12-2014

Computational Exploration of Chaotic Dynamics with an Associated Biological System

Akshay Galande
Clemson University, agaland@g.clemson.edu

COMPUTATIONAL EXPLORATION OF CHAOTIC DYNAMICS WITH AN ASSOCIATED BIOLOGICAL SYSTEM

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the
Degree Master of Science
Computer Science

---

by
Akshay Sunil Galande
December 2014

Accepted by:
Dr. Brian C. Dean, Committee Chair
Dr. Eleanor W. Jenkins
Dr. Elena S. Dimitrova

Abstract

Study of microbial populations has always been a topic of interest for researchers. This is because microorganisms have been of instrumental use in the various studies related to population dynamics, artificial biofuels etc. Comparatively short lifespan and availability are two big advantages they have which make them suitable for aforementioned studies. Their population dynamic helps us understand evolution. A lot can be revealed about resource consumption of a system by comparing it to the similar system where bacteria play the role of different factors in the system. Also, study of population dynamics of bacteria can reveal necessary initial conditions for the desired state of microbial population at some reference point in future. This makes it interesting for ecological and evolutionary disciplines.

Chaos is a mathematical concept which characterizes behavior of dynamical systems that are highly sensitive to the initial conditions. Small differences in the initial conditions such as those due to rounding errors of values of initial parameters yield widely diverging outcomes for such dynamical systems. The way biological systems behave in nature, there is a reason to believe that they do indeed follow chaotic regime. Various mathematical models have been proposed to mimic biological systems in nature. We believe that models which follow chaotic regime represent the biological systems in better way and also are more efficient. We propose a new software tool which may help simulate the mathematical model at hand and provide view

of different set of parameters which can keep the system in chaotic state. This may help researchers design better and efficient biological models or use existing models in better way.

Contents

List of Figures

Chapter 1

Introduction

The study of microbial populations has always been a topic of interest for researchers. This is because microorganisms have been of instrumental use in the various studies related to population dynamics, artificial bio-fuels etc. Comparatively small lifespan and availability are the two big advantages they have which make them suitable for use in the aforementioned studies. Their population dynamic helps us understand evolution. A lot can be revealed about resource consumption of a system by comparing it to a similar system where bacteria play the role of different factors in the system. Also, study of population dynamics of bacteria can reveal necessary initial conditions for the desired state of microbial population at some reference point in future. This makes it interesting for ecological and evolutional disciplines.

There have been various attempts to study population dynamics of bacteria under experimental conditions. Experimenters have tried to change the behavior of the system by changing some experimental parameters. Experiments done in [5] and [4] are good examples of that. One important aspect of the population dynamics which has been discussed in these papers is of chaotic state of dynamical systems representing population. One way of studying biological systems is to study its chaotic

dynamics. Various effects of the chaotic state on the biological system are the current subjects of studies in the area.

Chaos is a mathematical concept which characterizes behavior of dynamical systems that are highly sensitive to the initial conditions. Small differences in the initial conditions such as those due to rounding errors of values of initial parameters yield widely diverging outcomes for such dynamical systems. The fact that biological systems do indeed enter chaotic regime states that it is not just a theoretical concept. Chaotic state of the system may have a profound effect on the overall demographics of the system. One of the most popular examples is the chaotic demographic dynamics of laboratory populations of flee beetle *Tribolium castaneum* [3]. It explains chaotic demographic of flee beetle population with the help of experimental results. It shows how small perturbations done at the right time might produce larger desired change in demographic at later time. Computational analysis of chaotic and non-chaotic regions for the given microbial system may enable us to do the same.

The theoretical proof of chaos in the microbial systems was discussed in [5]. The readings in this paper were based on an actual biological experiment. A biological system was nurtured and population readings were taken periodically. Based on the data collected experimentally it was determined that the microbial system used in the experiment follows chaotic regime. We also have a mathematical model which mimics the experiments.These mathematical model goes into a chaotic state for some parameter values. If it can be showed that the actual biological system represented by that model follows a chaotic regime when the parameters are set to that of experimental conditions, it strengthens the claim of accuracy for the model. To do this we developed the tools to do the mathematical simulation of the model. Mathematical simulations are of great importance as they can generate results faster and minimize the need of actual experiment. Provided that the mathematical model mimicking the

biological experiment is accurate, the predictions done by the simulation are highly useful to understand the effects of changes being made to the system before actually making them.

Mathematical simulations have been used to predict the outcome of the biological experiments before. In [4] a study of plant growth promoting rhizobacteria (PGPR) was done to determine the environmental conditions necessary for their survival. Mathematical simulations showed that competition for the limiting resources was the important factor for the survival of PGPR. Finding solution to a biological problem by mathematical simulation minimized the necessity of real time experiments and made the system overall more efficient. Success of this experiment underlines the importance of a system which can generate mathematical simulations and also predict parameter values for a given mathematical system to be in chaotic state. A system which is blind to the input mathematical model and capable of simulating and predicting parameter values for a given dynamical system (so that it remains in chaotic state) will be of great value.

In this paper we describe Becks' sets of equations with different number of variables. Each set mimics a regulated biological system. We experimented with various parameters of these sets of equations and were able to get the resulting dynamical system in and out of chaotic state. We also have come up with a system which can generate mathematical simulations of the given dynamical system and also predict suitable initialization parameters from the set of variable parameters provided to the system. We also used the same system to study chaotic dynamics of the mathematical equations which are based upon the experiments performed in [5]. It uses a genetic algorithm to predict the parameter values necessary for the chaotic state. The genetic algorithm described, evolves the set of parameters such that their values cause the system represented by those sets of equations to be in chaotic state. The genetic

algorithm is explained in later chapters. We believe this feature to be a good addition in the toolset available for an evolutionary biologist.

It is important to note that there are multiple possible initial points starting from which a mathematical system can reach a chaotic state. To visualize the effect of each parameter on the system, it is necessary to be able to see a representative set of all possible starting points. To achieve this we used a Metropolis algorithm which simulates a random walk through all possible points' sample space. All the points selected using this random walk can be visualized on a screen. We ran the Metropolis algorithm on Becks' sets of equations with selected parameter space and plotted them in the parallel co-ordinate format to understand relationships between them. Parallel co-ordinate format is good for representing multidimensional data. It helps researchers to determine effect of a parameter value on the evolution of the mathematical system with respect to other parameter values. Data generated through the Metropolis algorithm implementation were made easier to visualize using this format.

We also found that the combination of the genetic and the Metropolis algorithm generates better results. We seeded the Metropolis algorithm by the results given by the genetic algorithm. This approach made sure that larger number of points visited by the Metropolis algorithm caused underlying dynamical system to be in chaotic state. Larger number of chaotic points also strengthens the conclusions about the relation between different parameter values made from the Metropolis algorithm.

To summarize everything here, this report contains some initial background of theoretical requirements of chaos and methods to determine chaose in the system. In later chapters it contains the description of mathematical equations based on [5], our manual experiments with the equations. Then it discusses the genetic algorithm used to predict initialization values of parameters and Metropolis algorithm to generate

the representative set of all starting points necessary for the mathematical system to follow a chaotic regime.

Chapter 2

Literature Review

Effect of chaotic state on the evolutionary behavior of the system has been a subject of study in the literature. Studies have been done to investigate effective use of chaotic state to change the final condition of the system to get the expected results. Work in [5] studies the dynamics of a defined predator-prey system consisting of a bacterivorous ciliate and two bacterial prey species. It shows that the dynamic behavior of such a two-prey, one-predator system includes chaotic behavior, as well as stable limit cycles and coexistence at equilibrium. While experimental data have been used to estimate chaotic behavior observed in [5], theoretical inspection of underlying mathematical model to check whether system can transform in and out of chaotic state is not present. We analyze underlying system of dynamical equations and check if it can transform in and out of chaos. Our study of various systems will enable us to design a tool which can generate possible set of parameter values for any given dynamical system of equations so that it remains in chaotic state. Also, may grant an ability to transform the given system in and out of chaotic state by changing experimentally controllable parameters.

2.1 Chaos in Biological Systems

Becks et al. [5] is one of the pioneer work in which showed the existence of chaos in the biological system. Even as large ecological systems as a whole do exhibit chaotic nature, the same nature was not theoretically shown in the smaller systems which in reality are a part of larger ecological system. Although in the number of biological experiments smaller ecological systems i.e. predator-prey systems does show population dynamic which appears to be chaotic. Simple predator-prey systems do show periodicity in the population density of each species with chaotic fluctuations in it. Predator-prey interactions were supposed to be the driving force for this chaotic nature. Field of chaos became a point of interest after presence of chaotic nature was shown on the basis of population data of predator-prey system collected experimentally.

2.1.1 Experimental Setup

The experiment involved three species i.e. two prey bacteria and one predator ciliate species. As preys two coexisting species i.e. rod shaped Pedobacter and coccus Brevundimonas were used and as a predator a ciliate Tetrahymena Pyriformis was used. In the previous experiments it was observed that Pedobacter had better fitness than Brevundimonas i.e. when kept in an environment consisting of only these two preys in the absence of any predator, Pedobacter always outperformed Brevundimonas in terms of population. It was also observed that ciliate Tetrahymena Pyriformis had a preference for the better performing prey i.e. Pedobacter.

Cultures of these species were developed inside single stage chemostat system, which were fed continuously with sterile medium. Rate of inflow of medium (dilution rate) could be controlled and varied. Interactions between species were studies under

different dilution rates and reading of population of each species was taken everyday. Depending on the dilution rate species either formed stable coexistence at equilibrium or aperiodic chaotic cycles.

2.1.2 Results

Results according to readings taken show different behavior of bacterial population based on dilution rate. Such as at highest dilution rate Brevundimonas died off early and rest of the species remained in stable coexistence at equilibrium. While for lower dilution rate dynamic behavior of the population was observed. From the experimental data it was shown that system was indeed in chaotic state.

This was an actual experiment which is not possible in every case. Readings in the above experiment were taken over a period of a month and this time may increase if readings are to be taken for different setups. Also, initial concentration of all the species was decided by results obtained from previous experiments. Mathematical simulation of this system might provide us with the numbers that represent population behavior in different cases. Additionally, a mechanism which can enable us to decide how much initial population should be selected can prove very useful in this case. Mathematical simulations may also be used to get a good idea about other systems as well.

2.2 Chaos and Population Control of Insect Outbreaks

Desharnais et al. in [3] have studied population dynamics of flour beetle - Tribolium Castaneum. This study was done in order to understand how flour beetle population

dynamics work and can be controlled to reduce overall population. Idea of nudging the parameters or state variables of system at point where system is most sensitive to change had been applied before but procedure itself was not tested. This approach was used and tested in this paper on flour beetle population.

To test the approach few rules were defined i.e. 'in box' rule and 'out box' rule. 'In box' rule corresponds to the changes made to the flour beetle population when system is highly sensitive to changes and 'out box' rule corresponds to the changes made otherwise. Difference between the effect of changes made by 'in box' and 'out box' rule can prove the idea of nudging the parameters when system is sensitive to changes.

2.2.1 Experimental Setup

For the conduction of the experiment nine lab populations of RR strain of flour beetle Tribolium Castaneum were developed. Experimental parameters i.e. adult mortality rate, adult recruitment rate etc. were set based on observations made in previous experiments on flour beetle population. Each population was maintained in half-pint milk bottle with 20g of standard media and kept in a dark incubator at 32^0C. Census was taken after every two weeks and larval, pupal and adult populations were counted.

The 'in box' and 'out box' rules were applied to selected populations. Adult beetles eat larvae and help controlling the population. To the populations following 'in box' rule, adult beetles were added when larvae population was less than 150. In case of 'out box' rule populations, adult beetles were added when larvae population was more than 150. Biweekly census readings were stored for 132 weeks for all populations. This whole experiment was also simulated using mathematical simulator and the results were compared.

2.2.2 Results

Readings of the all flour beetle populations show that 'in box' rule holds. Beetle larvae population shows significant decline when adult beetles were added using 'in box' rule while adults added using 'out box' rule fail to show their effect on larvae population growth. This rule was also proved by mathematical simulations which were run to mimic the actual experiment.

One important thing to consider other than experimental results is that mathematical simulations also predicted the same result. Thus, it can be concluded that mathematical simulations may be used to predict the outcome of the experiment. Every system is represented by a different mathematical model. Thus, a framework which can incorporate new mathematical models to simulate and provides interface to determine parameter values to keep system in chaotic state is of a great value.

2.3 Lyapunov Exponents

The claim whether the given system is showing chaotic behavior has to be backed by mathematical analysis. Lyapunov exponents of the given system are quite strong indicators to determine that. Lyapunov Exponents are roughly described for a trajectory of a dynamical system as mean exponential rate of divergence of trajectories surrounding it [1]. A dynamical system is said to be in chaotic state if at least one Lyapunov Exponent is positive. A dynamical system in chaotic state represents a bacterial population where each species and resource is active in the environment. Thus, determining whether a given bacterial system is in chaos becomes important. Various techniques have been discussed in the literature to calculate Lyapunov Exponents.

2.4 Theory

By definition Lyapunov Exponent is the mean exponential rate of diversion in the trajectories. Quantitatively, if δZ_0 is the initial separation in the two trajectories and $\delta Z(t)$ is the final separation, Lyapunov Exponent can be calculated as:

$$\lambda = \frac{1}{t} ln \frac{|\delta Z(t)|}{|\delta Z_0|}$$

Thus, calculating Lyapunov Exponent for single dimension dynamical system is easy. In a dynamical system with more than one variable, diversion in each direction can be different. Hence, for each dimension a separate Lyapunov Exponent can be calculated. Thus, as a result for a given system we get a spectrum of Lyapunov Exponents. Thus, if we are considering a continuous dynamical system in n-dimensional phase space, initial state can be viewed as n-sphere of initial conditions. Sphere will become an n-ellipsoid as the system evolves with time. The i^{th} one-dimensional Lyapunov Exponent is defined in terms of length of ellipsoidal principal axis $p_i(t)$.

$$\lambda_i = \lim_{t \to \infty} \frac{1}{t} log_2 \frac{p_i(t)}{p_i(0)}$$

To explain above formula, let us consider a typical dynamical system equation. A continuous dynamical system is defined as

$$\dot{U} = p(U; r)$$

where U is a vector of phase space coordinates of dimension n, and r is vector of control parameters. The evolution of volume in phase space is given by a Jacobian matrix J.

$$J = \left[\frac{\partial p^{(i)}}{\partial U^{(j)}}\right]$$

for all *i, j*, where $p^{(i)}$ is i^{th} function in p and $U^{(j)}$ is j^{th} component of U. Application of the Jacobian introduces stretching and rotation of a phase space. For example, consider Jacobian matrix J and its effect on a circle going through points (1, 0) and (0, 1).

$$J = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \therefore \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ c \end{bmatrix} \tag{2.1}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix} \tag{2.2}$$

As shown in fig 1, circle is transformed into an ellipse and points (1,0) and (0,1) are placed at new coordinates i.e. (a, c) and (b, d). A repeated application of a Jacobian matrix J to points on initial circle will make evolving ellipse more and more elongated. This happens because application of J stretches the phase space in the direction of major principal axis of the ellipse. i.e. in the direction of (a, c). Given enough number of iterations, the length of major principal axis of ellipse would become so large that lengths of other principal axes would be negligible.

This insight is very useful in the decision of the selection of the method to calculate Lyapunov Exponents. One of the tests to verify if the given dynamical system is in a chaotic state is to check if the Maximal Lyapunov Exponent (MLE) of the system is positive. As this is the chief reason for us to calculate Lyapunov Exponents, we will concentrate on calculating MLE of the system instead of the whole Lyapunov Spectrum.

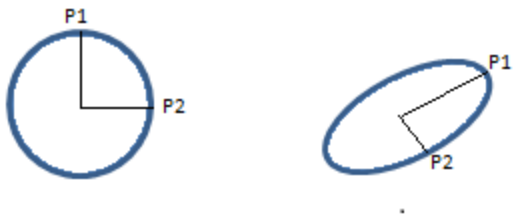

Figure 2.1: Effect of Jacobian

2.5 Calculating method

As discussed in the previous section after enough number of iterations of Jacobian J multiplication with U, the overall change brought to the initial value of U is mainly in a single direction i.e. the direction of the major principal axis of the ellipse. Also, changes in other directions in a phase space are negligible as compared to the first. So if a reference trajectory experiences some random perturbation, the difference between the final state of reference and perturbed trajectory is mainly comprised of difference in a single direction i.e. the direction in which application of J stretches the phase space which is same as the direction of major principal axis. Hence, MLE can be calculated using formula:

$$\lambda_{max} = ln\frac{||\gamma(t)||}{\delta t}.$$

where, γ is the difference between reference and perturbed trajectories. This can be achieved using following algorithm:

```
1. Introduce a small perturbation in the initial
   state U(0) i.e. U‘(0).
```

2. Measure the magnitude of the difference
   (d0 = ||U(0) - U‘(0)||).
3. Evolve both trajectories.
4. After short time interval measure the magnitude
   of difference between both trajectories (d1).
5. Calculate Lyapunov Exponent using formula:

$$\lambda \;=\; ln\ \frac{d1}{d0}$$

6. Again set perturbation trajectory at distance d0
   from reference trajectory.
7. Repeat steps 1-6 for n times.
8. Calculate arithmetic mean of all Lyapunov Exponents
   calculated.

To verify the algorithm, it was tested against the known dynamical systems for values of MLE. Both discrete-time and continuous-time dynamical system were used for the verification.

2.5.1 Henon system

Henon map is a discrete-time dynamical system with two variables. This system has been widely studied for its chaotic behavior. It is described as follows:

$$X_{n+1} = 1 - aX^2 + Y_n$$

$$Y_{n+1} = bX_n$$

As can be seen from equations values of X_{n+1} and Y_{n+1} depend on X_n, Y_n and two parameters a and b. Calculation of MLE in this case is done by introducing perturbation at any step n and taking ratio of difference in values at step $n + 1$.

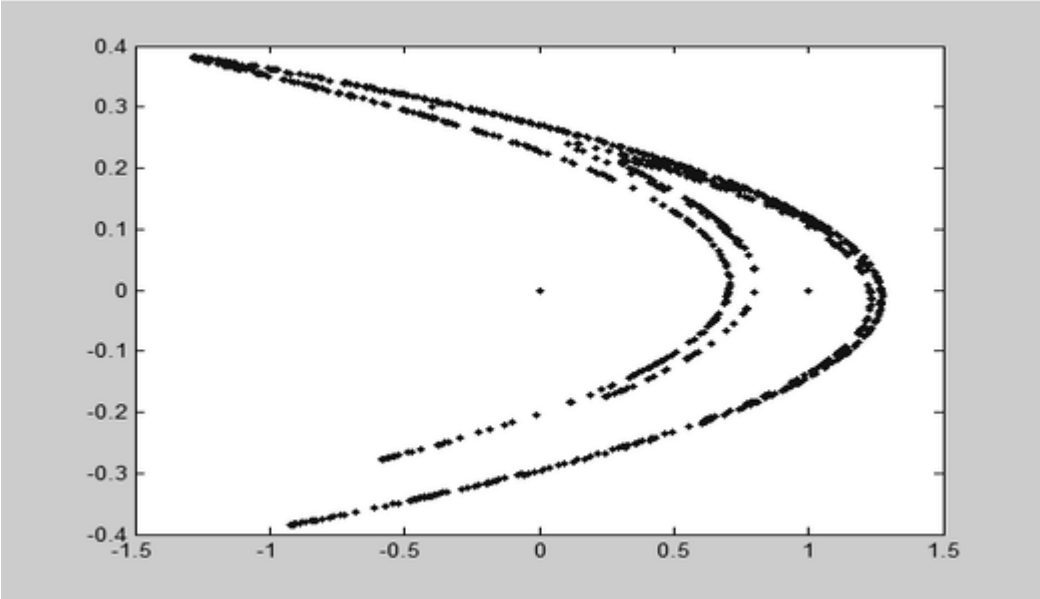

Figure 2.2: Henon Map

2.5.2 Lorenz System

Lorenz dynamical system is continuous-time dynamical system with three variables. The Lorenz model is also widely studied model as it represents atmospheric convection.

$$\dot{X} = \sigma(Y - X)$$

$$\dot{Y} = X(\rho - Z) - Y$$

$$\dot{Z} = XY - \beta Z$$

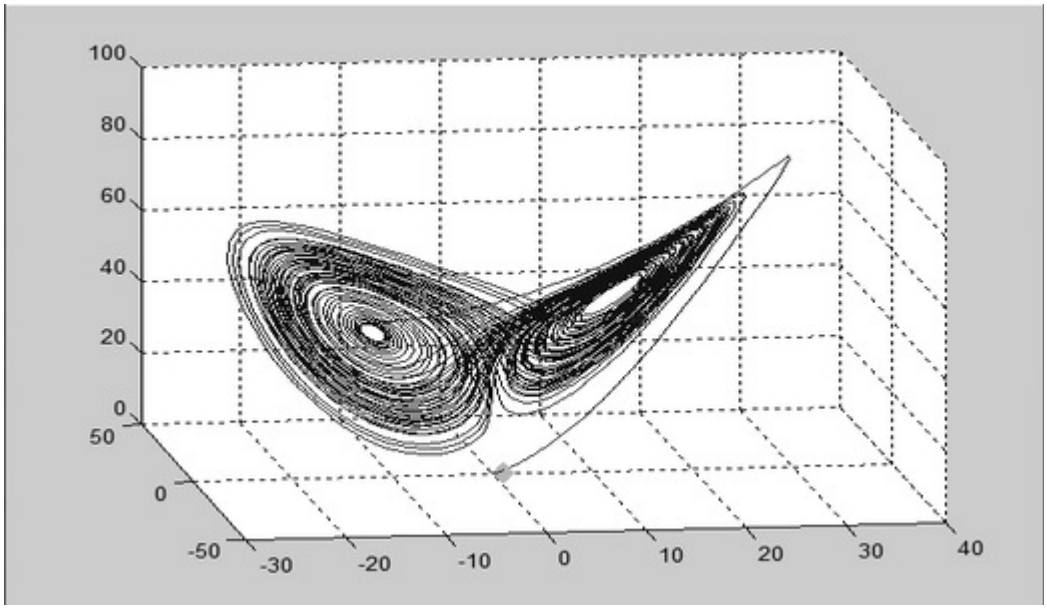

Figure 2.3: Lorenz Attractor

As this system is continuous, calculation of MLE can proceed as described in the algorithm mentioned above. Value of MLE depends on the parameters ρ, σ, β and initial state of the system.

System	Known Value of MLE	Calculated Value of MLE
Henon	0.42	0.419
Lorenz	1.50	1.50
Rossler	0.097	0.080

Table 2.1: Known and calculated Maximum Lyapunov Exponent values

Above readings show that algorithm calculates the approximate value of MLE for given system. For the decision about chaos in the system only sign of Lyapunov Exponent is sufficient. Hence, this algorithm does satisfy requirements of the determination of chaos in the system.

Chapter 3

Mathematical models

3.1 Selection of model

After having decided on the method to calculate Lyapunov Exponent, next important task is to select appropriate mathematical model which should be subjected to test. There are multiple mathematical models available that simulate the microbial population evolution. The experiment done in [5] is better simulated by the dynamical system of equations similar to one described in [2]. Hence, it makes more sense to design a system which is similar to this system and mathematically verify if it is possible to transform it in and out of chaotic state. Moreover, the microbial system described in [5] has been proved to be in a chaotic state based on the population data collected. If our mathematical model also shows the similar behavior and if it also can be shown to be in chaotic state, it may strengthen the claim that our equations represent the system used in the experiment accurately.

3.2 Becks Mathematical Model for Microbial Systems

As mentioned above, this model is very much similar to the one described in [2]. It has been designed by taking into consideration the facts we know about experimental conditions and characteristics of different species used. Every specie used in the experiment grows in number by certain rate. This parameter is called growth rate for particular specie and represented by μ.

As discussed in the previous chapter, to calculate Lyapunov Exponent we introduce a small perturbation. We evolve two different trajectories i.e. one with and one without perturbation for small number of timesteps. We calculate the diversion and in turn Lyapunov Exponent and set perturbation back to its original value.

The model discussed in [2] can be designed for different number of species. The larger the number of species, the higher is the complexity of the model. We started with a model of two variables i.e. one specie and one nutrient and then continued to add one specie at a time till we had model of four variables.

3.3 Becks equations - 2 variables

This model consists of one nutrient and one bacterial specie which lives off the nutrient. This is the simplest model for the bacterial population simulation. It is described as follows:

$$\frac{dx}{dt} = x(\frac{\mu_{nr} * y}{y + k_{nr}} - d_r) - D * x$$

$$\frac{dy}{dt} = D * N0 - \frac{x\mu_{nr}m_r y}{y_{nr}(y + k_{nr})}$$

where x is a bacterial population, y is nutrient content, μ_{nr} is growth rate of bacteria, k_{nr} is half-saturation constant, dr is death rate, N0 is initial nutrient content and D is a dilution rate for given chemostat experiment. As given system with two variables is a continuous dynamical system in Euclidean plan, according to Poincare-Bendixson theorem such a system cannot exhibit chaotic behavior. As a result we did not find chaos in the two variable system.

3.4 Becks equations - 3 variables

This model consists of one nutrient and two bacteria species. Single selected nutrient is essential for the growth of both bacteria and its flow is controlled. Both bacteria feed on the single nutrient and compete for it. The equations for this model are as follows:

$$\frac{dx}{dt} = x(\frac{\mu_{nr} * z}{z + k_{nr}} - d_r) - Dx$$

$$\frac{dy}{dt} = y(\frac{\mu_{nc} * z}{z + k_{nc}} - d_c) - Dy$$

$$\frac{dz}{dt} = D * N0 - \frac{x * \mu_{nr}}{y_{nr} * z(z + k_{nr})} - \frac{y * \mu_{nc}}{y_{nc} * z(z + k_{nc})} - Dz$$

where x,y is bacteria population, z is nutrient content, μ_{nr} and μ_{nc} are the growth rates and d_r and d_c are the death rates of bacteria species.

After experimenting with the dynamical system with different parameter values and initial conditions it was found that system always stays in chaotic state. Change in the growth and death rates of bacteria, dilution rates does not transform the system out of chaotic state. One of the possible reasons might be because this is

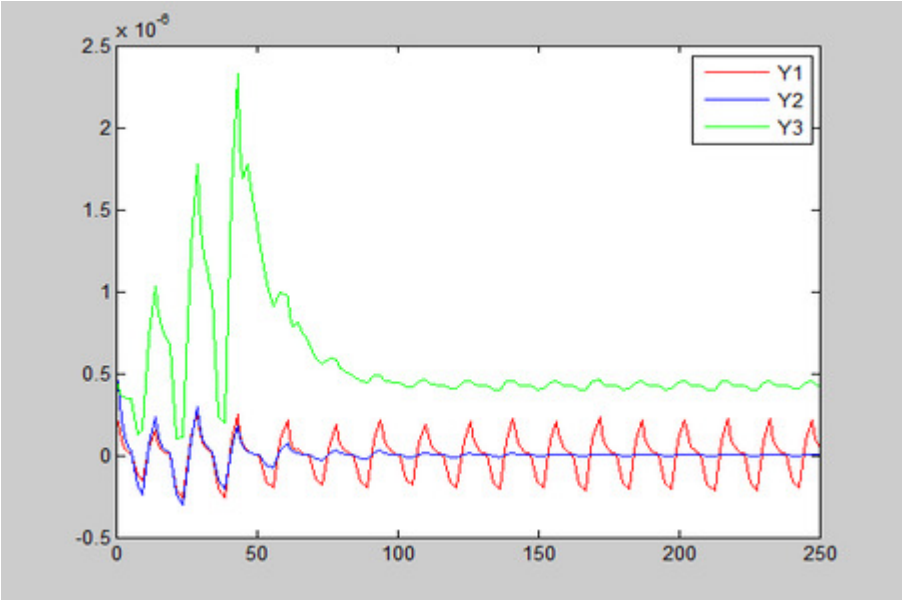

Figure 3.1: Chaotic behavior of 3 variable system

a purely mathematical system, variables do hold negative values which does not necessarily have any biological meaning but they exist as a part of dynamical system. This prevents the system from achieving any equilibrium it possibly could attain by complete extinction of any specie. Figure 3.1 shows an example where system is in chaotic state for given set of parameter values.

3.5 Becks equations - 4 variables

This model consists of one nutrient, two prey bacterial and one predator bacteria which has preference for one of the preys. Nutrient flow is controlled. Prey bacteria feed on nutrient and the predator feeds on both preys. This model is described as follows:

$$\frac{dx}{dt} = x(\frac{\mu_{nr} * w}{w + k_{nr}} - d_r) - \frac{\mu_{pr} * m_p xz}{y_{pr} m_r * (\frac{k_{pr}}{m_r} + x)} - Dx$$

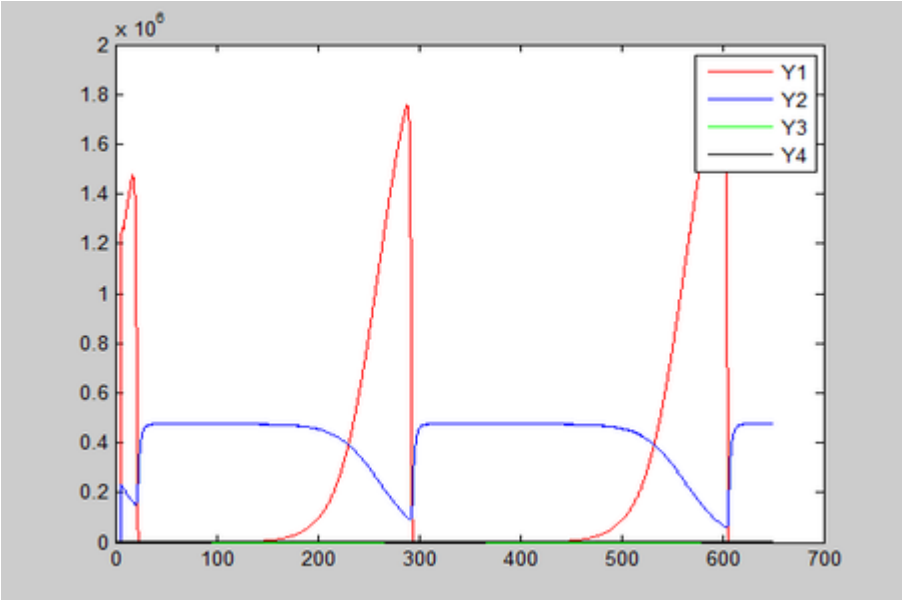

Figure 3.2: Chaotic behavior of 4 variable system

$$\frac{dy}{dt} = y(\frac{\mu_{nc}w}{w + k_{nc}} - d_c) - \frac{\mu_{pc} * m_p yz}{y_{pc}m_c * (\frac{k_{pc}}{m_c} + y)} - Dy$$

$$\frac{dz}{dt} = z(\frac{\mu_{pr}x}{(\frac{k_{pr}}{m_r} + x)} + \frac{\mu_{pc} * y}{(\frac{k_{pc}}{m_c} + y)} - d_p) - Dz$$

$$\frac{dw}{dt} = D * N0 - \frac{x\mu_{nr} * m_r w}{y_{nr}(k_{nr} + w)} - \frac{y\mu_{nc}m_c w}{y_{nc}(k_{nc} + w)} - Dw$$

where x, y are prey population, z is predator population, w is nutrient, y_{pr}, y_{pc}, y_{nc} and y_{nr} are yield coefficients. Experimenting with this particular system resulted in a range of outcomes. System could be transformed in and out of chaotic state by changing values of parameters. It was also possible to bring system in and out of chaos by changing dilution rate which is experimentally controllable parameter. Here we present 2 cases where given dynamical system is in chaos in first and out of chaos in second case. This state change was brought about by changing the dilution rate of the system.

Dilution rate of system in Figure 3.2 was 0.9 whereas that of system in Figure

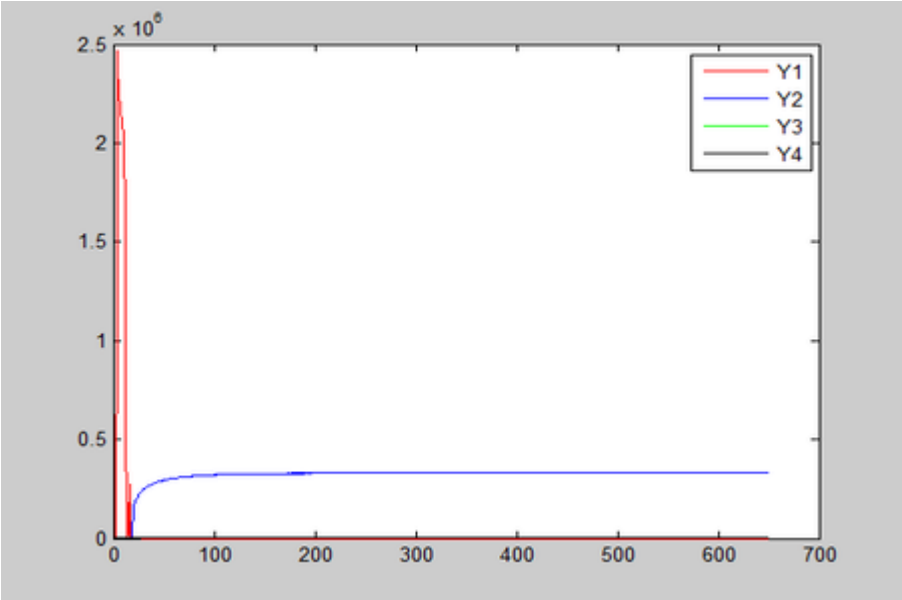

Figure 3.3: 4 variable system out of chaotic state

3.3 was 0.1 (1/day)

3.6 Conclusion

In all we can conclude that a given system can be transformed in and out of chaotic state by changing parameter values which are experimentally controllable. If we can develop a system which can generate the values of parameters for which the system stays in chaos and values for which it does not, it will be of great help to perform biological experiments dealing with population evolution of micro-organisms.

Chapter 4

The genetic Algorithm

4.1 Introduction

Previous chapter displays results of experimental variations in the dynamical system. As displayed, variation in starting concentrations, growth rates of bacteria and dilution rate parameters may bring the given dynamical system in and out of chaotic state. How to determine that for which set of parameters given dynamical system might enter the chaotic regime is an interesting problem. As there is no single specific solution for this problem different approach than usual solution finding algorithm is needed. A genetic evolutionary algorithm is one of the methods to reach a solution to a problem where non-specific solution is needed. Here, starting conditions of the dynamical system can be viewed as characteristics of the system which can be evolved using the genetic algorithm and the optimal solution is reached after several generations. Hence, we used the genetic search algorithm to address this problem.

4.2 The genetic Algorithm

4.2.1 Background

A genetic algorithm is search heuristic that mimics the process of natural selection. This is routinely used in the optimization problems. Finding a solution using a genetic algorithm usually consists of evolving of optimal solution from set the candidate solutions. The steps can be discribed as explained below.

A genetic algorithm starts with a set of random candidate solutions. These candidate solutions differ from each other in terms of the values of parameters which decide the behavior of a solution. For every candidate solution, based on values of its parameters fitness is calculated. Similar to process of evolution, candidate solutions with maximum fitness are allowed to reproduce to form the next generation. A section of candidate solutions is retained and continues in the next generation. Now, again fitness of all newly formed candidate solutions is calculated and the whole process is repeated for several generations. In the end, candidate solution with the highest fitness is taken as an optimized solution to the given problem.

4.2.2 Implementation

As stated above, the genetic algorithm starts with random solutions. If starting solutions were plotted in a phase space, they can be represented as n dimensional data points in the n dimensional ellipsoid. All these starting solutions are allowed to *evolve* with the hope that just like in the process of evolution the fittest solution will be selected and passed on to the next generation as a result of which optimal solution can be reached. Theoretically every generation passed, improves the final solution by small amount i.e. it gets closer to the optimal solution by small amount.

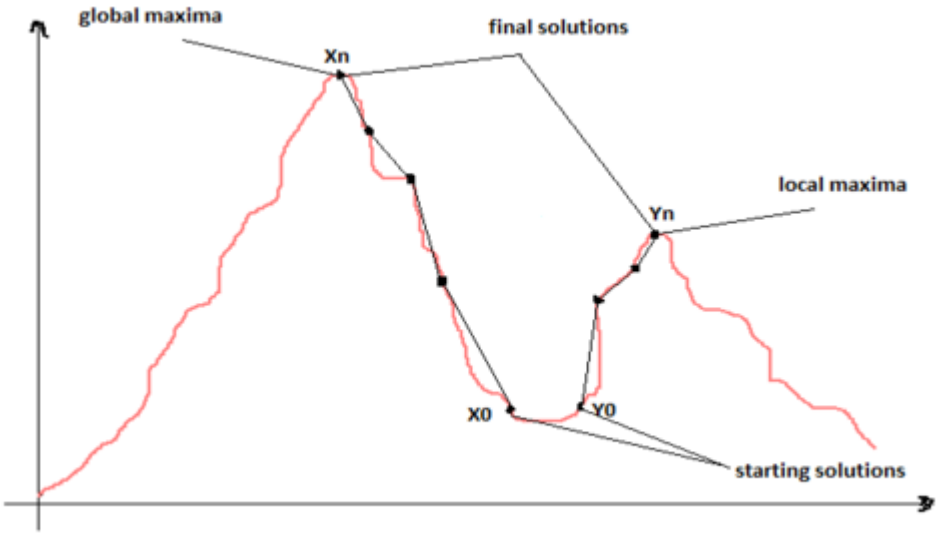

Figure 4.1: Genetic Algorithm - Phase Space

This phenomenon is explained in the Figure 4.1. It shows a three dimensional phase space. X_0 and Y_0 are the points representing random starting candidate solutions and X_n and Y_n are the final solutions returned by the genetic algorithm as a result of evolution. The genetic algorithm evolves the best performing solution in each generation. Hence, in the end of each generation we get a new solution which performs best among the overall population. These best solutions in the intermediate generations are shown by the points on the trajectories between points X_0 to X_n and Y_0 to Y_n. However, these two trajectories seem to move in different directions which highlights an important observation about the working of a genetic algorithm. The same genetic algorithm acts on X_0 and Y_0. Optimal solutions in both cases returned by the genetic algorithm are significantly different from each other. Thus, it is clear that the final solution given by the genetic algorithm is dependent on the initial candidate solution population. Moreover, if same initial condition is given as input, the genetic algorithm still may end up producing different result than previous

run. This is in accordance with the fact the there is an element of randomness in the evolutionary process.

The fact that the genetic algorithm is dependent on the initial candidate solution population becomes important in a solution space having local maxima. The trajectory followed by the genetic algorithm when fed with initial candidate solution Y_0 ended up at local maxima which has lower value than global maxima. There is no theoretical guaranty that the genetic algorithm will not end up at local maxima. Hence, the genetic algorithm should be properly seeded to reach the optimum solution. This can be achieved by trying random starting candidate solutions.

4.2.3 Approach

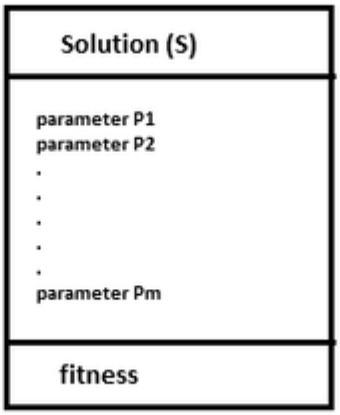

Figure 4.2: Expected Solution Format

To help clear the understanding of the working of the genetic algorithm, we present a symbolic example implementation. Expected solution from the genetic algorithm is a solution shown in Figure 4.2. A population of candidate solutions is given as input to the algorithm. A candidate solution is represented in exactly same way as

an expected solution. A randomly created population of candidate solutions is given as an input to the algorithm. Size of this randomly created input population depends on the implementation. It may be fixed or variable. As input candidate population is randomly created, each candidate solution is different from other and there is no relation among them or their parameter values. Figure 4.3 shows input candidate solution population to the algorithm.

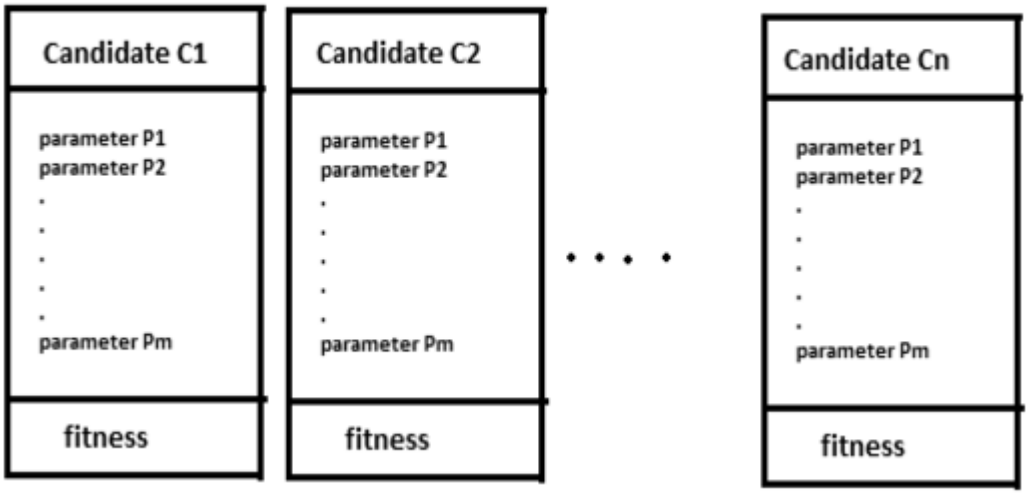

Figure 4.3: Expected Solution Format

Next importent step in the algorithm is to calculate the *fitness* of each candidate solution. *Fitness* of a candidate solution generally depends upon performance of the solution or closeness of candidate solution to the optimal solution. Therefore, if the candidate solutions in the population represents combination of parameter values used to recognize a face, *fitness* of candidate is more if that particular parameter values combination has higher rate of face recognition. In our case, we needed a combination of parameter values for which the mathematical model shows a chaotic behavior. Positive Lyapunov Exponent being an indication of chaotic behavior, it

makes sense to calculate *fitness* based on the same. By treating Lyapunov Exponent as a *fitness*, the genetic algorithm ends up returning a solution which has maximum fitness i.e. maximum Lyapunov Exponent.

Once the *fitness* for each candidate solution is calculated, top performing candidates can be selected to be a part of next generation and also to reproduce *offsprings* which are *similar* to them in terms of parameter values. A combination of these top performing candidates and new *offsprings* forms next generation and poor performing candidates are discarded. *Fitness* calculation and selection steps are repeated for next generation again and the whole process continues through number of generations. At the end of last generation, top performing candidate solution is returned as a solution by the genetic algorithm.

The complete process is explained in the flowchart shown in Figure 4.4.

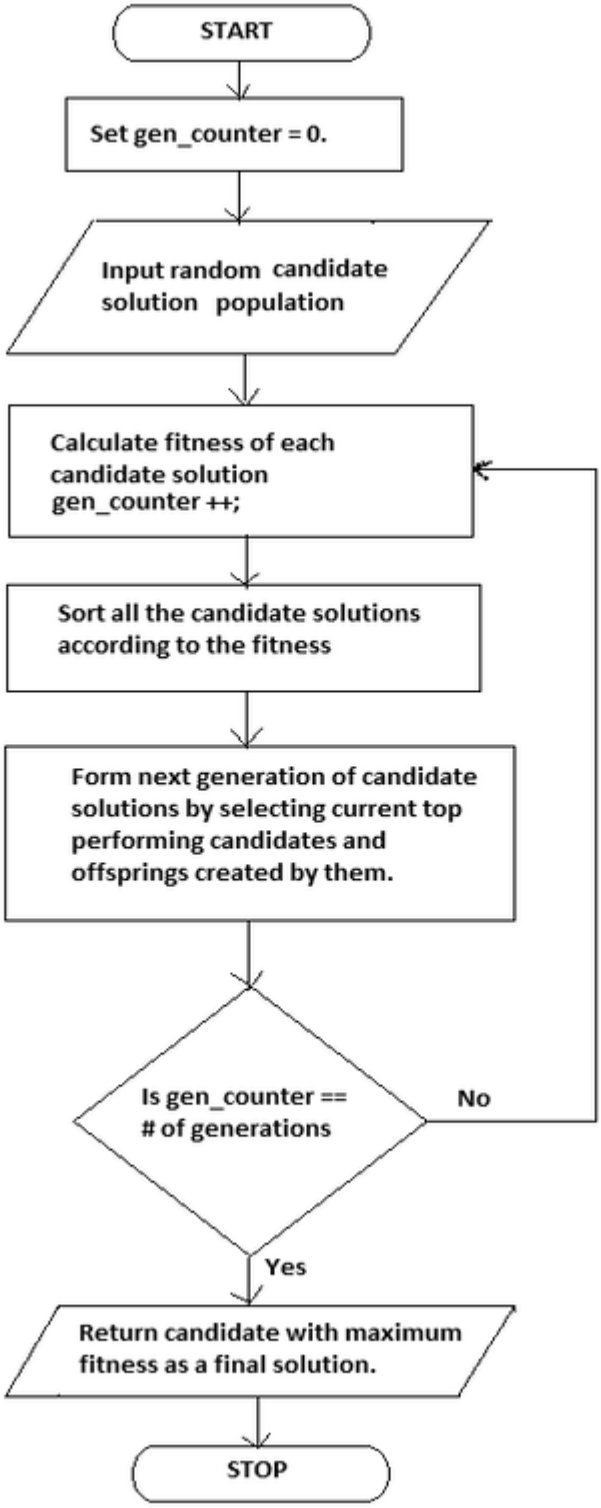

Figure 4.4: Genetic Algorithm Flow

4.3 Genetic Algorithm for Becks Equations

Becks dynamical system discussed in the previous chapter consists of initial concentrations of each species and nutrient. It also uses the growth rate and the death rate of species and the dilution rate used in the experiment. For given species, the growth rate and the death rate is unlikely to change by large values. Hence, varying initial concentrations to see if it takes system into and out of chaotic state is a most appropriate way to experiment with the system. Because of this we designed the candidate solutions to the problem with parameter values of initial concentrations of the species. To get given dynamical system into a chaotic state, selected parameters should cause system's Lyapunov Exponent to become positive. To draw final solution with higher and higher positive Lyapunov Exponent, we set it as a fitness measure of the candidate solution as mentioned in previous section. Thus, candidate solution with maximum Lyapunov Exponent is treated as the best performing solution and next generation of candidate solutions is formed from the candidate solution with maximum Lyapunov Exponent from the previous generation. The genetic algorithm thus returns the solution with local maximum Lyapunov Exponent. This solution can be used as an input for a desired chaotic state of the system.

4.4 Algorithm

```
Let n be the number of generations to be simulated.
1. Initialize a set of 20 candidate solutions with random
   parameter values where parameter values represent initial
   concentration of all species and nutrients.
2. Calculate Lyapunov Exponent as a fitness for each candidate
```

```
   solution.
3. Select candidate solution with the maximum Lyapunov
   Exponent.
4. Form 10 new candidate solutions by varying parameter values
   of candidate solution selected in step 4.
5. Top 10 candidate solutions from previous generation and 10
   new candidate solutions form the population pool.
6. Repeat steps 2-5 for n generations.
```

4.5 Results

The genetic algorithm described above was run for 50 generations on a population set of 20 candidate solutions. The returned values were then compared to experimentally found values for a dynamical system in a chaotic state.

Parameter	Experimental	Genetic Algorithm
Prey1	2.1e-7	4.05e-7
Prey2	4.56e-7	1.41e-6
Predator	4.3e-7	8.90e-7
Nutrient	4.0e-5	2.2e-4
Dilution rate	0.9	0.0011

It can be observed from the table that the parameter values returned by the genetic algorithm are from the region where the dilution rate is low. This is significantly different from the value which was obtained experimentally. This shows that the genetic algorithm may be able to find the unidentified parameter values which also might lead to the system in chaotic state.

Chapter 5

Metropolis Algorithm

A genetic algorithm is designed to return an optimized solution for a given problem. This is why we used it to find a parameter value combination so that mathematical system using those parameter values remains in a chaotic state. But as we know that there are many parameters which control this chaotic behavior of a mathematical system, it is logical to conclude that more than one combination of those parameter values causes the mathematical system to be in a chaotic state. Optimized answer may not represent for all the parameter value combinations possible which can keep system in chaotic state. Thus, instead of just having an optimized answer, a set of points in a phase space which represents all parameter value combinations sufficient for chaotic state might prove to be a desirable. However, computing set of all points which represent chaotic behavior in a mathematical system may be very expensive. Instead, a set of points which may act as representative set for all points can be computed. An algorithm used to compute this set of representative points is also known as the Metropolis algorithm.

5.1 Random Walk

The goal of generating a representative set of points which shows the distribution of entire set of points is achieved by randomly generating chain of points where every point in the chain differs the previous one in exactly one co-ordinate value. The process of generating data points randomly by varying the co-ordinates randomly is also known as a 'random walk'. Starting with the first point, every next point is generated by randomly selecting a co-ordinate and introducing random change in its value. This new point is then selected to be a part of the representative set with some probability.

'Random walk' as the term suggests, is not required to be simulated in the exact same manner as discussed above. A walk can be truly random instead of changing value only in one dimension at a time. Depending on the application different types of random walks are used in various fields i.e. ecology, economics, psychology, biology etc. the Metropolis algorithm discussed in this chapter requires random walk to be simulated with change in only one dimension at a time. Figure 5.1 shows an example of random walk in 2D plane. O(0,0) is a starting point of the walk. From there path represented by straight lines is followed. As change can be made only in one dimension, we see all the transition lines parallel to X and Y axes. It can also be observed that magnitude of change is also random in the random walk.

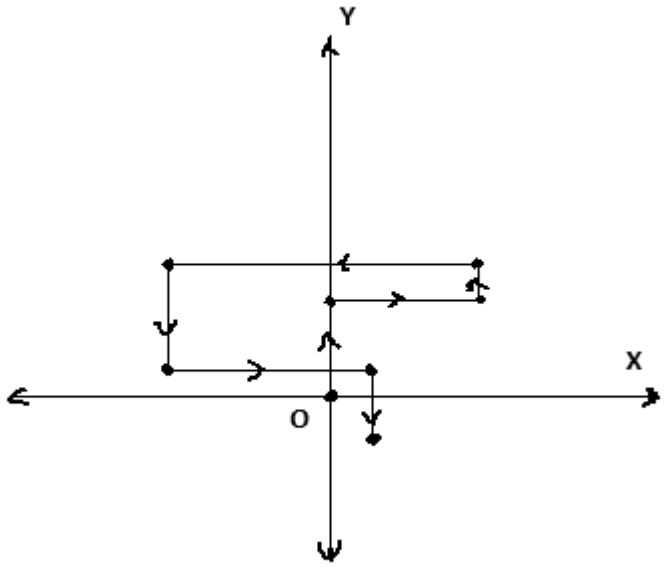

Figure 5.1: Random walk

The notion of points generated by random walk acting as a representative set of points seems counter intuitive. But, it can be showed that this generated set possesses same statistical properties as the complete set of points. The whole notion is based on a concept that if one takes a random walk, at some point he will come back to the point where he started and the places he visited on his path act as a representative set of the whole neighborhood. The points visited during the random walk in the Metropolis algorithm form a Markov chain. This is explained in the next section.

5.2 Markov Chain

Markov chain is named after a Russian mathematician Andrey Markov. It is a mathematical system which forms a chain of states that can transition from one state to another based on some probability. States which form a chain are valid configurations

system can be in. States are connected with other states by links called transitions. The process starts with the initial state and moves through the chain of states. At each state process may move to next state or remain in the same state through links with some probability. This probability is called transition probability. This transition probability does not depend previous states system was in. It only depends on the current state system is in.

This can be explained with an example shown in Figure 5.2. Figure shows an ordinary Markov chain consisting of three states $S = \{S_1, S_2, S_3\}$. Each state is defined by set of parameters X. X_1 is parameter set of state S_1, X_2 is for state S_2 and so on. Each of X_1, X_2 and X_3 may contain multiple elements such that X_1 can be represented as $X_1 = \{X_1^1, X_1^2, ...X_1^n\}$. X_2 and X_3 can also be represented similarly. P is a set of transition probabilities. The transition probability of a particular state transition is denoted by P_{ij}, i.e. probability of transition from state i to j. So probability of transition from state S_1 to S_2 is denoted by P_{12}. As can be seen in Figure 5.2 system can transition from any state to any other state i.e. transition probabilities for all the links are non-zero. Although, this is not a necessary condition for the structure to be called as Markov chain.

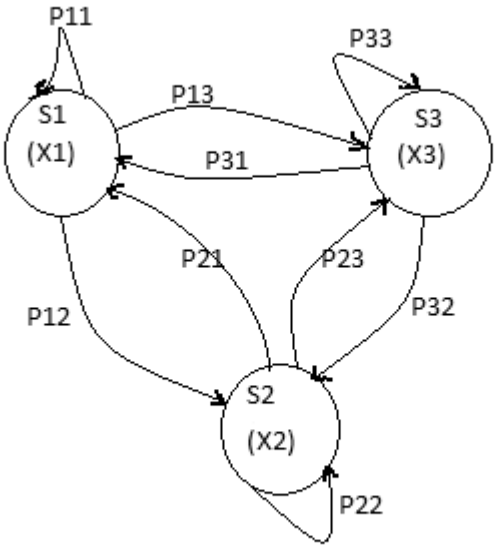

Figure 5.2: Markov Chain

Markov chain is an apt model to represent a random walk. In random walk, next state is dependent only on current state which is similar to the Markov chain system. To obtain a next state while performing a random walk, sampling of current state is needed so that next state is very close to current state. This is achieved by checking of potential next state obtained by changing only a single parameter of current state. Markov chain in theory can have next state with more than one parameter different than current state but Markov chain simulating random walk follows a restriction of having only single parameter value difference between two successive states.

The difference between an ordinary Markov chain and random walk can be explained with the help of example shown in Figure 5.3.

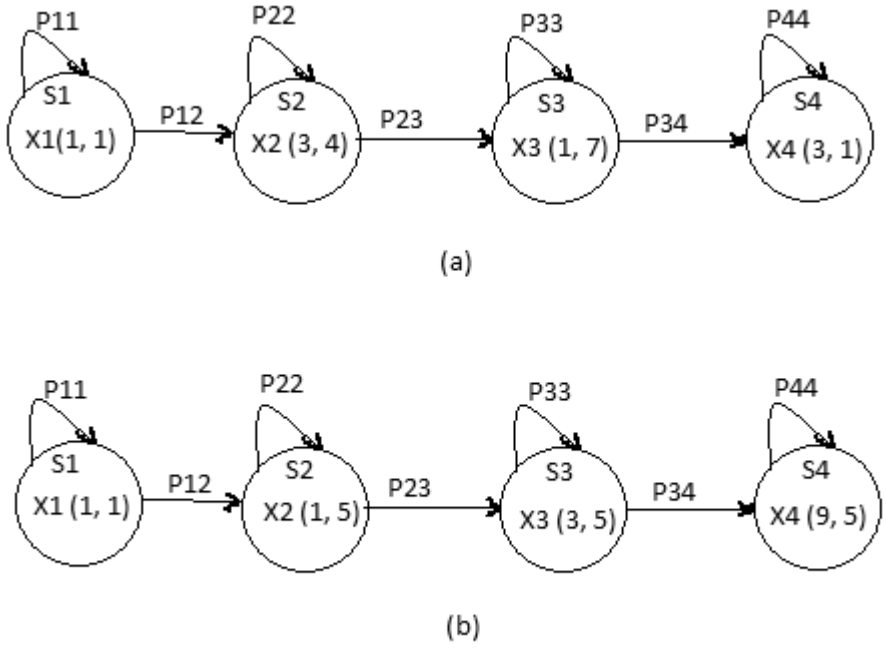

Figure 5.3: (a) ordinary Markov chain (b) Markov chain representing a random walk

Figure 5.3(a) shows an ordinary Markov chain. The initial state of chain has parameters $X_1 = (1, 1)$. The next state has parameters $X_2 = (3, 4)$. It can be seen that second state has all parameters different from initial state. Theoretically this Markov chain also can be interpreted as a random walk. But in a problem statement where possible number of states one can visit is infinite, it makes sense to make transitions between states that are most likely to happen. From a state with a particular set of parameters transitioning to a state with only single parameter different is much more likely event than that of transitioning to a state with more parameters have different values. As shown in Figure 5.3(b) each successive state differs the previous one by only single parameter value. In the next section we describe the Metropolis algorithm, advantages it brings to the data representation and its implementation details.

5.3 Metropolis Algorithm

As mentioned earlier we are using this algorithm to represent the data so that a researcher may find it easier to visualize chaotic landscape of the mathematical system at hand. The Metropolis algorithm is based on random walk approach through the available data and this characteristic of the algorithm helps enabling us to represent best approximation of entire dataset at hand graphically. The representative set generated by the algorithm is highly useful as there are ways available to represent it even if data that is in high dimensional form. One of the techniques that used to represent high dimensional data is 'parallel co-ordinates'. This would be discussed later in this section. The Metropolis process of visiting different points by changing parameter value in one dimension at a time, gives a closely linked Markov chain. This chain contains representative set of points which cause the system to be in chaotic state. Though the Metropolis algorithm cannot produce hard limits on the values of parameters of the mathematical system at hand to keep it in or out of chaotic state, it may give an estimation of boundary values. Its main advantage comes from the ability that may give a sense of relation between parameter values for chaotic behavior. For example, in case of 4-dimensional Becks equations executions of the Metropolis algorithm showed a relation between dilution rate and initial nutrient concentration i.e. their relative values leading to chaotic behavior of the Becks system.

5.3.1 Implementation

Approach to simulate the Metropolis algorithm starts with an initial point in a Markov chain. This initial point is taken from the Genetic algorithm output. This is not a necessary step but doing so produces better results of generated representative set. This is somewhat intuitive to assume as the Genetic algorithm gives a starting point

which causes system to be in chaotic state. As we are generally interested in looking at parameter value representation which keeps the system in chaotic state, starting at a point responsible for chaotic state helps generating more points as points closer to initial point may show similar behavior.

The entire set of points showing chaotic behavior has a very complex distribution. Let that complex distribution function be $\pi(x)$ (x may contain multiple parameters such that $x = \{x^{(1)}, x^{(2)}...x^{(n)}\}$). Also, $\widetilde{\pi}(x)$ be some known function which we will simulate to generate representative set of points for $\pi(x)$. Generally a Boltzmann distribution function is used as a $\widetilde{\pi}(x)$. The Boltzmann distribution is used to simulate a distribution with various possible states of points. This works very well with our need as all points in our sample set have one of the two possible states i.e. chaotic and non-chaotic.

The Boltzmann distribution function is of the form $\widetilde{\pi}(x) = e^{E(x)}$ where $E(x)$ is an energy function which is used to differentiate between two or more states to which points belong. This energy function can be replaced by the Lyapunov Exponent calculating function as states of points in our distribution depend on the Lyapunov Exponent value.

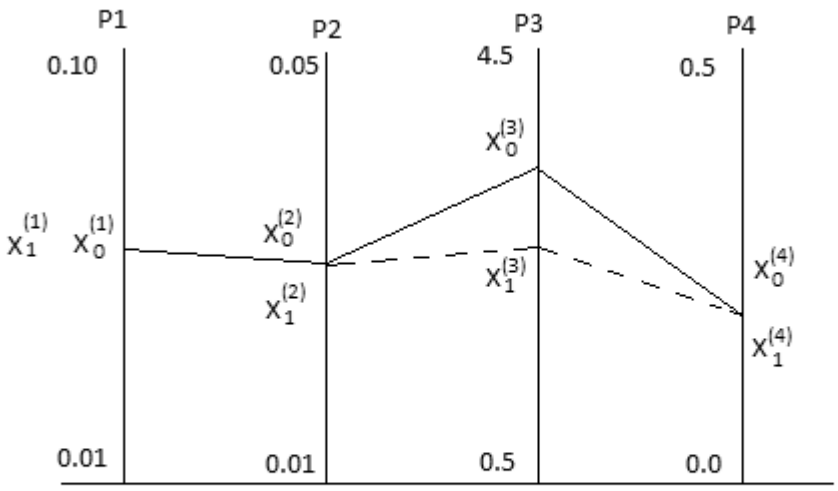

Figure 5.4: Parallel Co-ordinates

The relation between $\pi(x)$ and $\widetilde{\pi}(x)$ is given by: $\pi(x) = \frac{\widetilde{\pi}(x)}{z}$ where z is computationally very expensive to calculate. We start at the initial state $x_0 \in x$. Next potential point is obtained by sampling against $\widetilde{\pi}(x)$. This sample is done by selecting a single parameter of x_0 and adding a random value to it. This can be understood by observing the Figure 5.4. First point is represented by $x_0 = \{x_0^{(1)}, x_0^{(2)}, x_0^{(3)}, x_0^{(4)}\}$ and second point by $x_1 = \{x_1^{(1)}, x_1^{(2)}, x_1^{(3)}, x_1^{(4)}\}$. $P_1 - P_4$ are the parameters that represent the co-ordinates of x. Here, x_1 is obtained from x_0 by randomly changing 3^{rd} co-ordinate.

In the next step we sample a variable u against uniform distribution $p(0, 1)$. New potential point x_1 can be accepted if the ratio $\frac{\widetilde{\pi}(x_1)}{\widetilde{\pi}(x_0)}$ is greater than u. Else, x_1 is obtained again by resampling in same way described above. Once x_1 is accepted rest of the points also can be generated using the same process. This process can be

summarized as follows:

for i = 1, 2, 3,... n-1

1. Sample $\widetilde{\pi}(x)$ against symmetric probability distribution $Q(x|y)$.
2. Sample u against a uniform distribution $p(0, 1)$.
3. If $u < \frac{\widetilde{\pi}(x)}{\widetilde{\pi}(x_i)}$ then, accept x as x_{i+1}.

All generated points are too close in parameter values with each other. Hence, a subset of these points can be selected to shown in the results. This simplifies the results and make them readable.

5.3.2 Hybrid Approach

As mentioned earlier the Metropolis algorithm usually performs better when the initial point or state is taken as an input from the genetic search algorithm. There is a logical explanation for this behavior by the Metropolis algorithm. Of all points which can be used to represent parameter value sets for the mathematical system, only subset of those points leads to a chaotic behavior. At some points system may not be in valid state and at some points even if system is in valid state, it may not exhibit a chaotic behavior. Typically system follows chaotic regime only in selected set of points i.e. to a limited region in a phase space. If the Metropolis algorithm starts off in a region in a phase space which is non-chaotic, random walk from that point is less likely to reach a region with chaotic state or it may take a large number of steps before random walk reaches chaotic space. This in turn may directly affect the number of chaotic points collected as representative set.

This can be illustrated with an example in Figure 5.5. Consider a two dimensional phase space where P_1 and P_2 are the two parameters. Grey area shown in figure

contains value combinations of parameters P_1 and P_2 which lead system to have chaotic state. If the Metropolis algorithm starts off at point X_1 it has much larger chance to get large number of points which belong to chaotic region. Instead if it starts off at point X_2 then the random walk performed by the Metropolis may end up taking lot of points which do not lead to a chaotic state of a system.

This is why hybrid approach is used to get maximum chaotic points representation in the dataset.

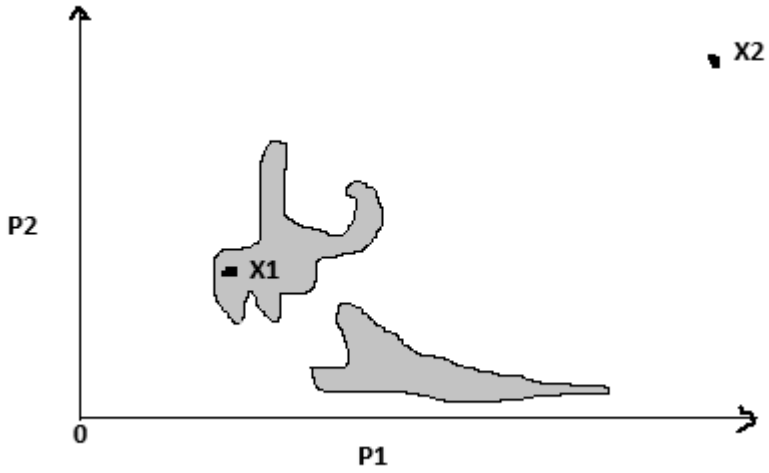

Figure 5.5: Hybrid algorithm approach

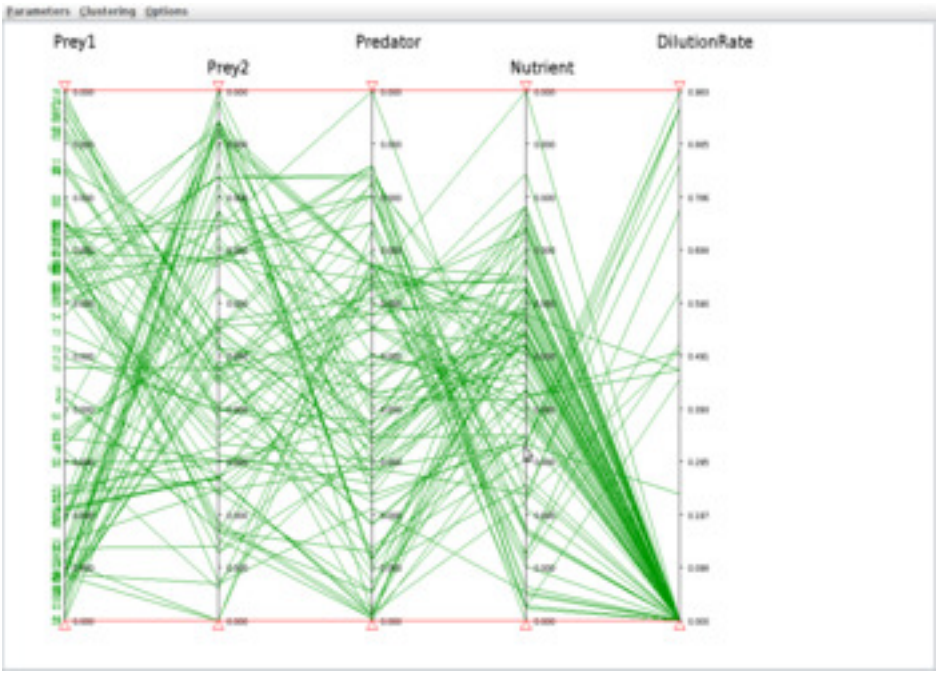

Figure 5.6: Parameter values leading to chaotic state of system

5.4 Results

We executed the Metropolis algorithm on the Becks set of equations and used a starting point given by the genetic search algorithm. To display the results we used parallel co-ordinate plotting system which is also shown in previous sections. This system is very useful in displaying high dimensional data and helps giving sense of relation between parameters being searched by the Metropolis algorithm.

When the Metropolis algorithm was executed, we got a representative set of points which lead to a system with chaotic state. When plotted all the points in parallel co-ordinate format they showed large number of values concentrated with dilution rate parameter close to zero.

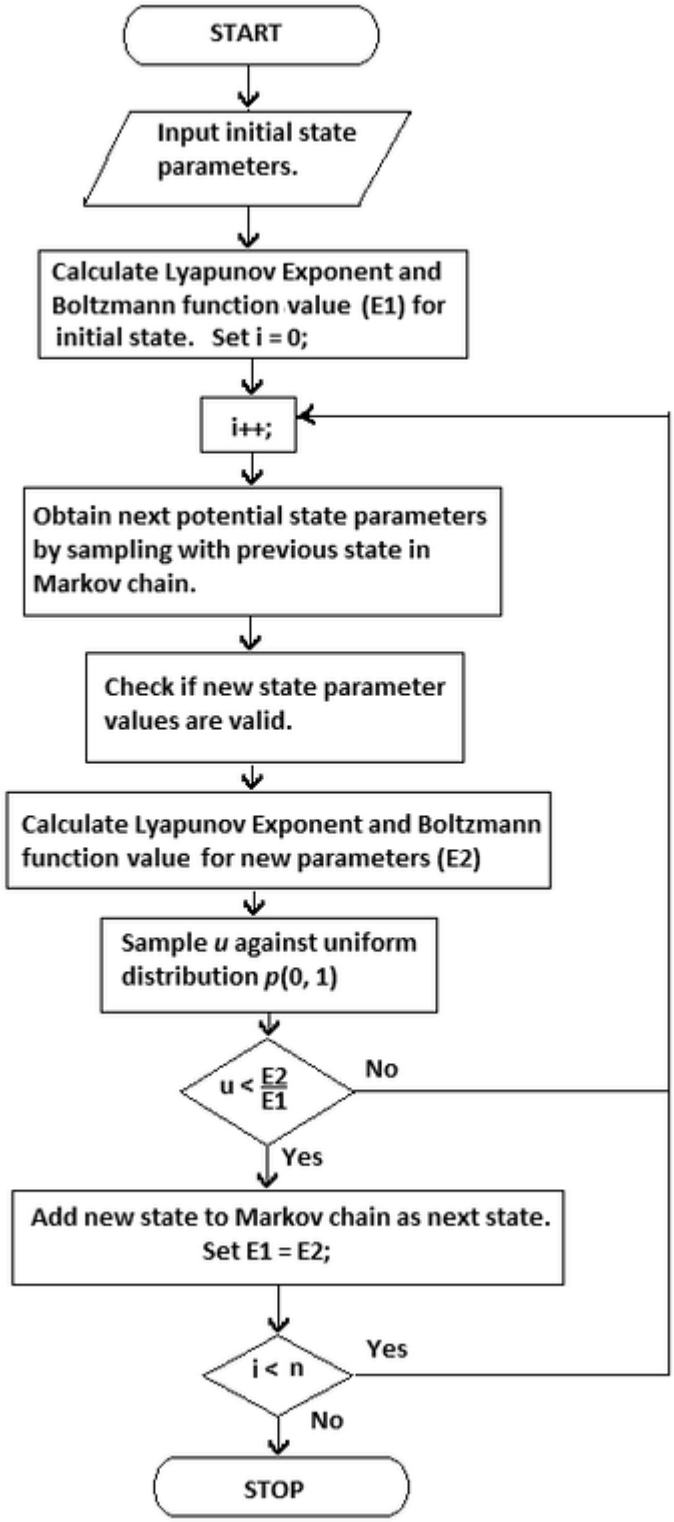

Figure 5.7: Metropolis Algorithm Flowchart

Chapter 6

Chaos Analysis Library

Previous chapters have described the basics of Chaos theory, dynamical systems, genetic and Metropolis algorithms. This chapter describes structure and working of Chaos Analysis Library (CAL) in detail. The experimental simulation of working of the mathematical models and algorithms is highly essential for any theory to get approved. CAL has facilitated the experimental simulations of tests done on Becks' mathematical model. It may perform similar role for many other mathematical models for the researchers. Next section discusses prime reasons which led to the development of this library. After that implementation details of CAL are discussed.

6.1 Need for CAL

As discussed before, study of chaotic behavior of various phenomena is rapidly developing research area in computational biology. Researchers in the fields like systems biology, biostatistics, evolutionary biology etc. often want to explore effects of chaos and chaotic state on the mathematical and in turn real biological systems. Due to large number of personnel working in this area, an easy to use tool which can be used

for any type of mathematical model containing dynamical systems will have a greater impact on the way biological experiments are performed in the future.

Personnel working on experiments in the field of systems biology, evolutionary biology etc are not expected to have extensive knowledge of programming. Instead of writing long programs or scripts for each type of mathematical model, writing small set of instructions defining equations of new mathematical model and using the rest of the functionality through CAL is very much easier task to do. This may also help speed up the overall research process.

Extendibility: CAL is extendable. Even if CAL's algorithm set contains only two algorithms i.e. the genetic search and Metropolis algorithm, it can be extended to support any number of other algorithms or mathematical operations. Once a new algorithm is added to CAL's algorithm set it can be used for any previously defined mathematical models using this library.

CAL facilitates use of hybrid model discussed in previous chapter where combination of the genetic search and the Metropolis algorithm is used to develop representative set of points that keep system in chaotic state.

In the next section implementation details of CAL are discussed.

6.2 Implementation Details

In this section we will discuss implementation details of Chaos Analysis Library, its overall structure and ways to use it. CAL is designed such that it reduces the need of rewriting of lot of programs and enables a user with a good toolset without having a steep learning curve. User can extend library by following minimal set of easy to follow instructions. This in turn helps in expedite the research work in effective way.

CAL is implemented on the top of Apache Commons Math API. As shown in

figure 6.1, Apache Commons API sits on the top of Core Java API. CAL is designed so that it calls APIs from Apache Commons library putting it in the upper layer to that of Apache Commons library. CAL consists of two main components as shown in figure 6.1. 1. Mathematical Models 2. Algorithms. These two components are responsible for facilitating new mathematical model implementation and new and old algorithms support for mathematical models.

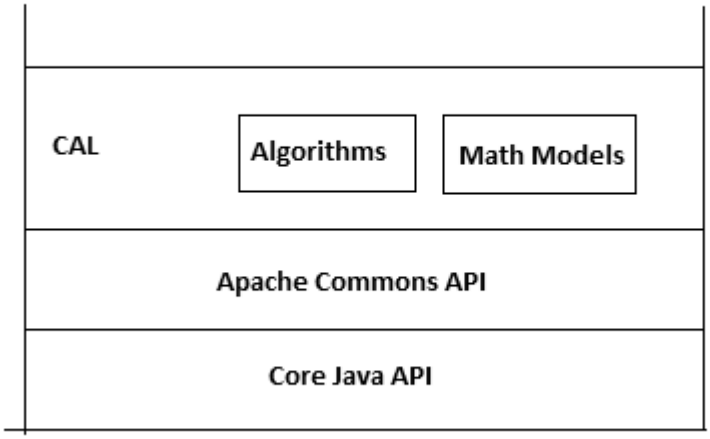

Figure 6.1: Chaos Analysis Library (CAL) API structure

6.2.1 Mathematical Models

This component is responsible to facilitate implementation of new mathematical models and also make sure that they follow the rules to be able to get support of algorithms from Algorithms component. At the base of this component is an abstract class MathModel. Every class that defines a new mathematical model has to extend MathModel class. MathModel class implements FirstOrderDifferentialEquations interface. This interface is a part of Apache Commons library and is responsible for implementing

first order differential equations.

MathModel class declares few methods which have to be implemented by every subclass.

1. getFeatureVector()

generateObject(double[] featureVector):

These two methods are essentially getters and setters of featureVector. 'featureVector' is an array which hold all the parameters which are supposed to be varied as a part of genetic search algorithm. For example, Becks 4 variable mathematical model consists of various parameters in the equations i.e. initial values of each variable $x_1, x_2, x_3, x_4 (a, b, c, d$ respectively), D (dilution rate), etc. Depending on different values of these parameters Becks system enters or comes out of chaotic state. If a genetic algorithm is supposed to vary only a, b and D and check for what values system goes into chaotic state, featureVector will contain only these three parameter values. getFeatureVector should return values of these three parameters as a featureVector. generateObject method accepts the featureVector (with same parameter values in same order) and generates a new object of mathematical model being used.

2. setFitness(double fitness)

getFitness():

These two methods are used to get and set the value of 'fitness' of parameter set. Typically fitness of parameter set is assigned to Lyapunov Exponent of the mathematical system with current set of parameter values. These two methods are used by genetic algorithm hence, have to be implemented. getFitness should contain the logic to calculate the value of fitness variable and setFitness should set the value of fitness.

3. getMetropolisParams()

setMetropolisParams(double[] metropolisParams):

Similar to featureVector, metropolisParams is a way to convey Metropolis algorithm which parameter values are to be varied to generate representative sample of points. getMetropolisParams should return parameter values which are to be varied and setMetropolisParams should implement the logic to set parameter values in the same order from the array passed to the method.

4. isValid(double[] y):

This method is used to check whether mathematical model is in valid state for given set of values. Mathematical models for some initial conditions may end up having one or more variable $(x_1, x_2...x_n)$ values to be negative. As these variables generally represent physical quantities negative values do not make sense and hence that state is considered invalid. Default implementation of this method checks only for values of $x_1, x_2...x_n$. If some other conditions are to be included to verify validity of state, this method should be overridden.

Here is implementation of MathModel class as shown below.

```
import org.apache.commons.math3.ode.FirstOrderDifferentialEquations;

public abstract class MathModel implements FirstOrderDifferentialEquations,
Comparable<MathModel>
{
    public abstract double[] getFeatureVector();
    public abstract MathModel generateObject(double[] featureVector);
    public abstract void setFitness(double fitness);
```

```
    public abstract double getFitness();
    public abstract void setMetropolisParams(double[] metropolisParams);
    public abstract double[] getMetropolisParams();

    final public int compareTo(MathModel o)
    {
        if (this.fitness < o.getFitness())
            return 1;
        else
            return -1;
    }

    public boolean isValid(double[] y)
    {
        boolean flag = true;
        for (int i=0; i<y.length; i++)
            if(y[i] < 0.0)
            {
                flag = false;
                break;
            }
        return flag;
    }
}
```

6.2.2 Algorithms

Algorithms component of CAL provides two main algorithms. The two main algorithms are 1. Genetic search algorithm and

2. Metropolis algorithm.

Genetic Search Algorithm

This algorithm as name suggests performs a genetic search. It evolves population of different parameter sets to return a parameter set which may keep the mathematical system in chaotic state. This algorithm is implemented inside class GeneticSearch. To use this algorithm a method shown below from GeneticSearch class should be used.

MathModel genetic(MathModel model, double[] y):

This method is easy to use. It requires object of a mathematical model class and initial conditions array as an input. The initial condition refers to the initial system variable values i.e. values of $x_1, x_2...x_n$. After the complete execution, this method returns an object of mathematical model class. This object consists of updated parameter values which successfully keep the system in a chaotic state. This object can be used to get parameter values causing a chaotic state or as an input to the Metropolis algorithm which produces the representative set of points.

Metropolis Algorithm

This algorithm performs random walk using the Metropolis algorithm over a given parameter space. This random walk then produces a set of points which may give a visual aid to show relation between two or more parameter values. This algorithm is implemented inside the class Metropolis.

generateDistribution(MathModel model, double[] y, double[] metropolisParams, double[] alpha):

This method generates the distribution of visited points during random walk. The arguments passed to the function are as follows:

1. model - object of mathematical model i.e. class MathModel

2. y - initial state of system

3. metropolisParams - initial values of parameters on which random walk is to be performed

4. alpha - this is the tolerance of each parameter under which value that parameter is to be varied.

For example, if dilution rate (D) is one of the parameters which is to be varied. If alpha value for D is 1.0 then value of D can be changed by 1.0 at the most in a single step. The value by how much to change D is selected at random from interval [0.0, 1.0] everytime. Choice of an alpha value is important as too high alpha value will allow value for that parameter to be varied on large scale and if chaotic behavior in that direction has limited range then it might directly affect number of points generated which show chaotic behavior and if it is too small, it might not be able to show expected varience in the value of parameter. Repeated executions of Metropolis algorithm may help determining right alpha value for each parameter.

CAL also contains one more mathematical model designed as a part of the Mathematical Models component. This model is designed after the Kravchenko set of equations. Along with the Becks equations experiments were also done on the Kravchenko equations.

Bibliography

[1] Harry A. Swinney John A. Vas Alan Wolf Jack B. Swift. "Determining Lyapunov exponents from time series". In: *Physica* 16D (1985).

[2] M. P. Boer B. W. Kooi. "Chaotic behavior of a predator-prey system in the chemostat". In: *Dynamics of Continuous, Discrete and Impulsive Systems Series B* (2003).

[3] R. A. Desharnais et al. "Chaos and population control of insect outbreaks". In: *Ecology Letters* 4.3 (2001), pp. 229–235.

[4] Lev V. Kravchenko Kinkolay S. Strigul. "Mathematical Modeling of PGPR inoculation into the rhizosphere". In: *Elsevier* (2005).

[5] Horst Malchow Klaus Jurgens Hartmut Arndt Lutz Becks Frank M. Hilker. "Experimental demonstration of chaos in a microbial food web". In: *Nature* 435-30 (2005).