));
end
fprintf(fp, 'TrkEnd\n');
%  begin another track
fprintf(fp, 'MTrk\n');
fprintf(fp, '0 PrCh ch=3 p=70\n');  % The instrument is no. 70 on channel 3
for m=1:length(Keys);
  y(m+1)=x(m)+11;
  x(m+1)=y(m)+4;
 fprintf(fp,'%d On ch=3 n=%2d vol=120\n ',x(m),pKeys(m));
 fprintf(fp,'%d On ch=3 n=%2d vol=0\n ',y(m),pKeys(m));
end
fprintf(fp, 'TrkEnd\n');
status = fclose(fp);
```

==

■■■

A word on using the compiler...

After coming up with a matlab matrix from 'Fernprog3' we can create a MIDI text file using a C program.
The first is to configure the default options file to create MEX-files using the compiler. (This has been accounted for in the 'Caller' files).

The switch (entered in at the matlab prompt) is:

mex –setup

and the following comes up :

Please choose your compiler for building external interface (MEX) files.

Would you like mex to locate installed compilers [y]/n? n

Select a compiler:
[1] Compaq Visual Fortran version 6.6
[2] Lcc C version 2.4
[3] Microsoft Visual C/C++ version 6.0

[0] None

Choose the "Lcc C version 2.4" *[2]*

To compile the program will need to type in matlab:

mex mFormat.c

mFormat("matlabmatrix",'miditext.txt') where "matlabmatrix " is the output from 'wavconverter.m' , and 'miditext.txt ' is the name of the text file being created.

****

mFormat.c

```
/*=========================================================
* mFormat.c
**************REF***********[W5]
* Takes a matrix and compiles it to become a midi file text
*
=========================================================
#include <stdio.h>
#include "mex.h"

// This is used to print a number and add the appropriate spaces,
//  depending on if it is one, two or three digits long.
void PrintNumber(FILE *theFile, int theNumber)
{
  if (theNumber >= 100) {
    fprintf(theFile, "%d", theNumber);
  } else if (theNumber < 10) {
    fprintf(theFile, "  %d", theNumber);
  } else {
    fprintf(theFile, " %d", theNumber);
  }

  return;
}

// This is the main function.
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
  double *x;                    // x is the input matrix
  char *y;                     // y is the name of the text file created
  int mrows;                    // number of entries (ie., rows) in x
  int i;                    // index variable
  int buflen;                  // number of characters in y
  int status;                  // makes sure y is the full input string
  int tempo;                    // the tempo of the song, in microseconds
  int volume;                   // the volume of each note

  int thirdCol,seventhCol;
  int eighthCol;                 // constant added to the index of x in
                          //  order to access that many columns over

  int curSubBeat = 0;  //  quantized beat

  int matlabRatio = 1000;        // used to make sure C doesn't round off any
                          //  input number
  int fixedQ, fixedQQ;    // the newest beat on and off respectively

  int kickHit = 10000;          // the value in x used to represent a kick
  int dingHit = 30000;          //  hit, ding hit, snare hit or high hat hit
  int snareHit = 50000;
  int highHatHit = 60000;
```

```
FILE *fp;                    // the text file written to

/* Check for proper number of arguments. */
if(nrhs!=2) {
  mexErrMsgTxt("Exactly two inputs required: MIDI matrix and MIDI file name.");
}

/*  Get the dimensions of the matrix input x. */
mrows = mxGetM(prhs[0]);

/* Second input must be a string. */
  if ( mxIsChar(prhs[1]) != 1)
    mexErrMsgTxt("Second input must be a string.");

/* Assign a pointer to the first input. */
x = mxGetPr(prhs[0]);

/* Input must be a row vector. */
if (mxGetM(prhs[1])!=1)
    mexErrMsgTxt("Input must be a row vector.");

/* Get the length of the input string. */
buflen = (mxGetM(prhs[1]) * mxGetN(prhs[1])) + 1;

/* Allocate memory for input and output strings. */
y = mxCalloc(buflen, sizeof(char));

/* Copy the string data from prhs[1] into a C string
 * input_ buf.
 * If the string array contains several rows, they are copied,
 * one column at a time, into one long string array.
 */
status = mxGetString(prhs[1], y, buflen);
if(status != 0)
  mexWarnMsgTxt("Not enough space. String is truncated.");

tempo = x[0];
thirdCol = 2 * mrows;
seventhCol = 6 * mrows;
eighthCol = 7 * mrows;

fp = fopen(y, "w");

// Print head information into the text file.
fprintf(fp, "MFile 0 1 24\n");
fprintf(fp, "MTrk\n");
```

```
  fprintf(fp, "0 TimeSig 4/4 24 8\n");
fprintf(fp, "0 Tempo 600000\n");

  // Put in the beats using the MIDI matrix.
  for (i = 3; i < mrows; i++) {

    // On Note

    fixedQ = x[i + thirdCol] / matlabRatio;
   // Print the sub-beat number recognized cumulatively.
    if (fixedQ > curSubBeat) {
      PrintNumber(fp, fixedQ);
      curSubBeat = fixedQ;
    }

    fprintf(fp, " On ch=10 n=");
    // Print the note's pitch
    if (x[i + seventhCol] == kickHit) {
      fprintf(fp, "36");
    } else if (x[i + seventhCol] == dingHit)  {
      fprintf(fp, "68");
    } else if (x[i + seventhCol] == snareHit) {
      fprintf(fp, "40");
    } else {
      fprintf(fp, "42");
    }

    // Print the note's volume
    fprintf(fp, " v= ");
    volume = x[i + eighthCol] / matlabRatio;
    PrintNumber(fp, volume);
    fprintf(fp, "\n");

    // Off Note -- code is similar, but with a few changes and relies on many
    //          variables, so I didn't want to make it a separate function
fixedQQ = x[i + thirdCol] / matlabRatio + 4;
   // Print the sub-beat number recognized cumulatively.
    if (fixedQ > curSubsBeat) {
      PrintNumber(fp, fixedQQ);
      curSubsBeat = fixedQQ;
    }

    fprintf(fp, " On ch=10 n=");
    if (x[i + seventhCol] == kickHit) {
      fprintf(fp, "36");
    } else if (x[i + seventhCol] == dingHit) {
      fprintf(fp, "68");
    } else if (x[i + seventhCol] == snareHit) {
      fprintf(fp, "40");
    } else {
      fprintf(fp, "42");
    }

    // Print the note's volume
    fprintf(fp, " v= 0");
    fprintf(fp, "\n");


  }


  fprintf(fp, "TrkEnd\n");
  fclose(fp);

  return;
}

/* End *.

==========================================================
```

7.0 DISCUSSION AND CONCLUSION

The creating of acoustic signatures for chaos and attractors was achieved .One of the challenges was to convert a wav file made from fractal data to midi. A Midi disassembler which is encouraged by other users was unattainable. The programs used to obtain a midifile from a text format and vice versa were mf2t.exe and t2mf.exe. All other midi programs have been created to suit these programs.

For the Iterated function system (IFS) fractals, the location of the points on the plot were mapped into midi.Taking the y axis as an imaginary axis the abstract value was computed by combining the x axis as real and y axis as imaginary(as shown in 'Random_IFS.m).

To obtain the acoustic signature for the complex fractals, the color values are sought which are in a range of less than 1.It was computed from the exp(abs(Z)) ('Complexnotes.m), where Z, is the function to get the complex fractal. The function used to create the'signal-like' vector was nansum, which is the sum of the non-NaN elements of the matrix. For example, nansum(X) is a row vector containing the sum of the non-NaN elements in each column of X.).So this row vector was mapped into midi.

7.1 RECOMMENDATIONS

1 A real time process that involves reconstruction from the signature to the fractal. This suits the application in the theatre halls.

2 Making a video for the fractals together with the acoustic signature in real time scenario.

3 For a good view of fractals, a computer with much more memory is required.

4 Analysis of the 'signals' before making wav files, using wavelet based signal processing methods.

5 Every acoustic signature for the complex fractals involved data at the last iteration. There is need to be able to make a signature with every iteration.

8 APPENDIX

Appendix 1

8.1 RECONSTRUCTION PROGRAMS

These programs are intended to be used to be able to reconstruct a fractal or even to make a fractal from a MIDI file. The MIDI file is first converted to the text format using the executable file outside of Matlab.
Then using 'midiback.m' we are able to get the matrix of notes that were on. This is the same as notes off, the difference being that the volume for the notes off is zero.
The matrix of the Notes on which is 'NoteOn' can be considered as the positions on a plot like the x axis. Another matrix has to be obtained to form a y axis. This can be left to the discretion of the user, e.g., the sin of 'NoteOn'. Then the two matrices can be used as input parameters in the recipe that generates fractals like for the Julia set or Mandelbrot set. And hence a fractal is obtained.
Steps in reconstruction:

- Record a midi song from an instrument use Music Time Deluxe to save as a midi file. This is then moved to the current directory in Matlab
- In Matlab type in:

  ```
  ! mf2t 'midifile'.mid 'textfile'.txt
  ```

  this creates a text file.
- Run 'Caller11.m'. This creates a matlab matrix of the notes that were played. And uses other programs for reconstructing the fractals.

==

Caller11.m

```
% This program is an attempt to reconstruct the fractals
%from a midifile.

%First step is to change the midi file to a text file.
clc

filename=uigetfile('*.*','Choose .MID Input Sound File');
str=sprintf('mf2t %s mytext.txt',filename);
```

```
unix(str);
%
disp(' a text called "mytext.txt" has been created for next step')
%
Midiback;
%
while 1    clc;
  v = input('select a program\n\n R1(1), R2(2), R3(3), ? ');
    if v >= 1 & v <= 3;        break
    end
 end

switch v
   case 1
      disp('create a Julia fractal');
     [W]= Midijulia;
   case 2
      disp('create a mandelbrot fractal');
      [W] =Midimandel;
   case 3
      disp('create an IFS fractal ');
      [X1,Y1] =Midiginger;

 end
```

===

MIDIBACK.m

```
function midiback(FileName)

% Extracts MIDI 'notesOn' from thetext-format MIDI file
% The midi file is assumed not to have much variation in the channel numbers.
% The only point of interest is the note numbers.
% They display some similarity and yet they are a non-uniformly distributed.
% The only input required is the filename called 'source'
% the text file used must be the one obtained from the conversion of the
% midi file using the 'mf2t' application.(Most likely in big CAPS)

% Name of MIDI text file with extension added
[filename,path]=uigetfile('*.*','Choose .txt Input Sound File');
source = [path,filename];

% Name of this file
myname = 'MIDIBACK: ';
```

```
% Initialize the note-on and note-off arrays
NoteOn = [ ];
NoteOff = [ ];

% Open the .TXT file in text read mode
[fid emessage] = fopen(source,'rt');
if fid == -1
   disp([myname, emessage])
   return
end

% Read the header chunk ID; verify that first five characters
% are as expected
s=fscanf(fid,'%s',1);
if (s ~= 'MFile')
  disp([myname, “” ' source  ' ” is not a text-format MIDI file']);
  disp([myname, '   (expected first five characters to be "MFile" ']);
  return
end

% Read the file type, number of tracks, and division
FileType = fscanf(fid,'%d',1);
NumTracks = fscanf(fid,'%d',1);
Division = fscanf(fid,'%d',1);

% Read all the tracks
for k=1:NumTracks

  % Read the track chunk ID; verify that characters are correct
  s=fscanf(fid,'%s',1);
        if (s ~= 'MTrk')
                disp([myname, ‘ “ ‘ source “ ’ is garbled (1)']);
        return
   end

   % Set flag to indicate when finished reading track
   TrackDone = 0;

   while (TrackDone ~= 1)
   % Read delta time string; test if it represents end of track
   s=fscanf(fid,'%s’,1);
   if (isletter(s(1)))
     % Appears to be a track end label
     if (s == 'TrkEnd')
  disp([myname, ‘Finished reading Track ',num2str(k),' of ',num2str(NumTracks)])
```

```
        TrackDone = 1;
      else
        disp([myname, ' " 'source' " is garbled (2)']);
      end
    else

      % Read the MIDI event name
     MidiEventName = fscanf(fid,'%s',1);

      % Read the MIDI event data
           s = fgetl(fid);
% Parse the MIDI event
      switch (MidiEventName)
          case 'On'
          % Read note number
        loc=findstr(s,'n=');
        MidiNumber = sscanf(s(loc+2:length(s)),'%d');

        % Read velocity
        loc=findstr(s,'v=');
        Volume = sscanf(s(loc+2:length(s)),'%d');

        % Append to note-on array if velocity is nonzero, otherwise
        % append to note-off array
        if (Volume ~= 0)
          NoteOn = [NoteOn; [ MidiNumber]];
        else
          NoteOff = [NoteOff; [ MidiNumber]];
        end
        case 'Off'
           % Read note number
        loc=findstr(s,'n=');
        MidiNumber = sscanf(s(loc+2:length(s)),'%d');

        % Append to note-on array
        NoteOff = [NoteOff; [ MidiNumber]];

      end

    end

    end

end

% Close the file
```

```
fclose(fid);
%
subplot(211)
plot(NoteOn,'.')

% display the last N notes.
N = input( '  Enter the number of notes to display  N : ' )
%
subplot(212)
NN = length(NoteOn)-N+1;
if NN<=0;
   stairs(NoteOn);
   disp('your request is large')
else
stairs(NoteOn(NN:end))
end
title('Note Sequence'), ylabel(' Scale Values')
==============================================================
```

Midijulia.m

```
function  [W]= Midijulia(NoteOn);
% the program cretes a julia fractal from the midi file . the values of x
% and y are obtained by manipulation.
clear
load('NoteOn')    % loads NoteOn.mat
x=(NoteOn)/max(NoteOn);
y=rand(1,length(x))';
d=x+y;
x=sortrows(d);
c = -0.765 + i*0.11;
N = 100;
r = (1:N)'/N;
theta = pi.*x(1:length(N));   % the values of NoteOn.mat come into use here.
XX = r.*cos(theta);
YY = r.*sin(theta);
[X, Y] = meshgrid(XX,YY);
Z=X +i*Y;
 for itr = 1:N,
Z=Z.^2 +c;
W=exp(-abs(Z));
 if abs(Z) > 30,
       N= itr; break,
     end
  end
grid off
```

```
pcolor(W)
axis ('equal', 'tight', 'off')
```

==

Midimandel.m

```
function  [W] =Midimandel;
% the program cretes a mandelbrot fractal from the midi file . the values of x
% and y are obtained by manipulation.
clear
load('NoteOn')    % loads NoteOn.mat
x=(NoteOn)/max(NoteOn);
y=rand(1,length(x))';
d=x+y;
x=sortrows(d);
%
N = 100;
r = (1:N)'/N;
theta = pi.*x(1:length(N));   % the values of NoteOn.mat come into use here.
XX = r.*cos(theta);
YY = r.*sin(theta);
[X, Y] = meshgrid(XX,YY);
Z=zeros(length(X));
C=X +i*Y;
 for k = 1:N,
Z=Z.^2 +C;
W=exp(-abs(Z));
 if abs(Z) > 30,
       N= k; break,
     end
  end
figure(2)
pcolor(W)
axis ('equal', 'tight', 'off')
```

==

Midiginger.m

```
%This program reconstructs the ginger fractal from  midi file outputs.
function [x,y] =midigingerman(n)

clear
load('NoteOn')
```

```
x=sin(NoteOn);
y=rand(1,length(x));

n=input('Input the number of points \n n=');
if n>=30
   error(['use a maximum of 20']);
end
clf

for k = 1:n
x(k+1) = 1 - y(k) + abs(x(k)) ;
y(k+1) = x(k) ;
end;
%
plot(x,y,'*')
pause
axis fill
image
=============================================================
```

OTHER FRACTALS

KOCH1.m

```
%KOCH1(n)
% this program draws the Koch curve fractal.
% n is the number of iterations.
function koch1(n)

n=input('input the value of n\n\ n=')

if nargin ~= 1
        error(['One input argument is required.']);
    elseif n > 6
    error(['Takes too long to give you the curve-use a ' ...
       'maximum of 6']);
end
xl = zeros(10,1);
xr = xl;
yl = xl;
yr = yl;
xr(n) = 1;
r = sqrt(1/3^2 - 1/6^2);
clf;
fig=figure
```

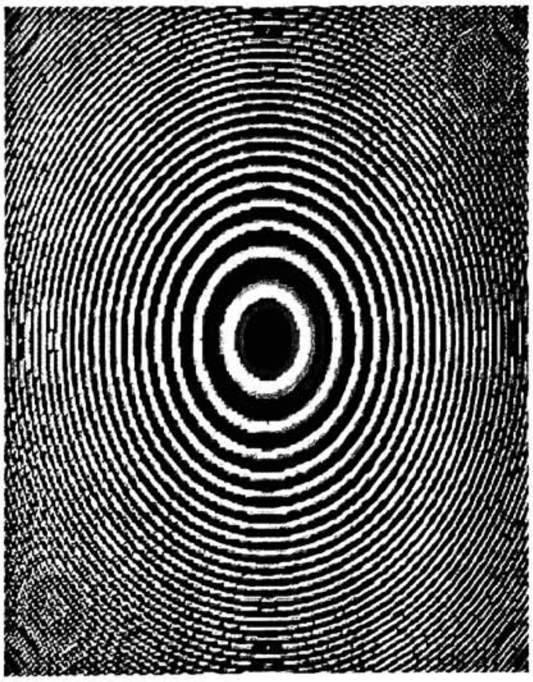
```
 set(fig,'DoubleBuffer','on');
set(gca,'FontSize',8);
set(gcf,'Color',[1,1,1]);
set(gca,'NextPlot','replace','Visible','off')
hold on;
levels(xl,xr,yl,yr,n,r);
title('Koch Curve');
text(0.5,-0.05,(['Number of iterations: ' num2str(n)]), ...
           'HorizontalAlign','center','FontSize',12);
hold off;
axis equal; axis tight; axis off;

%-------------------------------------------------------
function levels(xl,xr,yl,yr,n,r)

if (n<2)
   plot([xl(1) xr(1)],[-yl(1) -yr(1)],'b-')
G=getframe;
   return
end
%doing more iterations
n=n-1;
%
xl(n)=xl(n+1);
yl(n)=yl(n+1);
xr(n)=1/3*xr(n+1)+2/3*xl(n+1);
yr(n)=1/3*yr(n+1)+2/3*yl(n+1);
levels(xl,xr,yl,yr,n,r);
%
xl(n)=xr(n);
yl(n)=yr(n);
xr(n)=.5*xr(n+1)+.5*xl(n+1)-r*(yl(n+1)-yr(n+1));
yr(n)=.5*yr(n+1)+.5*yl(n+1)+r*(xl(n+1)-xr(n+1));
levels(xl,xr,yl,yr,n,r);
%
xl(n)=xr(n);
yl(n)=yr(n);
xr(n)=2/3*xr(n+1)+1/3*xl(n+1);
yr(n)=2/3*yr(n+1)+1/3*yl(n+1);
levels(xl,xr,yl,yr,n,r);
%
xl(n)=xr(n);
yl(n)=yr(n);
xr(n)=xr(n+1);
yr(n)=yr(n+1);
levels(xl,xr,yl,yr,n,r);
```

```
n=n+1;

return;
```

==

Circles.m

```
%this program creates a circular patterns.

x=-50:0.5:50;
y=x;
[X,Y]=meshgrid(x,y);
a=5;
Z = a*(X.^2 + Y.^2);
W=floor(Z);
colormap prism(256)
pcolor(W);
shading interp
axis('equal','square','off')
```



==

Complex fractals

Julia_1.m

```
% Julia_1.m

for k=1:3
x=0;
x=x^2+c;
end
XX=abs(x);
clear x
t=linspace(0,2*pi,1000);
X=XX*(cos(t)+i*sin(t));
Y=c;
plot(real(X),imag(X),'k.');
hold on
%Two images
for k1=3
  Z1=((-1)^k1)*sqrt(X-c);
  plot(real(Z1),imag(Z1),'b.');
  for k2=1:3
    Z2=((-1)^k2)*sqrt(Z1-c);
    plot(real(Z2),imag(Z2),'g.');
  end
end
plot(real(c),imag(c),'r*');
hold off
clear Z1 Z2
pause
plot(real(X),imag(X),'k.');
hold on

%Four preimages
for k1=1:3
  Z1=((-1)^k1)*sqrt(X-c);
  plot(real(Z1),imag(Z1),'b.');
  for k2=1:3
    Z2=((-1)^k2)*sqrt(Z1-c);
    plot(real(Z2),imag(Z2),'g.');
    for k3=1:3
      Z3=((-1)^k3)*sqrt(Z2-c);
      plot(real(Z3),imag(Z3),'y.');
      for k4=1:3
        Z4=((-1)^k4)*sqrt(Z3-c);
```

```
          plot(real(Z4),imag(Z4),'m.');
        end
      end
    end
  end
  plot(real(c),imag(c),'r*');
  hold off
  clear Z1 Z2 Z3 Z4
  pause

  plot(real(X),imag(X),'k.');
  hold on
  for k1=1:3
    Z1=((-1)^k1)*sqrt(X-c);
    plot(real(Z1),imag(Z1),'b.');
    for k2=1:3
      Z2=((-1)^k2)*sqrt(Z1-c);
      plot(real(Z2),imag(Z2),'g.');
      for k3=1:3
        Z3=((-1)^k3)*sqrt(Z2-c);
        plot(real(Z3),imag(Z3),'y.');
        for k4=1:3
          Z4=((-1)^k4)*sqrt(Z3-c);
          plot(real(Z4),imag(Z4),'m.');
          for k5=1:3
            Z5=((-1)^k5)*sqrt(Z4-c);
            plot(real(Z5),imag(Z5),'k.');
            for k6=1:3
              Z6=((-1)^k6)*sqrt(Z5-c);
              plot(real(Z6),imag(Z6),'b.');
              for k7=1:3
                Z7=((-1)^k7)*sqrt(Z6-c);
                plot(real(Z7),imag(Z7),'g.');
              end
            end
          end
        end
      end
    end
  end
plot(real(c),imag(c),'r*');
hold off

=========================================================
```

Julia_2.m

```
%
function [W] = Julia_2(c,N);
%
clf

x=-1.4:0.008:1.0;
y=-1.2:0.008:1.2;
[x0, y0] = meshgrid(x,y);
n = zeros(size(x0));
for k = 1:length(y),
 for m = 1:length(x),
  Z = x0(k,m) + i*y0(k,m);
  for itr = 1:N,
   Z=sin(Z)+cos(Z)+ c;
      W=exp(-abs(Z));
   if abs(Z) < 1,
    n(k,m) = itr; break,
   end,
  end,
 end,
end,
figure(1)
mesh(x0,y0,n);
axis('square','equal','off');
az = 0;
el = 90;
view(az, el);
grid off
```

==

Julia_3.m

```
% Julia_3.m
function [W] = Julia_3(c,N);
%
x=-1.4:0.008:1.0;
y=-1.2:0.008:1.2;
[x0, y0] = meshgrid(x,y);
n = zeros(size(x0));
%
for k = 1:length(y),
 for m = 1:length(x),
   Z = x0(k,m) + i*y0(k,m);
   for itr = 1:N,
     Z=cos(Z)+ i*cos(Z)+ c;
          W=exp(-abs(Z));
     if abs(Z) < 1,
       n(k,m) = itr; break,
     end,
   end,
 end,
end,
figure(1)
mesh(x0,y0,n);
axis('square','equal','off');
az = 0;
el = 90;
view(az, el);
grid off
==========================================================
```

Julia_4.m

```
% A Julia_4.m
% The value of N is for specifying the resolution and not the number of
% iterations mesh(x0,y0,n);

function [W] =julia_4(c,N);
%
x0=-1.3:1.3;
y0=-1.3:1.3;
%
x1 = (max(x) - min(x))/N;
y1 = (max(y) - min(y))/N;
%
[x,y]=meshgrid([-1.3: x1:1.3], [-1.3: y1:1.3]);
Z=x+i*y;
iter = 1;
for iter=1:30
   Z=Z.^2 + c;
   W=exp(-abs(Z));
  end
pcolor(x,y,abs(Z)), shading('flat');
hold on, axis('square');

%colormap(hsv);
colormap([0 0 0.4; 1 1 1]);
```

```
 set(gca,'XTick',[ ],'YTick',[ ]);
```

==

Julia_5.m

```
% Julia_5.m

function [W]= Julia_5(N);
%
x = -1.5:0.008:1.5;
y = -1.5:0.008:1.5;
%
[x0, y0] = meshgrid(x,y);
p = zeros(size(x0));
for k = 1:length(y),
  for m = 1:length(x),
    Z = x0(k,m) + i*y0(k,m);
        for itr = 1:N,
      Z = Z.^2 + c;
      W=exp(-abs(Z));
        if abs(Z) > 10,
        p(k,m) = itr; break,
      end
    end
  end
end
mesh(x0,y0,p);
axis('square','equal','off');
az = 0;
el = 90;
view(az, el)
```

==

Julia_6.m

```
% Julia_6.m

function [W]= Julia_6(c,N);
%
x = -2:0.008:2;
y = -2:0.008:2;
%
```

```
[x0, y0] = meshgrid(x,y);
p = zeros(size(x0));
for k = 1:length(y),
  for m = 1:length(x),
    Z = x0(k,m) + i*y0(k,m);
        for itr = 1:N,
      Z = Z.^6 + Z.^4 +c;
      W=exp(-abs(Z));
        if abs(Z) > 10,
        p(k,m) = itr; break,
      end
    end
  end
end
mesh(x0,y0,p);
axis('square','equal','off');
az = 0;
el = 90;
view(az, el)
```

==

Julia_7.m

```
%Julia_7.m
%
function [W] = Julia_7(c,N);
%
x=-1.4:0.008:1.0;
y=-1.2:0.008:1.2;
[x0, y0] = meshgrid(x,y);
n = zeros(size(x0));
%
for k = 1:length(y),
  for m = 1:length(x),
    Z = x0(k,m) + i*y0(k,m);
    for itr = 1:N,
      Z=sin(Z)+(Z.^2)+ c;
     W=exp(-abs(Z));
      if abs(Z) < 1,
        n(k,m) = itr; break
      end
    end
  end
end
```

```
mesh(x0,y0,n);
axis('square','equal','off');
az = 0;
el = 90;
view(az, el);
```

===

Julia_8.m

```
%Julia_8.m
%
function [W] = Julia_8(c,N);
%
x=-1.4:0.008:1.2;
y=-1.2:0.008:1.0;
[x0, y0] = meshgrid(x,y);
n = zeros(size(x0));

for k = 1:length(y),
 for m = 1:length(x),
   Z = x0(k,m) + i*y0(k,m);
   for itr = 1:30,
     Z=Z.*Z+ c;
     W=exp(-abs(Z));
     if abs(Z) >4,
       n(k,m) = itr; break
     end
   end
 end
end
mesh(x0,y0,n);
axis('square','equal','off');
az = 0;
el = 90;
view(az, el);
grid off
```

===

Julia_9.m

```
%Julia_9.m
% This use pseudo color plot.
function [W] = Julia_9(c,N);
%
p=400;
```

```
x=0;
y=0;
g=1.5;
x=linspace(x-g,x+g,p);
y=linspace(y-g,y+g,p);
[X,Y]=meshgrid(x,y);
Z=X+i*Y;
%
for k=1:N;
Z=Z.^2+c;
W=exp(-abs(Z));
end
colormap prism(256)
pcolor(W);
shading flat;
axis('square','equal','off');
```

==

Julia_10.m

```
%

function [W] = Julia_10(c,N);
%
p=400;          %number of points.
x=0;
y=0;
g=1.5;
x=linspace(x-g,x+g,p);  %creates a vector of p points between 2 limits.
y=linspace(y-g,y+g,p);  %creates a vector of p points between 2 limits.
[X,Y]=meshgrid(x,y);         % creates 3-D surface plots.
 Z=X+i*Y;
%
for k=1:N;
Z=Z.^Z+c;
W=exp(-abs(Z));     %to establish values between 0 and 1
end

colormap prism(256)       % for creating a map
pcolor(W);   %creates a pseudocolor plot with the values specified in W.
shading flat;
axis('square','equal','off');
```

==

Julia_11.m

```
% Julia_11.m
%
function [W] = julia_11(c,N);
%
x = -2:0.009:2;
y = -2:0.009:2;

k = 1:length(y);
m = 1:length(x);

[x0, y0] = meshgrid(x,y);
p = zeros(size(x0));
%
  Z = x0(k,m) + i*y0(k,m);
  for itr = 1:N;
    Z = Z.*(1-Z) + c;
       W=exp(-abs(Z));
    if abs(Z) > 10,
      p(k,m) = itr; break
    end
  end

colormap prism(256)
pcolor(W);
shading flat;
axis('square','equal','off');
```

==

Julia_12.m

```
%Julia_12.m
%
function [W] = Julia_12(c,N);
%
x=-1.4:0.008:1.0;
y=-1.2:0.008:1.2;
%
k = 1:length(y);
m = 1:length(x);
[x0, y0] = meshgrid(x,y);
n = zeros(size(x0));

  Z = x0(k,m) + i*y0(k,m);
  for itr = 1:N,
   Z=sqrt(Z) - c;
   W=exp(-abs(Z));
    if abs(Z) < 1,
      n(k,m) = itr; break,
    end,
  end,

colormap prism(256)
pcolor(W);
shading flat;
axis('square','equal','off');
```

===

Julia_13.m

```
% a
function [W] = Julia_13(c,N);
%
x=-1.4:0.008:1.0;
y=-1.2:0.008:1.2;

%
for k = 1:length(y);
  for m = 1:length(x);

[x0, y0] = meshgrid(x,y);
n = zeros(size(x0));

   Z = x0(k,m) + i*y0(k,m);
   for itr = 1:N,
     Z=sin(Z)*c;
     W=exp(-abs(Z));
     if abs(Z) < 1,
       n(k,m) = itr; break
     end
   end
mesh(x0,y0,n);
axis('square','equal','off');
az = 0;
el = 90;
view(az, el);
grid off
```

===

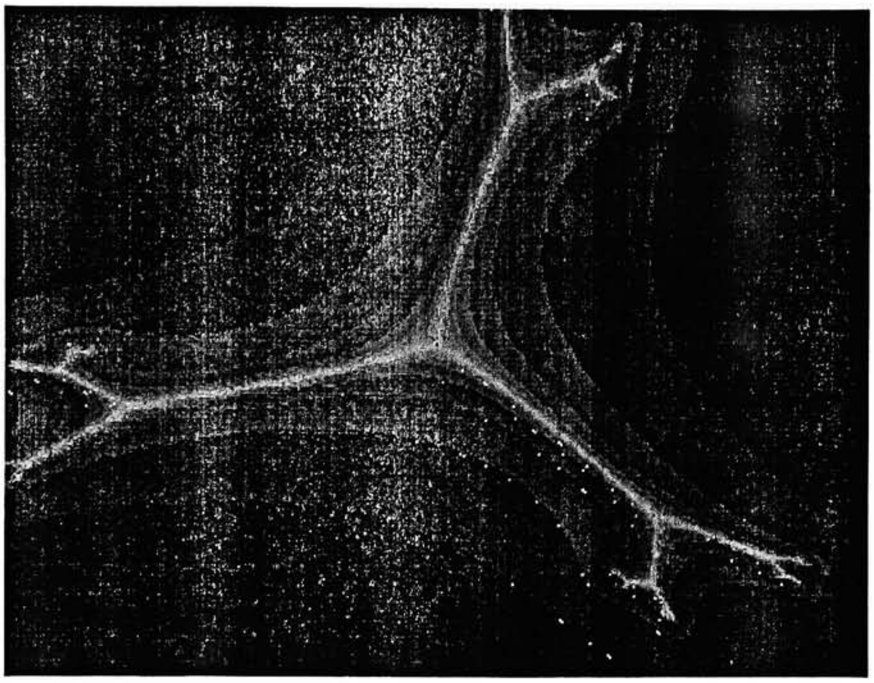
Julia_14(N)

```
function  Julia_14(N)

if nargin ~= 1
        error(['One input argument is required.']);
end

c = 0.46 + i*0.2;
x = -2:0.01:2; y = -2:0.01:2;

k = 1:length(y);
m = 1:length(x);
[x0, y0] = meshgrid(x,y);
p = zeros(size(x0));

  z = x0(k,m) + i*y0(k,m);
  for itr = 1:N;
    z = (c^exp(-1)).*(z.^2) + c;
   if abs(z) > 10,
    p(k,m) = itr; break,
   end,
  end,
mesh(x0,y0,p);
axis('square','equal','off');
az = 0;
el = 90;
view(az, el)
```

==

Julia_15(N)

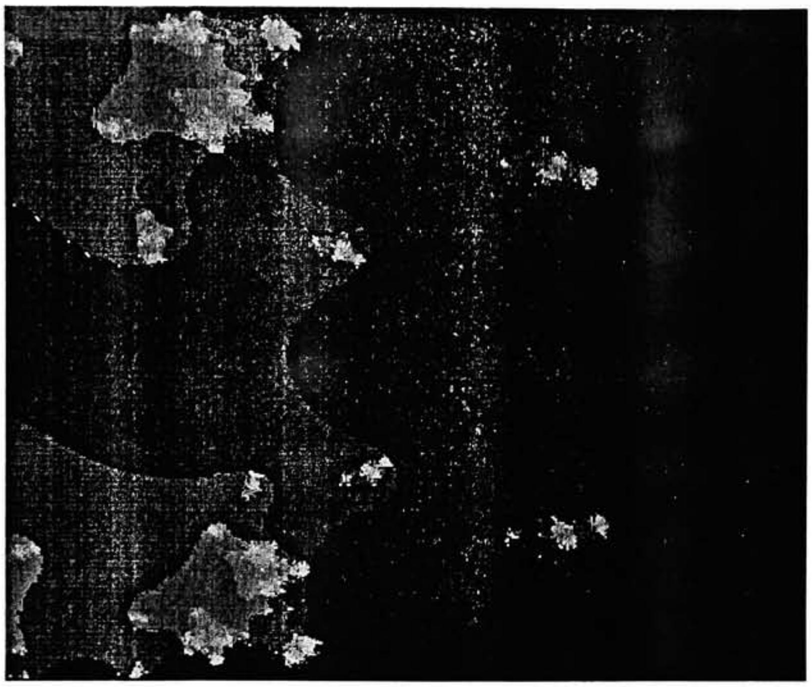
```
%
function  Julia_15(N)

Like Julia_14 with:

C = 1 + i*1.0287137682187 25;
 Z = Z.^2 + C.*conj(Z);
```



==

Julia_16.m

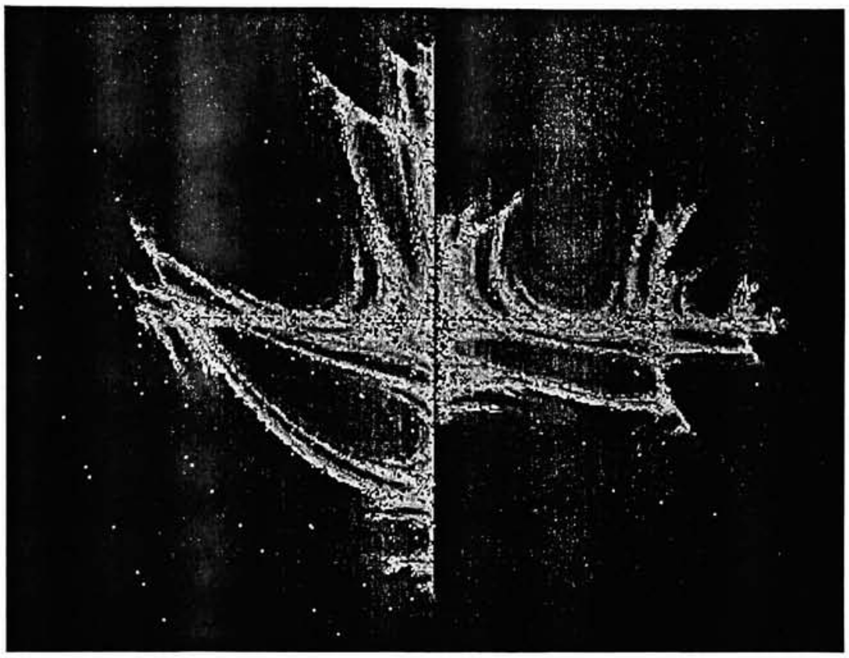
```
% Julia with a function and exponential
function  Julia_16(N)

Like Julia_14 with:

C = 0.46 + i*0.2;
 Z = sin(Z) + exp(Z) + C;
```



==

Julia_17.m

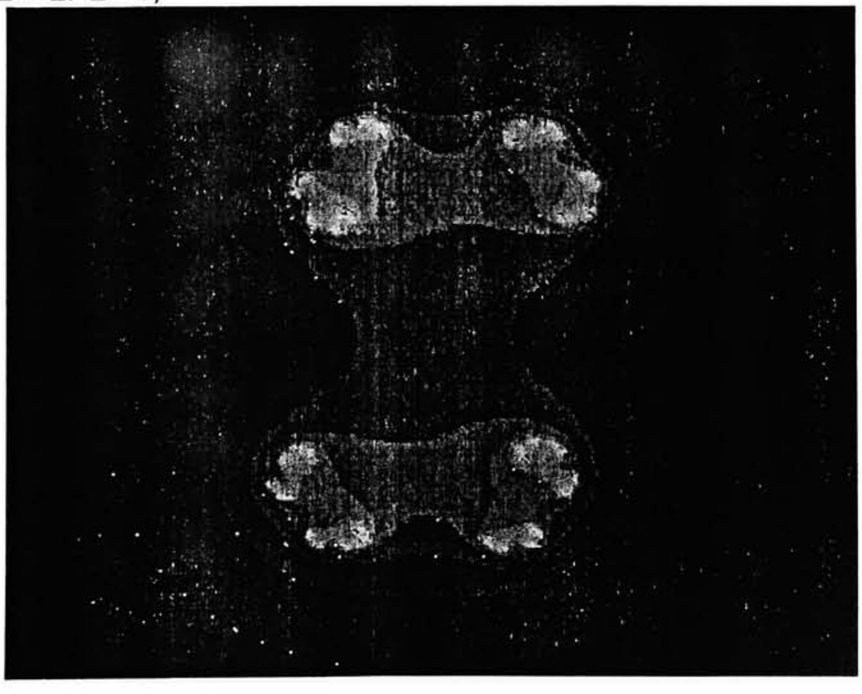
```
% Julia with a function and exponential
function  Julia_17(N)

Like Julia_14 with:

C = 0.46 + i*0.2;
if real(Z) > 0
   Z = (real(Z)^2 - imag(Z)^2 - 1)+ i * (2*real(Z) * imag(Z));
else
  Z = (real(Z).^2 - imag(Z)).^2 - 1 + real(c).* real(Z)+ i * (2*real(Z) .* imag(Z)) +
imag(c).* real(Z);
      end
```



==

Julia_18.m

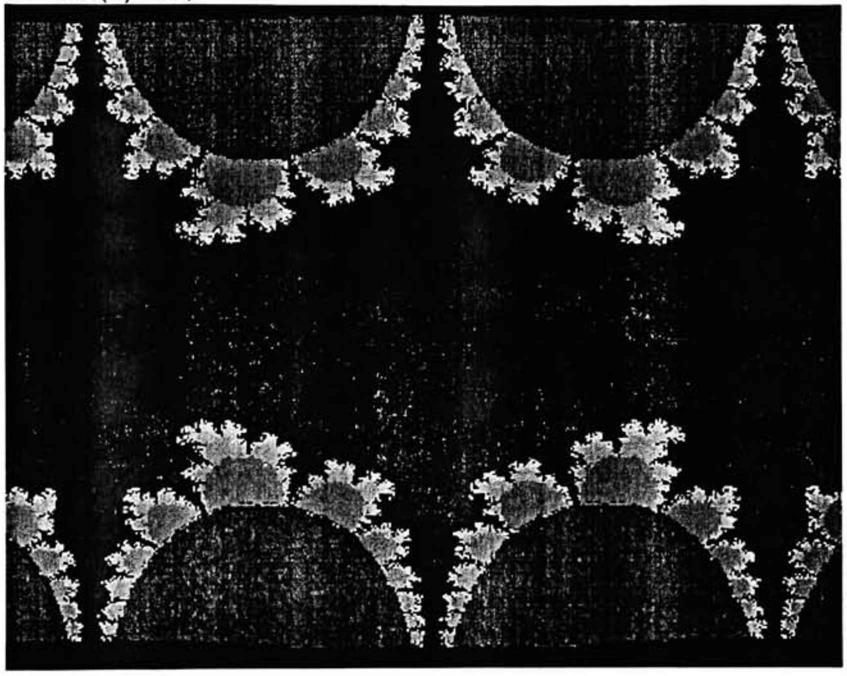
```
% Julia with a function and exponential
function  Julia_18(N)
Like Julia_14 with:

c = -.74543+ i* .11301;
z = z.^2 - c;
```



==

Julia_19.m

```
% Julia with a function and exponential
function  Julia_19(N)

Like Julia_14 with:

c = i*0.1;
z = sin(z).^2 ;
```



==

Mandel_1.m

```
% Mandel_1.m
% This creates a mandelbrot set.
function [W]=Mandel_1(N);

N=20;
r=10;
m=400;
tx=0.25;
ty=0;
```

```
l=0.3;
x=linspace(tx-l,tx+l,m);
y=linspace(ty-l,ty+l,m);
[X,Y]=meshgrid(x,y);
Z=zeros(m);
C=X+i*Y;
for k=1:N;
Z = (C.^exp(-1)).*(Z.^2) + C;
W=exp(-abs(Z));
end
colormap jet(256);
pcolor(W);
shading flat;
axis('square','equal','off')
```

==

Mandel_2.m

```
%Mandel_2
% This creates a mandelbrot set.

function [W]=Mandel_2(N);

x = -2:0.01:2; y = -2:0.01:2;
[x0, y0] = meshgrid(x,y);
n = zeros(size(x0));
k = 1:length(y);
m = 1:length(x);
c = x0(k,m) + i*y0(k,m);
   z=0;
  for itr = 1:N;
     z=z.*7+c;
         W=exp(-abs(z));
     if abs(z)>8
n(k,m) = itr; break
     end
   end


colormap winter(256);
pcolor(W);
shading flat;
axis('square','equal','off')
```
==

Mandel_3.m

```
% Mandel_3.m
% This creates a mandelbrot set.
function [W]=Mandel_3(N);

m=400;
cx=-.2;
cy=0;
l=1.5;
x=linspace(cx-l,cx+l,m);
y=linspace(cy-l,cy+l,m);
[X,Y]=meshgrid(x,y);
Z=zeros(m);
C=X+i*Y;
%
for k=1:N;
Z=Z.^50+C;
W=exp(-abs(Z));
end
%
colormap copper(256);
pcolor(W);
shading flat;
axis('square','equal','off');
```

==

Mandel_4.m

```
% Mandel_4.m
% This creates a mandelbrot set.
function [W]=Mandel_4(N);

x = -2:0.01:2; y = -2:0.01:2;
[x0, y0] = meshgrid(x,y);
n = zeros(size(x0));
k = 1:length(y);
m = 1:length(x);
  c = x0(k,m) + i*y0(k,m);
  z=0;
  for itr = 1:N,
    z = z*z + c;
    W=exp(-abs(z));
    if abs(z) > 10,
      n(k,m) = itr; break
    end
  end
colormap winter(256);
pcolor(W);
shading flat;
axis('square','equal','off');
```

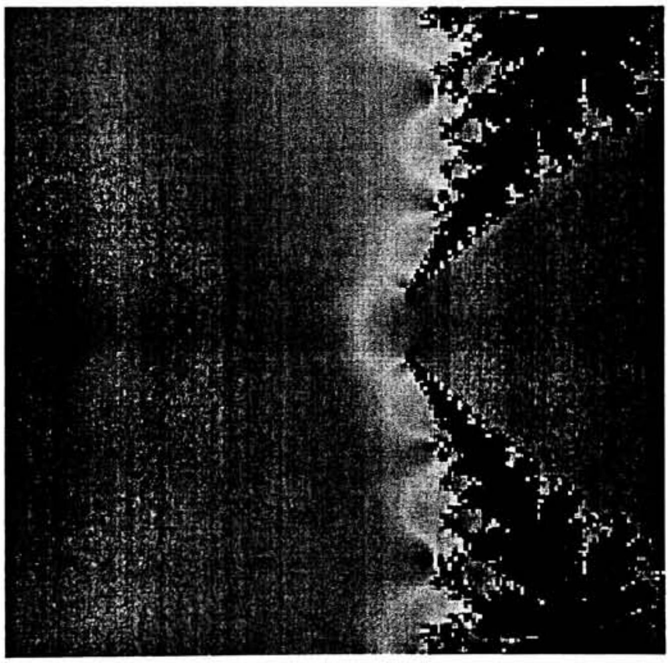
===

Mandel_5.m

```
% Mandel_5.m
% This creates a mandelbrot set.
```

Like Mandel_1 with:

```
Z = C+sin(Z).^2;
```



===

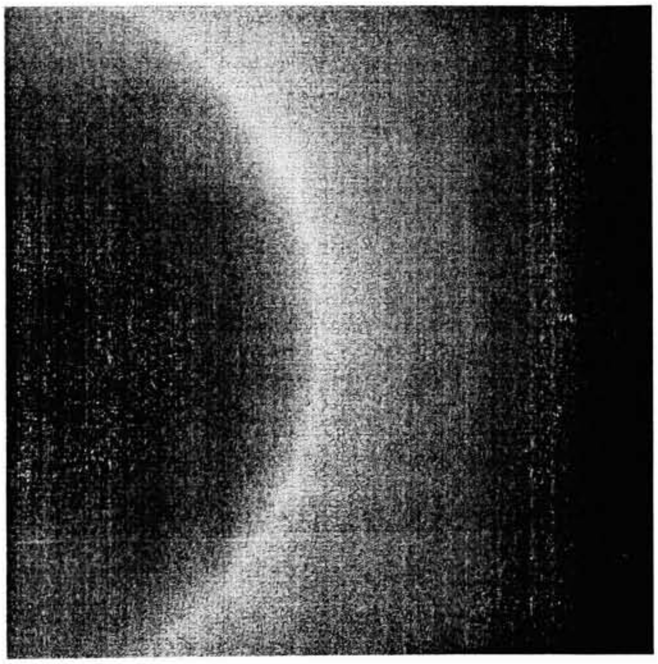
Mandel_6.m

```
% Mandel_6.m
```

% This creates a mandelbrot set.

Like Mandel_1 with:

```
for k=1:N;
if real(z) > 0
    z = (z-1) .* c;
 else
      z = (z+1) .* c;
end
W=exp(-abs(Z));
end
```



==

Newtmthd.m

```
% uses Newton method

p=20;
m=300;
cx=0;
cy=0;
l=10;
x=linspace(cx-l,cx+l,m);
y=linspace(cy-l,cy+l,m);
[X,Y]=meshgrid(x,y);
Z=X +i*Y;
c=-0.5-i*0.8660254;
for k=1:p;
Z=2/3*Z + 1/3*1./( Z.^2);
end
%
W=abs(Z-c);
A=angle(Z);
colormap prism(256);

mesh(W-A);
shading flat;
axis('square','equal','off');
az = 0;
el = 90;
view(az, el);
```

==

Mosaic2.m

```
% Mosaic2

%this program creates a mosaic pattern. Gives the option to choose a pattern

x=-50:1:50;
y=x;

[X,Y]=meshgrid(x,y);
% select a function that creates a portrait
 while 1    clc;
 v = input(' Z1(1), Z2(2),Z3(3),Z4(4),Z5(5),Z6(6),Z7(7),Z8(8),Z9(9) ? ');
   if v >= 1 & v <= 9;      break
   end
 end

switch v
   case 1
      disp('Z1');
      Z=sin(X.^2+Y.^2);
   case 2
      disp('Z2');
     Z=sin((X + Y ) + sin(3*X) + sin(3*Y));
   case 3
      disp('Z3');
     Z=sin((X.^2 + Y.^2) + sin(3*X)+ sin(3*Y));
   case 4
      disp('Z4');
      Z=sin((X +Y) + (sin(3*X).^2)+ (sin(3*Y).^2));
   case 5
      disp('Z5');
      Z=sin((X +Y) + tan(3*X) + tan(3*Y));
  case 6
      disp('Z6');
     Z=sin((X.^6 + Y.^6) + (sin(3*X).^4)+ (sin(3*Y).^4));
   case 7
      disp('Z7');
      Z=sin((X +Y) + sin(Y).*sin(3*(Y)).*sin(X).*sin(3*(X)));
   case 8
      disp('Z8');
      Z=sin((X +Y) + sin(3*X + 3*Y + sin(2*(X)+sin(2*Y))));
  case 9
      disp('Z9');
      Z=sin((X + Y ) + sin(10*X) + sin(10*Y));
end

pcolor(X,Y,Z);
shading interp
axis('equal','square','off')


=========================================================
```

Tiling.m

```
%

%this program creates a tiling mosaic pattern.

x=-50:0.5:50;
y=x;
[X,Y]=meshgrid(x,y);
g=rand
Z=g.*(sin(pi*X)+sin(pi*Y));
pcolor(X,Y,Z);
shading interp
axis('equal','square','off')
```

==

Sierpinski.m

```
function [xpts,s]=sierpinski(h)
% h is the height of the gasket.
clf
h=input('Input the height \n\n h=');
clf;
mov = avifile('sierpy.avi', 'fps',30,'quality',100)
for y=0:h-1
  for x=0:y
    s= [x+h-.5*y h/2-y];
    xpts = s;
    if bitand(x,(y-x)) == 0 % returns bit wise of x and y-x
      plot(s(1),s(2),'r.');
      axis equal; axis off;
      set(gcf,'Color',[1,1,1]);
      end
     xpts = [xpts s];
      G=getframe(gca);
    mov = addframe(mov,G);
      hold on
   end
  end
set(gca,'FontSize',14);
title('The Sierpiski Gasket');
hold off;
mov = close(mov)
```

==

Tinkerbell.m

```
% Tinkerbell Mapping
  function [X1,Y1] =Tinkerbell(n);

if nargin ~= 1
          error(['One input argument is required.']);
     elseif n <4000
     error(['Bell does not look good -use a ' ...
        'minimum of 4000']);
end
x(1) = 0.1 ;
y(1) = 0.5;
%
for k = 1:n
   x(k+1)= x(k)^2 - y(k)^2 + 0.9*x(k) - 0.6013*y(k);
   y(k+1)= 2*x(k)*y(k) + 2*x(k) + 0.5*y(k);
end

%
plot(x,y,'*')

X1=x;
Y1=i.*y;

Random_IFS
```

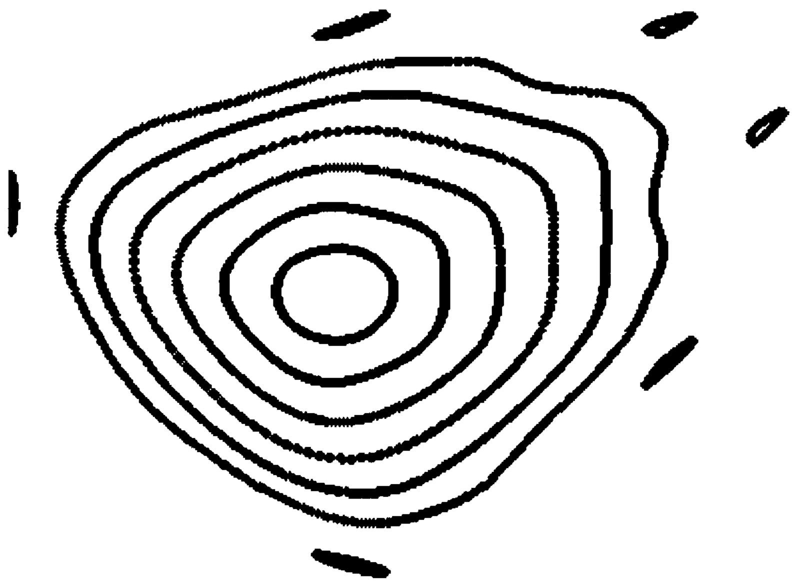
==

Henon1.m

```
% Generalized Henon Mapping
function [X1,Y1] = Henon1(N)
clf
if nargin ~= 1
          error(['One input argument is required.']);
     elseif n <1000
     error(['Bell does not look good -use a ' ...
        'minimum of 2000']);
end
mov = avifile('Henon1.avi','fps',2'quality',100))

for nn = 2:10
a = 1.265;
x(1) = 0.1 + (nn-2)/10;
y(1) = 0.1 + (nn-2)/10;
```

```
%
for k = 1:N
   x(k+1)=x(k)*cos(a) - (y(k)-(x(k))^2)*sin(a);
   y(k+1)=x(k)*sin(a) + (y(k)-(x(k))^2)*cos(a);
end;
%
plot(x(1000:N),y(1000:N),'.')
G=getframe;
mov=addframe(mov,G);
hold on
axis off
end
mov=close(mov)
```

example
N=2000



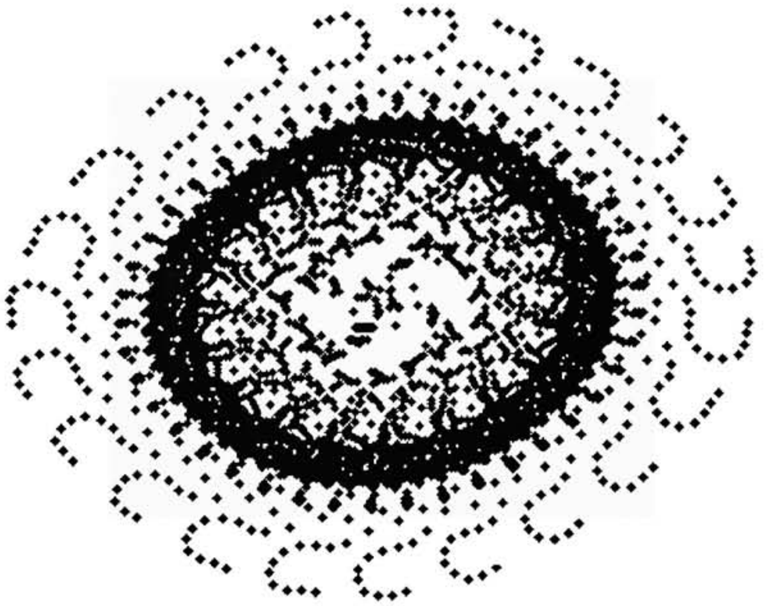
==

Quadattract.m

```
% Program creates a Quad attractor.
function [X1,Y1] = quadattract(N)
```

```
clf
N=input('Input the number of points \n N=');
mov = avifile('quadattract.avi','fps',1,'quality',100))
for nn = 2:10
a = pi;
b=0.3;
c=0.5;
x(1) = 0.1 + (nn-2)/10;
y(1) = 0;
%
for n = 1:N
x(n+1) = y(n) - sign(x(n)).*sqrt(abs(b.*x(n)-c))* atan(sqrt(log(abs(c*x(n)-b))));
    y(n+1) = a - x(n);
end;

%
plot(real(x(1:N)),real(y(1:N)),'.')
G=getframe;
mov=addframe(mov,G);
hold on

end
mov=close(mov)
```

Example N=200

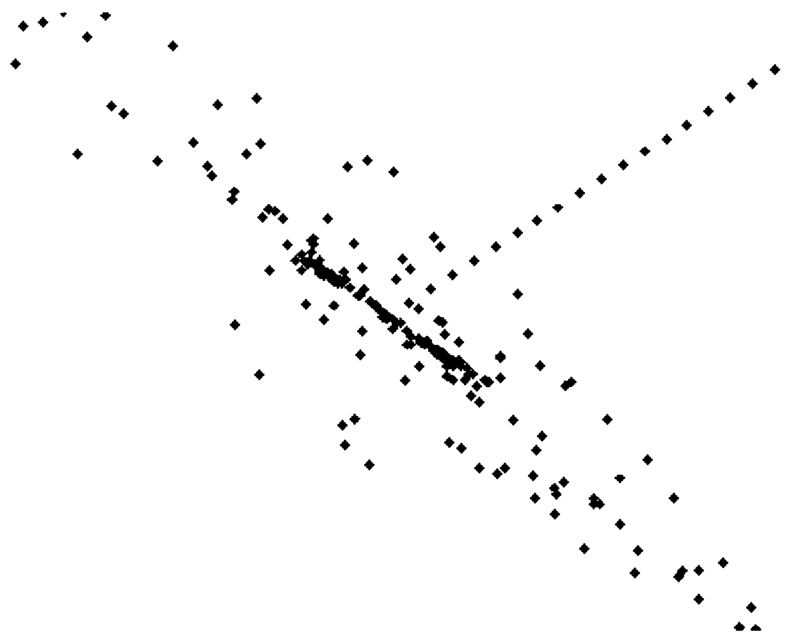


==

Rings.m

```
% Generalized Saturn rings
function [X1,Y1] = rings(N)

%
clf
for nn = 2:20
a = -2.8; b = 2.5; c = 0.7; d = 1.3;
x(1) = 0.1 + (nn-2)/10;
y(1) = 0.1 + (nn-2)/10;
%
for k = 1:N
   x(k+1)= sin (y(k) * b) + c * sin(x(k) * b);
   y(k+1) = sin (x(k) * a) + d * sin(y(k) * a);
end;

%
plot(x(1:N),y(1:N),'.')
hold on

end
```

example
N=500



==

Rossler.m

```
% Rossler attractor

function [X1,Y1,Z1] = Rossler(N);
if nargin ~= 1
         error(['One input argument is required.']);
    elseif N <2000
    error(['not looking very good-use a ' ...
       'minimum of 2000']);
end

dt = .04, a = .2, b = .2, c = 5.7
%
x(1)=0;
y(1)=0;
z(1)=0;
for k = 1:N
  x(k+1) = x(k) - y(k)*dt -  z(k)*dt;
  y(k+1) = y(k) + x(k)*dt + a*y(k)*dt;
  z(k+1) = z(k) + b*dt + x(k)*z(k)*dt - c*z(k)*dt;

end
%
X1 = x;
Y1 = y;
Z1=z;

plot(x, z)
```

==

Lorenz.m

```
% Lorenz Attractor
%

function [X1,Y1,Z1] = lorenz(N);
if nargin ~= 1
         error(['One input argument is required.']);
    elseif N <1000
    error(['not looking very good-use a ' ...
       'minimum of 1000']);
end
dt = .02; a = 5; b = 15; c = 1;
```

```
%
x(1)=1;
y(1)=1;
z(1)=1;
for k = 1:N
   x(k+1) = x(k) + (-a*x(k)*dt) + (a*y(k)*dt);
   y(k+1) = y(k) + ( b*x(k)*dt) - (y(k)*dt) - (z(k)*x(k)*dt);
   z(k+1) = z(k) + (-c*z(k)*dt) + (x(k)*y(k)*dt);

   plot(x, z)
end
%
X1 = x(k);
Y1 = y(k);
Z1=z(k);
```

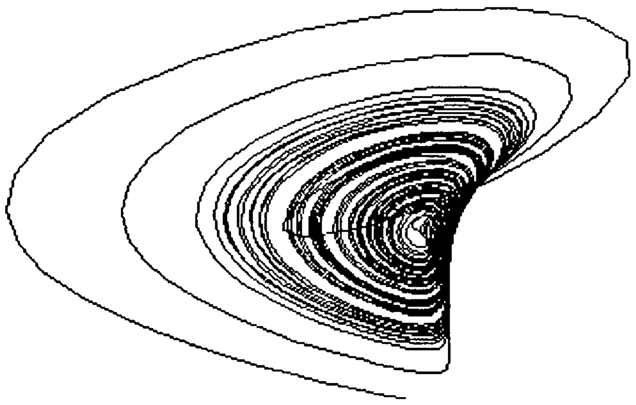
==

Lorenz3.m

```
% Lorenz one lobe Attractor
%

function [X1,Y1,Z1] = lorenz3(N);
if nargin ~= 1
        error(['One input argument is required.']);
    elseif N <1000
    error(['not looking very good-use a ' ...
       'minimum of 1000']);
end


dt = .02; a = 5; b = 15; c = 1;
%
x(1)=1;
y(1)=1;
z(1)=1;
for n = 1:N
norm = sqrt(x(n)^2 + y(n)^2);
  x(n+1) = x(n) + (-a*dt-dt)*x(n) + (a*dt-b*dt)*y(n)...
        + (dt-a*dt)*norm + y(n)*dt*z(n);
      y(n+1) = y(n) + (b*dt-a*dt)*x(n) - (a*dt+dt)*y(n)...
        + (b*dt+a*dt)*norm - x(n)*dt*z(n) - norm*z(n)*dt;
      z(n+1) = z(n) +(y(n)*dt/2) - c*dt*z(n);

   plot3(x,y,z)
```

```
end
%
X1 = x(n);
Y1 = y(n);
Z1=z(n);
```

example
N=5000



==

Lorenz3d.m

```
% Lorenz three lobe Attractor
%

function [X1,Y1,Z1] = lorenz3d(N);
if nargin ~= 1
        error(['One input argument is required.']);
    elseif N <1000
    error(['not looking very good-use a ' ...
       'minimum of 1000']);
end


dt = .02; a = 5; b = 15; c = 1;
%
x(1)=1;
y(1)=1;
```

```
z(1)=1;
for n = 1:N
norm = sqrt(x(n)^2 + y(n)^2);
  x(n+1) = x(n) +(-(a*dt+dt)*x(n) + (a*dt-b*dt+z(n)*dt)*y(n))/3 ...
        + ((dt-a*dt)*(x(n)^2-y(n)^2)...
        + 2*(b*dt+a*dt-z(n)*dt)*x(n)*y(n))/(3*norm);
     y(n+1) = y(n) +((b*dt-a*dt-z(n)*dt)*x(n) - (a*dt+dt)*y(n))/3 ...
        + (2*(a*dt-dt)*x(n)*y(n) ...
        + (b*dt+a*dt-z(n)*dt)*(x(n)^2-y(n)^2))/(3*norm);
     z(n+1) = z(n) +(3*x(n)*dt*x(n)*y(n)-y(n)*dt*y(n)^2)/2 - c*dt*z(n);

     plot3(x,y,z)
end
%
X1 = x(n);
Y1 = y(n);
Z1=z(n);
```

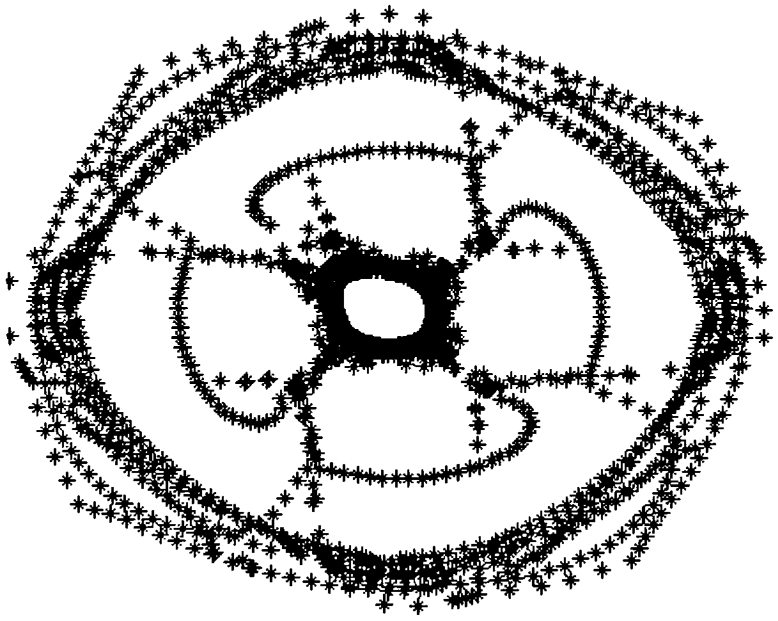
===

Chip.m

```
% Chip.m
% this program produces the chip attractor

function [x,y] =chip(N);
if nargin ~= 1
       error(['One input argument is required.']);
end

fig =figure;
set(fig,'DoubleBuffer','on');
set(gca,'NextPlot','replace','Visible','off')
mov = avifile('chip3.avi','fps',1'quality',100))
for nn = 2:10
x(1) = 0.1 + (nn-2)/10;
y(1) = 0.1 + (nn-2)/10;
a=3; b=4;c=.5;

for n = 1:N
x(n+1) = y(n) - sign(x(n)).* cos(sqrt(log(abs(b.*x(n)-c))))...
                          * atan(sqrt(log(abs(c.*x(n)-b))));
     y(n+1) = a - x(n);
end;
```

```
%
plot(x,y,'*')
G=getframe(gca);
mov=addframe(mov,G);
hold on
end
mov=close(mov)
```

example
N=250



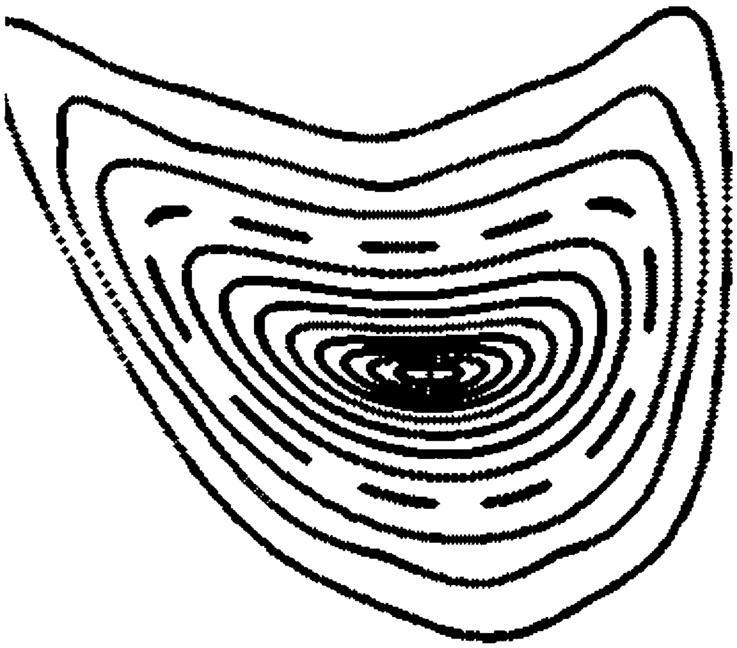
```
==============================================================
```

Kamtorus.m

```
% Generalized Kamtorus Mapping
function [xpts,x,y] = kamtorus(N)

if nargin ~= 1
        error(['One input argument is required.']);
    elseif N <=1000
    error(['not looking very good-use a ' ...
       'minimum of 2000']);
end
mov = avifile('kamtorus.avi','fps',2,'keyframe',5, 'quality',100))
for nn = 2:11
```

```
a = 3.265;
x(1) =  (nn-2)/3;
y(1) =  (nn-2)/3;
%
for k = 1:N
   x(k+1)=x(k)*cos(a) + (x(k).*x(k)-y(k))*sin(a);
   y(k+1)=x(k)*sin(a) - (x(k).*x(k)-y(k))*cos(a);
end;

%
plot(x(1000:N),y(1000:N),'.') %
G=getframe;
mov=addframe(mov,G);
hold on
axis off
end
mov=close(mov)
```

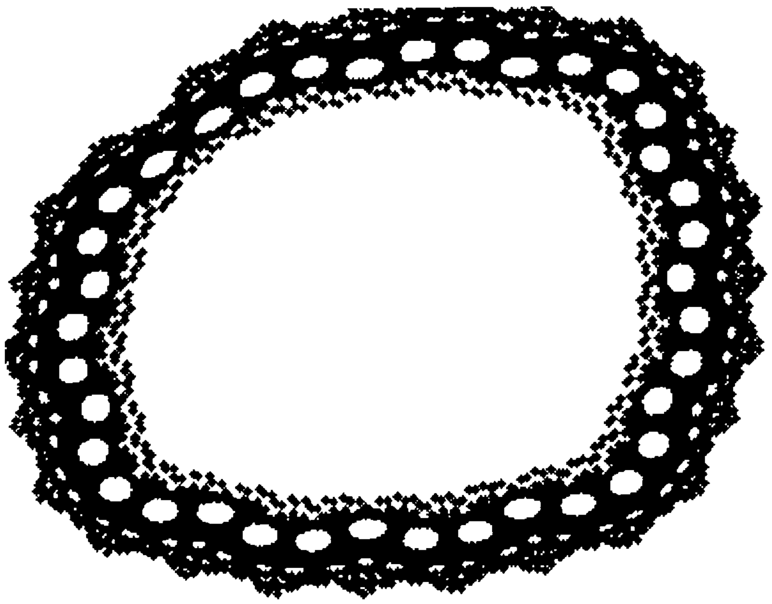
example
N=2000



==

Barryfrac.m

```
function [X1,Y1] = barryfrac(N);
if nargin ~= 1
        error(['One input argument is required.']);
    elseif N <400
    error(['not looking very good-use a ' ...
       'minimum of 500']);
end
mov = avifile('barry.avi','fps',1,'keyframe',5, 'quality',100))
for nn = 2:20 ;
   a=100;b= 5;c=3;
x(1) = 0.1 + (nn-2)/10;
y(1) = 0;
%
for n = 1:N

x(n+1) = y(n) - sqrt(abs(b .* x(n) - c)) * sign(x(n));
y(n+1) = a - x(n);
end;
%
plot(x,y,'.')
G=getframe;
mov=addframe(mov,G);
hold on
end
mov=close(mov)
X1=x;
Y1=y;
```



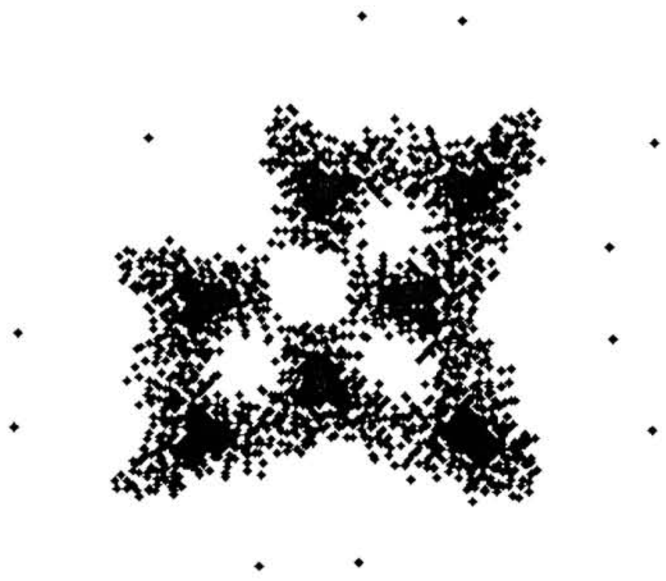
example N=4000

==

Orbitatt.m

```
% Orbitatt.m
% Program creates an attractor for different orbits.
function [X1,Y1] = orbitatt(N)

if nargin ~= 1
        error(['One input argument is required.']);
    elseif N >=1100
    error(['not looking very good-use a ' ...
       'maximum of 1000']);
end
mov = avifile('orbitatt.avi','fps',1, 'quality',100))
for nn = 2:10
a = 1.265;
b=0.3;
c=0.5;
x(1) = 0.1 + (nn-2)/10;
y(1) = 0;
%
for n = 1:N
  x(n+1) = y(n) - sign(x(n)).*sqrt(abs(b.*x(n)-c));
      y(n+1) = a - x(n);
end;

%
plot(x(1:N),y(1:N),'.')
G=getframe;
mov=addframe(mov,G);
hold on

end
mov=close(mov)
X1=x;
Y1=y;
```

example
N=500

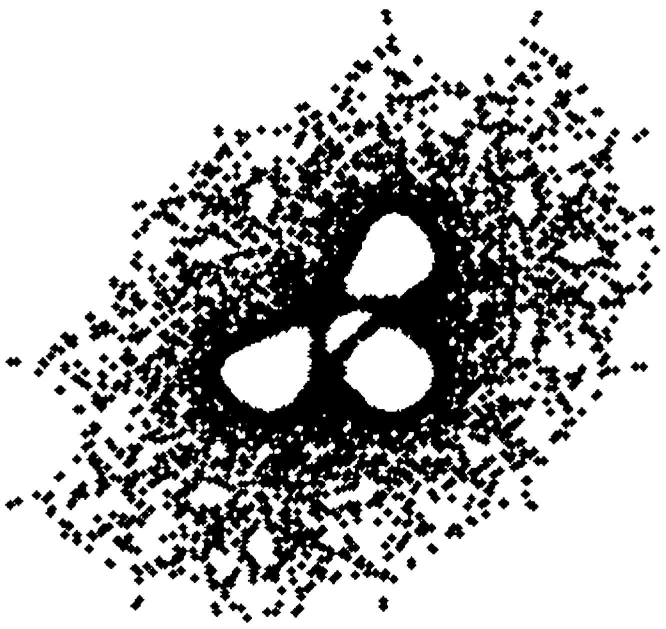


==

Orbitattract.m

```
%Orbitattract.m
% Program creates a Martin attractor.
function [X1,Y1] = orbitattract(N);
clf
N=input('Input the number of points \n N=');

mov = avifile('orbitattract.avi','fps',1, 'quality',100));

for nn = 2:10
a = pi;
b=0.3;
c=0.5;
x(1) = 0.1 + (nn-2)/10;
y(1) = 0;
%
for n = 1:N
x(n+1) = y(n) - sign(x(n)).*(abs(sin(x(n))*cos(b)+ c - x(n)*sin(a+b+c)));
    y(n+1) = a - x(n);
end;
plot(real(x(1:N)),real(y(1:N)),'.')
G=getframe;
mov=addframe(movG);
hold on
end
mov=close(mov)
```

example
N=1000



==

Martin.m

```
% Martin.m
% forms spirals.
function [X1,Y1] = martin(N);

if nargin ~= 1
        error(['One input argument is required.']);
    elseif N <10
    error(['not looking very good-use a ' ...
       'minimum of 20']);
end
mov = avifile('Martin.avi','fps',1, 'quality',100))

for nn = 2:20 ;
   a=-2*pi;

x(1) = 0.1 + (nn-2)/10;
y(1) = 0;
%
for n = 1:N
```

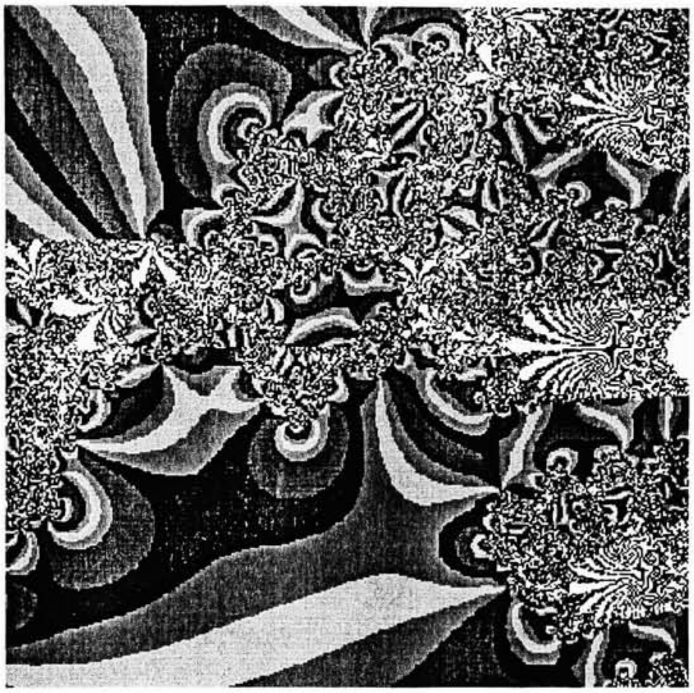
```
x(n+1) = y(n) - sin(x(n)) ;
y(n+1) = a - x(n);

end;

%
plot(x,y,'.')
hold on
G=getframe;
mov=addframe(mov,G);
hold on
axis off
end
```

===

complex_generator.m

```
%

c=.377 -.248*i;
N=30;
p=500;
x=0;
y=0;
g=4;
x=linspace(x-g,x+g,p);
y=linspace(y-g,y+g,p);
[X,Y]=meshgrid(x,y);
Z=X+i*Y;
%
for k=1:N;
Z=Z.^Z+ Z/5 +c;
W=exp(-abs(Z));
if abs(Z)>100 break
end
end
colormap prism(256);
pcolor(W);
shading flat;
axis('square','equal','off');
```

example



==

MOVIES

FERNMOVIE.m

```
function fernmovie(n)

% This program creates a fractal fernmovie.
%
A1 = [0 .0; .0 .5];  A2 = [.42 -.42; .42 .42];
A3 = [.42 .42; -.42 .42];   A4 = [0.1 0; 0 .1];
T1 = [0 ; 0];  T2 = [ 0 ; .2];
T3 = [0 ; .2];   T4 = [0; 0.2];
%
P1 =.05; P2 = .4; P3 = .4; P4 = .15;   % Probabilities
%
clf
```

```
%
s=rand(2,1);
h=plot(s(1),s(2),'.');
set(h,'MarkerSize',12);
axis([-0.5 .5 -.1 .5])
axis square
grid off
 hold
%
for j=1:n
   r=rand;
   nframes = 1;
 for k = 1:nframes
   if r <= P1, s=A1*s+T1;
   elseif r <= P1+P2, s=A2*s+T2;
   elseif r <= P1+P2+P3, s=A3*s+T3;
   else s=A4*s;
   end
  set(h,'XData',s(1),'YData',s(2))
    plot(s(1),s(2),'.')
      M(k) = getframe;
 end
end
hold off
movie(M,1);
```

==

HENONMOVIE.m

```
function [X1,Y1] = Henonmovie(N)
clf
if nargin ~= 1
         error(['One input argument is required.']);
    elseif n <100
    error([' does not look good -use a ' ...
       'minimum of 200']);
end
mov = avifile('Henonmovie.avi','fps',2'quality',100))

for nn = 2:10
a = 1.4 ;b=0.3;
x(1) = 0.1 + (nn-2)/10;
y(1) = 0.1 + (nn-2)/10;
%
```

```
for k=1:N
x(k+1) = 1 + y(k)-a*x(k)^2;
   y(k+1) = b*x(k);
end
   h=  plot(x,y,'.')
set(h,'Markersize',12)
G=getframe;
mov=addframe(mov,G);
hold on
axis off
end
mov=close(mov)
```

==

TREEMOVIE2

```
function [xpts,s] =treemovie2(n)

% This program creates a movie of a fractal tree with
%different values of the affine transformation.
%
A1 = [0 0;0 0.5];  A2 = [0.42 -0.42;0.42 0.42];
A3 = [0.42 0.42; -0.42 0.42];   A4 = [0.1 0;0 0.1];
T1 = [0 ; 0];  T2 = [ 0 ; 0.2];
T3 = [0 ; 0.2];   T4 = [0; 0.2];
%
P1 =.05; P2 = .4; P3 = .4; P4 = .15;   % Probabilities
%
clf
%
s=rand(2,1); xpts = s;
h=plot(s(1),s(2),'.');
set(h,'MarkerSize',6);
axis square
grid off
 hold
%

for j=1:n
   r=rand;
   nframes = 1;
 for k = 1:nframes
   if r <= P1, s=A1*s+T1;
   elseif r <= P1+P2, s=A2*s+T2;
   elseif r <= P1+P2+P3, s=A3*s+T3;
   else s=A4*s;
   end
end
     h= plot(s(1),s(2),'.');
     xpts = [xpts s];
       set(h,'XData',s(1),'YData',s(2));
      M(k) = getframe;
 end
hold off
movie(M,1)
```

==

Juliamovie.m

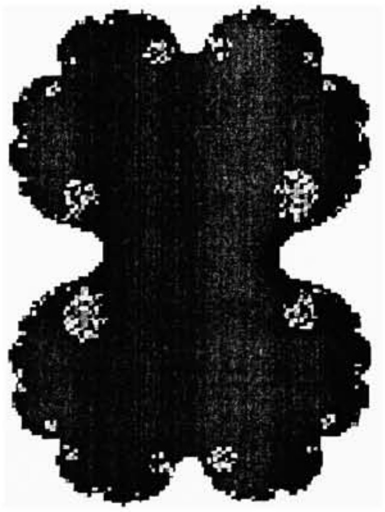
```
%A connected set and movie
c=-0.1+0.8*i;
rho=2.1;
t=linspace(0,2*pi,50);
X=rho*(cos(t)+i*sin(t));

plot(real(X),imag(X),'k.');
hold on
for k1=1:2
   Z1=((-1)^k1)*sqrt(X-c);
   plot(real(Z1),imag(Z1),'b.');
   for k2=1:2
      Z2=((-1)^k2)*sqrt(Z1-c);
      plot(real(Z2),imag(Z2),'g.');
      for k3=1:2
         Z3=((-1)^k3)*sqrt(Z2-c);
         plot(real(Z3),imag(Z3),'y.');
         for k4=1:2
            Z4=((-1)^k4)*sqrt(Z3-c);
            plot(real(Z4),imag(Z4),'m.');
            for k5=1:2
               Z5=((-1)^k5)*sqrt(Z4-c);
               plot(real(Z5),imag(Z5),'k.');
               for k6=1:2
                  Z6=((-1)^k6)*sqrt(Z5-c);
                  plot(real(Z6),imag(Z6),'b.');
                  for k7=1:2
                    Z7=((-1)^k7)*sqrt(Z6-c);
                     plot(real(Z7),imag(Z7),'g.');
```

```
              for k8=1:2
                 Z8=((-1)^k8)*sqrt(Z7-c);
                 plot(real(Z8),imag(Z8),'r.');
                 for k9=1:2
                    Z9=((-1)^k9)*sqrt(Z8-c);
                    plot(real(Z9),imag(Z9),'m.');
                       F=getframe;
                 end
             end
           end
        end
     end
   end
  end
 end
end
plot(real(c),imag(c),'r*');
hold off
movie(F,1)
```

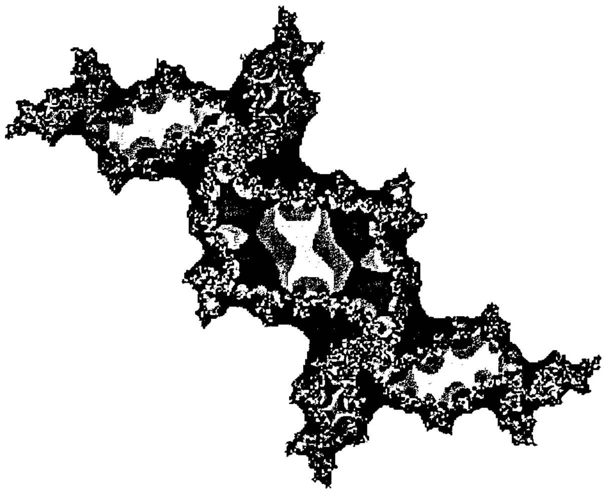
==

Other Julia sets

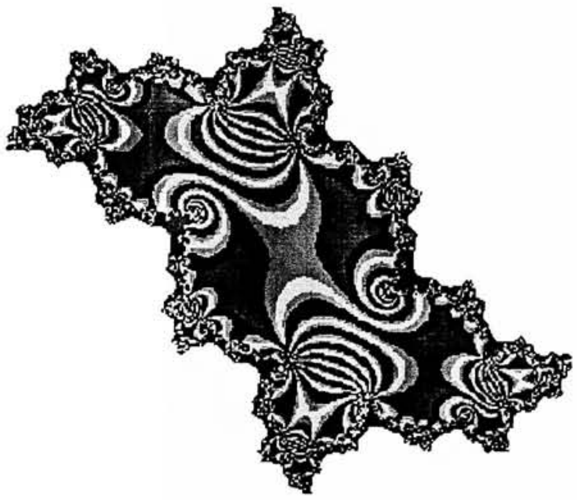
Julia set with c=0.32+0.0043*i;



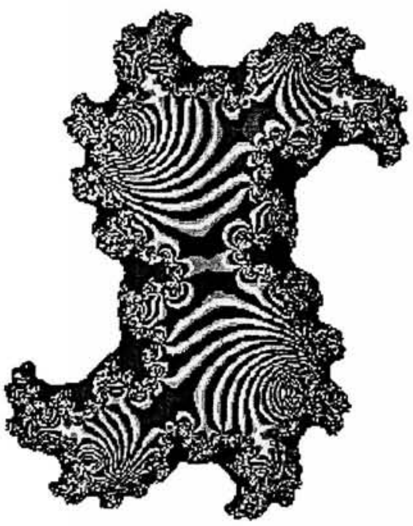
Julia set with c=0.32+0.0043*i;



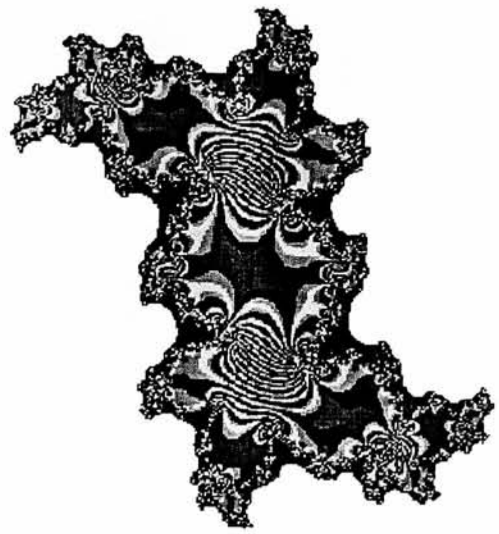
Julia set with c= -0.1194+0.6289 *i;



Julia set with c= 0.377 - 0.248*i;

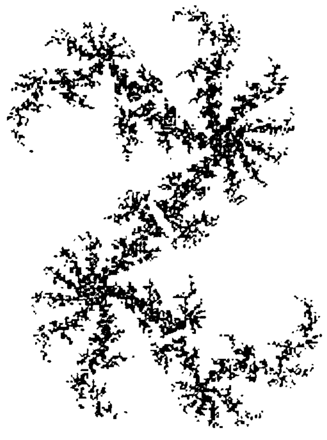


Julia set with c= 0.25 + 0.52*i;



Julia_set

```
Z = Z^2 +sin(Z) +c;
c= 0.1 +0.1*i;
```



Below is the same fractal about the centre ‘zooming in’ (x 4)

Appendix 2

CHART 2
General Midi Instruments List General MIDI program changes

Piano	Chromatic Percussion	Organ
000 Acoustic Grand Piano	008 Celesra	016 Drawbar Organ
001 Bright Acoustic Piano	009 Glockenspiel	017 Percussive Organ
002 Electric grand Piano	010 Music Box	018 Rock Organ
003 Honky Tonk Piano	011 Vibraphone	019 Church Organ
004 Electric Piano 1	012 Marimba	020 Reed Organ
005 Electric Piano 2	013 Xylophone	021 Accordion
006 Harpsichord	014 Tubular bells	022 Harmonica
007 Clavinet	015 Dulcimer	023 Tango Accordion
Guitar	Bass	Strings/Orchestra
024 Nylon Acoustic Guitar	032 Acoustic Bass	040 Violin
025 Steel Acoustic Guitar	033 Electric Fingered Bass	041 Viola
026 Jazz Electric Guitar	034 Electric Picked Bass	042 Cello
027 Clean Electric Guitar	035 Fretless Bass	043 Contrabass
028 Muted Electric Guitar	036 Slap Bass 1	044 Tremolo Strings
029 Overdrive Guitar	037 Slap Bass 2	045 Pizzicato Strings
030 Distorted Guitar	038 Syn Bass 1	046 Orchestral Harp
031 Guitar Harmonics	039 Syn Bass 2	047 Timpani
Ensemble	Brass	Reed
048 String Ensemble 1	056 Trumpet	064 Soprano Sax
049 String Ensemble 2 (Slow)	057 Trombone	065 Alto Sax
050 Syn Strings 1	058 Tuba	066 Tenor Sax
051 Syn Strings 2	059 Muted Trumpet	067 Baritone Sax
052 Choir Aahs	060 French Horn	068 Oboe
053 Voice Oohs	061 Brass Section	069 English Horn
054 Syn Choir	062 Syn Brass 1	070 Bassoon
055 Orchestral Hit	062 Syn Brass 2	071 Clarinet
Pipe	Synth Lead	Synth Pad
072 Piccolo	080 Syn Square Wave	088 New Age Syn Pad
073 Flute	081 Syn Sawtooth Wave	089 Warm Syn Pad
074 Recorder	082 Syn Calliope	090 Polysynth Syn Pad
075 Pan Flute	083 Syn Chiff	091 Choir Syn Pad
076 Bottle Blow	084 Syn Charang	092 Bowed Syn Pad
077 Shakuhachi	085 Syn Voice	093 Metal Syn Pad
078 Whistle	086 Syn Fifths Sawtooth Wave	094 Halo Syn Pad
079 Ocarina	087 Syn Brass & Lead	095 Sweep Syn Pad

Synth Effects 096 SFX Rain 097 SFX Soundtrack 098 SFX Crystal 099 SFX Atmosphere 100 SFX Brightness 101 SFX Goblins 102 SFX Echoes 103 SFX Sci–fi	Ethnic 104 Sitar 105 Banjo 106 Shamisen 107 Koto 108 Kalimba 109 Bag Pipe 110 Fiddle 111 Shanai	Percussive 112 Tinkle Bell 113 Agogo 114 Steel Drums 115 Woodblock 116 Taiko Drum 117 Melodic Tom 118 Syn Drum 119 Reverse Cymbal
Sound Effects 120 Guitar Fret Noise 121 Breath Noise 122 Seashore 123 Bird Tweet 125 Telephone Ring 125 Helicopter 126 Applause 127 Gun Shot		

General MIDI Drums note number
(MIDI channel 10 is used for drums)

035 B0 Acoustic Bass Drum	048 C2 High Mid Tom
036 C1 Bass Drum 1	049 C#2 Crash Cymbal 1
037 C#1 Side Stick	050 D2 High Tom
038 D1 Acoustic Snare	051 D#2 Ride Cymbal 1
039 D#1 Hand Clap	052 E2 Chinese Cymbal
040 E1 Electric Snare	053 F2 Ride Bell
041 F1 Low Floor Tom	054 F#2 Tambourine
042 F#1 Closed Hi Hat	055 G2 Splash Cymbal
043 G1 High Floor Tom	056 G#2 Cowbell
044 G#1 Pedal Hi Hat	057 A2 Crash Cymbal 2
045 A1 Low Tom	058 A#2 Vibraslap
046 A#1 Open Hi Hat	059 B2 Ride Cymbal 2
047 B1 Low Mid Tom	

060 C3 High Bongo (CENTER C)	072 C4 Long Whistle
061 C#3 Low Bongo	073 C#4 Short Guiro
062 D3 Mute High Conga	074 D4 Long Guiro
063 D#3 Open High Conga	075 D#4 Claves
064 E3 Low Conga	076 E4 High Wood Block
065 F3 High Timbale	077 F4 Low Wood Block
066 F#3 Low Timbale	078 F#4 Mute Cuica
067 G3 High Agogo	079 G4 Open Cuica
068 G#3 Low Agogo	080 G#4 Mute Triangle
069 A3 Cabasa	081 A4 Open Triangle
070 A#3 Maracas	
071 B3 Short Whistle	

Made by Marco Zanon http://www.marcozanon.com
Based upon a previous work by Danchan and the standard GM list, freely distributable

Appendix 3.0

<u>MIDI TERMS</u>

- Channel messages are those which apply to a specific Channel, and the Channel number is included in the status byte for these messages.
- System messages are not Channel specific, and no Channel number is indicated in their status bytes.
- Channel Voice Messages are used to send musical performance information. The messages in this category are the Note On, Note Off, Key Pressure, Channel Pressure, Pitch Bend Change, Program Change, and the Control Change messages.
- Note ON- When a key is pressed on a MIDI keyboard instrument or MIDI keyboard controller, the keyboard sends a Note On message on the MIDI OUT port.
- Note OFF - The Note Off message also includes data bytes for the key number and for the velocity with which the key was released.
- The Pitch Bend Change message is normally sent from a keyboard instrument in response to changes in position of the pitch bend wheel. The pitch bend information is used to modify the pitch of sounds being played on a given Channel.
- The Program Change message is used to specify the type of instrument that should be used to play sounds on a given Channel.
- Channel Mode messages (MIDI controller numbers 121 through 127) affect the way a synthesizer responds to MIDI data.
- MIDI System Messages are classified as being System Common Messages, System Real Time Messages, or System Exclusive Messages. System Common messages are intended for all receivers in the system.
- The MIDI System Real Time messages are used to synchronize all of the MIDI clock-based equipment within a system, such as sequencers and drum machines.
- System Exclusive messages may be used to send data such as patch parameters or sample data between MIDI devices.

- A MIDI file allows easy editing of the individual musical parts, because each part is usually assigned to its own MIDI channel, and it's easy to separate that part's MIDI data from the other parts' MIDI data, based upon the MIDI channel in each MIDI message.
- A sequencer is a machine that "plays" musical performances. It tells equipment that can make musical sounds (i.e., play pitches, chords, etc.) what musical notes to play, and when to play them. It does this using MIDI messages. One advantage of using this is one can change the playback speed.

LIST OF MATLAB PROGRAMS

8.2 REFERENCES

1. Computers, patterns, chaos and beauty –graphics from an unseen world, Clifford A. Pickover 1990.
2. Fractals, Form, chance and Dimension, Benoit B.Mandelbrot, 1977
3. Chaos, Fractals and dynamics –computer experiments in mathematics, Robert L. Devaney, 1990
4. Matlab for engineers, Adrian Biran, Moshe Breine, 1995.
5. Introduction to Fractals and Chaos, Richard M. Crownover, 1995.
6. Fractals: the pattern of Chaos John Briggs
7. Spectral methods in Matlab, Lloyd N. Trefethen, 2000.
8. Advanced engineering mathematics, Erwin Kreyszig, 1999.
9. Engineering analysis Y.C Pao, 1999
10. Principles of vibration and sound, Thomas D Rossing, Neville H. Fletcher, 1995
11. Fractals-a user's guide for the natural sciences, Harold M Hastings and George Sugihara.
12. Fractals: endlessly repeated geometrical figures, Hans Lauwerier, 1991.
13. A random walk through fractal dimensions, Brian H. Kaye, 1989.
14. "Multi-fractal description of a rugged fine particle profile," Part. Charact., 1(1984) 14-21, B. H Kaye
15. A first course in Dynamics: with a panorama of recent developments, Boris Hasselblatt and Anatole Katok, 2003
16. Introduction to dynamical systems, Michael Brin and Garrett stuck, 2002
17. Dynamical systems: stability ,symbolic dynamics , and chaos ,Clark Robinson,1999
18. An introduction to dynamical systems, D.K Arrowsmith and C.M Plac,1990
19. Dynamical systems: Differential equations, maps and chaotic behavior,
20. An introduction to chaotic dynamical systems, Robert L. Devaney, 1989
21. Chaos and Fractals, Heinz-Otto Peitgen, Hartmut Jürgens, Dietmar Saupe, 1992
22. Analytical Mechanics with introduction to dynamical systems, Josef S Török, 2000
23. Chaos in dynamical systems, Edward Ott, 1993

24. Chaos: An introduction to dynamical systems, Kathleen T.Alligood, Tim D. Sauer, James A. Yorke, 1996
25. Fractal programming in C, Roger T. Stevens, 1989
26. The science of fractal images ,Springer –Verlag, 1988
27. Fractal geometry: mathematical Foundations and applications
28. A course in digital signal processing, Boaz Porat, 1997
29. Digital Signal Processing: Theory, applications and hardware, Richard A. Haddad and Thomas W. Parsons, 1991.
30. Fractal music, Hypercards and more, Martin Gardner, 1991.
31. Internet Audio sourcebook, Lee Purcell and Jordan Hemphill, 1997.
32. Becoming a computer musician, Jeff Bowen, 1994.
33. Sound Processing ,D.Rocchesso
34. Fractals in music: Introductory Mathematics for Musical Analysis, Charles Madden, 1999.
35. "Computer Recreations", A.K Dewdney, Sc AM257, 108-11, July '87.
36. "Designing a pleasant sound mathematically", Mathematics magazine, Vol 74, N0.2, April 2001.
37. Tutorial on Midi and Music Synthesis, Jim Heckroth, 1995.
38. The MIDI manual, David Miles Huber, 1999.
39. Maximum MIDI: Music applications in C++, Paul Messick, 1998.
40. Computer Music in C, Phil Winson and Gene Delisa, 1990.
41. Signal measurement, analysis and testing, Jerry C. Whitaker, 2000
42. "Estimation of Lyapunov exponents using a semi-discrete formulation", Applied Mechanics review, Vol 46, No 11, 1993. , Josef S Török.
43. Fractal Music Composer, version 2, Hugh McDowell
44. Advanced Engineering Mathematics, Erwin Kreyszig, 8TH Ed.

Web sites
W1) www.borg.com
W2) www.kenschutte.com/midi
W3) www.artemis.cs.yale.edu/~als48
W4) www.rose-hulman.edu/Class/ee/doering/ece481/mini-project_5.htm
W5) www.artemis.cs.yale.edu