Optimization of the Sensitivity of Avian Compass using Genetic Algorithms

Thesis

Submitted in partial fulfillment of the requirements of

BITS F424 Thesis

By

Tejas Bugdani

2012B5AB469P

Under the supervision of

Dr. J. N. Bandyopadhyay,

Department of Physics,

BITS Pilani

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

(RAJASTHAN)

8th May 2017

Acknowledgement

I am truly indebted and thankful to my supervisor Dr. J. N. Bandyopadhyay for giving me this opportunity and his support and guidance throughout the project. This work wouldn't have been possible without his patience and faith in me to learn and work in a new and emerging field of Quantum Biology without any previous experience.

I am grateful to the Physics department, BITS Pilani for allowing me to pursue this thesis.

I would also like to thank my family. This project wouldn't have been possible without their love and support.

Tejas Bugdani

CERTIFICATE

This is to certify that the Thesis entitled**, 'Optimization of the sensitivity of the Avian Compass using Genetic Algorithms'** and submitted by **Tejas Bugdani (ID No. 2012B5AB469P)** in partial fulfillment of the requirement of BITS F424T Thesis embodies the work done by him under my supervision.

Date:

Signature of Supervisor

Dr. J. N. Bandyopadhyay,

Assistant Professor,

Department of Physics,

BITS Pilani

List of Symbols and Abbreviations used

H: Hamiltonian

I: Nuclear Spin Operator

A: Hyperfine tensor with components (Ax, Ay, Az)

B: External magnetic Field

γ: Gyromagnetic Ratio

$\sigma(0)$: The initial state of the system (State at time=0)

$\sigma(t)$: State at time t=t

S_i : Spin state of ith electron

I2: 2x2 Identity Matrix

P^s: Singlet projection operator

$\rho s(t)$: fraction of radical pair in the singlet state at time=t

φs: Singlet producy yield

Δs: Sensitivity of the avian compass

Abstract

Thesis Title: Optimization of the sensitivity of the Avian Compass using Genetic Algorithms

Supervisor: Dr. J. N. Bandyopadhyay

Semester: Second **Session:** 2016-17

Name of the student: Tejas Bugdani **ID No.:** 2012B5AB469P

Abstract:

Recent studies in the past few decades have indicated a possible link between some biological processes and Quantum Physics. Quantum mechanical processes are believed to be the working force underlying some biological processes like Photosynthesis, charge transfer in DNA and Cellular Respiration. The impressive navigational ability of the migratory birds like the Robin are also shown to be derived from quantum mechanical processes undergoing in the bird's retina. This quantum mechanical navigational system is called an 'Avian Compass'. The bird's retina is believed to have photoreceptor pigments with molecular axis alignment in varying angles along a curve. Studies have shown that when light is incident on the bird's retina, electron transfer takes place from the photoreceptor pigments to adjacent pigments forming radical pairs which recombine after a short time. The Earth's geomagnetic field has an effect on the spin states of the exited electrons in the radical pairs. This effect results in varying amount of recombination chemical product formed, thereby creating a profile of the magnetic field and giving the bird a sense of direction. A quantum mechanical model has been proposed by my supervisor for this avian compass. The Hamiltonian of this model assumes the hyperfine tensor components and other elements like the initial state of the spins and the recombination rate. This thesis aims to optimize the sensitivity of this Avian compass with respect to the components of the hyperfine tensor by using Genetic algorithms.

TABLE OF CONTENTS

Chapter 1 – Introduction

The idea that some biological processes might operate on quantum mechanical laws has been around for about a century. Experiments in the past few decades showing evidence of link between biological processes and quantum mechanics have given rise to a new interdisciplinary field of study called 'Quantum Biology'. Some examples of such biological processes include photosynthesis, cellular respiration and avian magnetoreception. In this thesis, we are concerned with avian magnetoreception.

Avian magnetoreception is the ability of migratory birds to sense the geomagnetic field. Experiments with European Robins have showed that the navigational ability of the bird is dependent on the local geomagnetic field. It has been shown that if a sufficiently strong artificial radio-frequency (RF) field is introduced in the environment at a non-zero angle with the local geomagnetic field, the birds get disoriented. A bird's retina is believed to contain photoreceptor pigments (like cryptochrome), aligned along the curvature of the retina, with varying molecular axis directions. When light is incident on the retina, electron transfer takes place between adjacent pigments in the retina. This polarization leads to the formation of radical pairs (pair of charged molecules) with each molecule having an electron with unpaired spin. Under the influence of the geomagnetic field, the spins undergo either singlet or triplet transitions. The radical pairs formed are short-lived (lifetime of order of microseconds) and recombine to form singlet chemical product and triplet chemical product based on their spin states. The coherence time is estimated to be in the order of microseconds. The amount of the chemical product formed varies along the curvature of the retina and is dependent on the geomagnetic field and the molecular axis orientation. This information of the varying chemical product in the bird's retina is transferred to the bird's brain and it processes the information to correlate with the geomagnetic field. Thus, the profile of the local geomagnetic field is transferred to the bird's brain in the form of profile of varying amount of chemicals formed. This process has a coherence time of the order of microseconds. This results in the bird getting a sense of direction repeatedly.

The spin of the electron is coupled with the spin of its nucleus and the spin of the other electron in the radical pair. A study has shown that the spin of one of the electrons in the radical pair is effectively uncoupled from that of any other particle and is only dependent on the geomagnetic field. This mechanism of sensing the geomagnetic field is called as the 'Avian Compass'. The sensitivity of this avian compass is calculated as the difference of the singlet product yield between the ends of the field of vision of the bird. The higher the sensitivity, better is the navigational ability. This sensitivity is dependent on variables like the hyperfine tensor, the recombination rate of radical pairs and the initial state of the system.

The aim of my thesis is to optimize the sensitivity of this avian compass with respect to the components of the hyperfine tensor. Since the hyperfine tensor has 3 components, traditional optimization algorithms would be inefficient and may give the wrong answer. Genetic algorithms are used for this purpose. They generate random solution space, calculate the sensitivity for each one of the possible solutions, choose the most promising solutions as the next solution space. This procedure is repeated as many times as required to reach a tentative solution. Genetic algorithms imitate the nature’s process of evolution, by making constant improvements over many generations. They may not provide the best possible solution, but they provide many good solutions for the problem. Further chapters in the report will explain genetic algorithms in detail along with the derivation of the sensitivity equation and the results.

Chapter 2 – Derivation of the Sensitivity Equation

A simple qualitative model of the avian compass is used which involves coupling of the electronic spins with the geomagnetic field and with the molecular part of the radical pair. Experiments have shown that the spins of one of the radical pair electrons is effectively uncoupled from any other particle and is only affected by the geomagnetic field. The spin of the other electron is coupled to the molecular part. Taking S_1 and S_2 as the spins states of the two radical pair electrons, the Hamiltonian is written as,

$$\mathbf{H} = I.\mathbf{A}.S_1 + \gamma\, \mathbf{B}.(S_1 + S_2)$$

Where,

I = Nuclear spin operator = $\{\sigma x, \sigma y, \sigma z\}$ A = Hyperfine Tensor = diag{Ax,Ay,Az}

$S_i = \{\sigma x^{(i)}, \sigma y^{(i)}, \sigma z^{(i)}\}$ **B =** External magnetic field (Geomagnetic field)

γ = Gyromagnetic Ratio = $\frac{1}{2} * \mu_0 * g$ where g is taken as 2 (electronic g-factor of free electron) and, μ_0 =Bohr Magneton= 9.274e-24 $J.T^{-1}$

A general representation of the magnetic field in 3D space is,

B = B_0 (sinθ*cosΦ x + sinθ*sinΦ y +cosθ z)

Here, B_0 = Magnitude of the local geomagnetic field. B_0 is taken as 47e-06 T for all calculations.

Since we are only concerned about the lateral sense of direction, we can safely take Φ = 0 deg. This reduces the **B** equation to,

B = B_0 (sinθ x + cosθ z)

$$I.\mathbf{A}.S_1 = \begin{pmatrix} \sigma x & \sigma y & \sigma z \end{pmatrix} \begin{pmatrix} Ax & 0 & 0 \\ 0 & Ay & 0 \\ 0 & 0 & Az \end{pmatrix} \begin{pmatrix} \sigma x(1) \\ \sigma y(1) \\ \sigma z(1) \end{pmatrix}$$

$$= \begin{pmatrix} \sigma x & \sigma y & \sigma z \end{pmatrix} \begin{pmatrix} Ax\, \sigma x(1) \\ Ay\, \sigma y(1) \\ Az\, \sigma z(1) \end{pmatrix} = \text{Ax}\, \sigma x \otimes \sigma x(1) + \text{Ay}\, \sigma y \otimes \sigma y(1) + Az\, \sigma z \otimes \sigma z(1)$$

$\gamma\, \mathbf{B}.(S_1 + S_2) = \gamma\, (\mathbf{B}.S_1 \otimes I2 + I2 \otimes \mathbf{B}.S_2)$ here, I2 = identity matrix of 2x2 size

$$= \gamma\, (\text{Bx}\, \sigma x(1) + \text{Bz}\, \sigma z(1)) \otimes I2 + I2 \otimes (\text{Bx}\, \sigma x(2) + \text{Bz}\, \sigma z(2))$$

Since S_1 and S_2 are defined in different Hilbert spaces, we have to bring them into a common space. We carry out tensor product of S_1 with the identity matrix in the Hilbert space of S_2 and vice versa.

Since $I.\mathbf{A}.S_1$ and $\gamma\ \mathbf{B}.(S_1 + S_2)$ are both in different Hilbert spaces, we have to bring them into a common space by multiplying with each other's identity tensor 2x2 matrix. The Hamiltonian can now be calculated,

$$\mathbf{H} = (I.\mathbf{A}.S_1) \otimes I2 + I2 \otimes \gamma\ \mathbf{B}.(S_1 + S_2)$$

The Hamiltonian will be an 8x8 Matrix,

$$\begin{pmatrix}
\boldsymbol{Az} + \mathbf{2}\gamma\,\mathrm{B}\cos\theta & \gamma\mathrm{B}\sin\theta & \gamma\mathrm{B}\sin\theta & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{Ax} - \boldsymbol{Ay} & \mathbf{0} \\
\gamma\mathrm{B}\sin\theta & \boldsymbol{Az} & \mathbf{0} & \gamma\mathrm{B}\sin\theta & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{Ax} - \boldsymbol{Ay} \\
\gamma\mathrm{B}\sin\theta & \mathbf{0} & -\boldsymbol{Az} & \gamma\mathrm{B}\sin\theta & \boldsymbol{Ax} + \boldsymbol{Ay} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \gamma\mathrm{B}\sin\theta & \gamma\mathrm{B}\sin\theta & -\boldsymbol{Az} - \mathbf{2}\gamma\,\mathrm{B}\cos\theta & \mathbf{0} & \boldsymbol{Ax} + \boldsymbol{Ay} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \boldsymbol{Ax} + \boldsymbol{Ay} & \mathbf{0} & -\boldsymbol{Az} + \mathbf{2}\gamma\,\mathrm{B}\cos\theta & \gamma\mathrm{B}\sin\theta & \gamma\mathrm{B}\sin\theta & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{Ax} + \boldsymbol{Ay} & \gamma\mathrm{B}\sin\theta & -\boldsymbol{Az} & \mathbf{0} & \gamma\mathrm{B}\sin\theta \\
\boldsymbol{Ax} - \boldsymbol{Ay} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \gamma\mathrm{B}\sin\theta & \mathbf{0} & \boldsymbol{Az} & \gamma\mathrm{B}\sin\theta \\
\mathbf{0} & \boldsymbol{Ax} - \boldsymbol{Ay} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \gamma\mathrm{B}\sin\theta & \gamma\mathrm{B}\sin\theta & \boldsymbol{Az} - \mathbf{2}\gamma\,\mathrm{B}\cos\theta
\end{pmatrix}$$

We diagonalize the Hamiltonian to get its eigenvalues and eigenstates. Since H is 8x8 matrix, there will be 8 eigenvalues and 8 eigenvectors.

Suppose |m> and |n> are two eigenstates of the Hamiltonian. Every eigenstate can be represented as,

$$|m> = \sum_{x=1}^{8} a\,|x>$$

Where m = 1, 2, 3…8

We carry out time evolution of the Hamiltonian using Liouville-von Newmann equation,

$$\sigma(t) = e^{-iHt}\sigma(0)\ e^{iHt}$$

We assume the system to be in pure singlet state initially at t=0 and take σ(0) as,

$$c|s >< s|$$

Where, |s> = $\frac{1}{2^{0.5}} * (|01 > \ -|10 >)$

Since |s><s| is also the projection operator onto the singlet state,

$$P^s = |s >< s|$$

$$\sigma(0) = \frac{1}{2} * I2 \otimes P^s$$

Now we find out the variation of the singlet product, we calculate the fraction of radical pairs in the singlet state at time t as,

$$\rho s(t) = Tr[P^s \sigma(t)]$$

Where $[P^s \sigma(t)]$ represents the singlet part of the $\sigma(t)$. Therefore,

$$\rho s(t) = \sum_{m=1}^{8} < m|P^s \sigma(t)|m >$$

$$= \sum_{m=1}^{8} < m|P^s I2\, \sigma(t)|m >$$

By completeness relation, we can write,

$$I2 = \sum_{n=1}^{8} |n >< n|$$

Therefore,

$$\rho s(t) = \sum_{m=1}^{8} \sum_{n=1}^{8} < m|P^s|n >< n|\sigma(t)|m >$$

Expanding the second term,

$$< n|\sigma(t)|m > = < n\left|e^{-iht}\sigma(0)e^{iht}\right|m >$$

Since $< n|\omega n =< n|H$ and $\omega m|m > = H|m >$,

$$= e^{-i(\omega n)t} < n|\sigma(0)|m > e^{i(\omega m)t}$$

$$= e^{-i(\omega m - \omega n)t} < n|\sigma(0)|m >$$

$$= e^{i(\omega mn)t} < n|\sigma(0)|m >$$

$$= \frac{1}{2} * e^{i(\omega mn)t} < n|I2 \otimes P^s|m >$$

$$= \frac{1}{2} * e^{i(\omega mn)t} P^s nm$$

Therefore,

$$\rho s(t) = \sum_{m=1}^{8}\sum_{n=1}^{8} < m|P^s|n > \frac{1}{2} * e^{i(\omega mn)t} P^s nm$$

$$= \frac{1}{2} * \sum_{m=1}^{8}\sum_{n=1}^{8} e^{i(\omega mn)t} |P^s mn|^2$$

When we expand the above equation, we can observe that it is sum of terms which are complex conjugate of each other. So, in the final expression, only real part will remain. So we can write $\rho s(t)$ as,

$$\rho s(t) = \frac{1}{2} * \sum_{m=1}^{8}\sum_{n=1}^{8} |P^s mn|^2 \cos((\omega mn)t)$$

The system has varying proportions of singlet and triplet product yield depending on the time and Hamiltonian. As singlet and triplet are the only possible states,

$$\rho s(t) + \rho t(t) = 1$$

We assume the recombination rates of singlet and triplet molecules to be the same and equal to k. We calculate the singlet product yield by integrating over $\rho s(t)$ with respect to time,

$$\phi s = k \int_0^{inf} \rho s(t)\, e^{-kt} dt$$

$$= \frac{1}{2} * \sum_{m=1}^{8}\sum_{n=1}^{8} |P^s mn|^2 (\frac{k^2}{k^2 + (\omega mn)^2})$$

$$\phi s = \frac{1}{2} * \sum_{m=1}^{8}\sum_{n=1}^{8} |P^s mn|^2 f(x)$$

Where, $f(x) = \frac{k^2}{k^2 + x^2}$

The above derived equation for singlet yield is used for calculating yield at ϴ=0 deg and ϴ=90 deg for each different combination of values generated for (Ax,Ay,Az).

Sensitivity of the avian compass is defined as the difference in amount of singlet yields at ϴ=0 deg and ϴ= 90 deg with the geomagnetic field.

$$\Delta s = \left| \phi s^{max} - (\phi s)^{min} \right|$$

$$= |\phi s^{0\ deg} - (\phi s)^{90\ deg}|$$

The objective is to maximize the Sensitivity Δs with respect to (Ax, Ay, Az), taking k as a constant, with the help of genetic algorithms.

Chapter 3 – Genetic Algorithms

Genetic algorithms are optimization algorithms used for optimizing functions with multiple variables. They are different than traditional approach to optimization which involves varying the value of the variables linearly, moving in one direction, generating data for the entire range and then finding the global maximum (minimum). For functions having more than 2 variables, it becomes a time consuming and computationally expensive task when using traditional algorithms. Genetic algorithms work in a different manner by generating random sets of possible solution values which span across the entire permitted range, and then moving towards the more promising solutions and disregarding the solutions which are farther away from the optimum value.

Genetic algorithms are inspired from the natural evolution process. In nature, in every generation of a species, the part of the population with better survival traits have a higher tendency to mate and higher chance of passing their traits to the next generation. Therefore, every generation is on average better than the previous generation. Genetic algorithms use this principle to move towards the optimum solution generation by generation. Genetic algorithms do not necessarily always give the best possible solution, but they provide a small number of good solutions.

The terminology used in genetic algorithms is explained below –

Population: The solution space at any given point of time

Phenotype: Population in real world solution space

Genotype: Population in computational space

Chromosome: An element of the population

Gene: An element position in the chromosome

Allele: Possible value a gene can take

Generation: each stage of population

Fitness Function: function to be optimized

3.1 Encoding- The encoding of our current sensitivity optimization problem is as follows:

Fitness function: Sensitivity (Δs)

Variables: Components of the Hyperfine tensor (Ax, Ay, Az)

Population: Variable (usually taken 10^5)

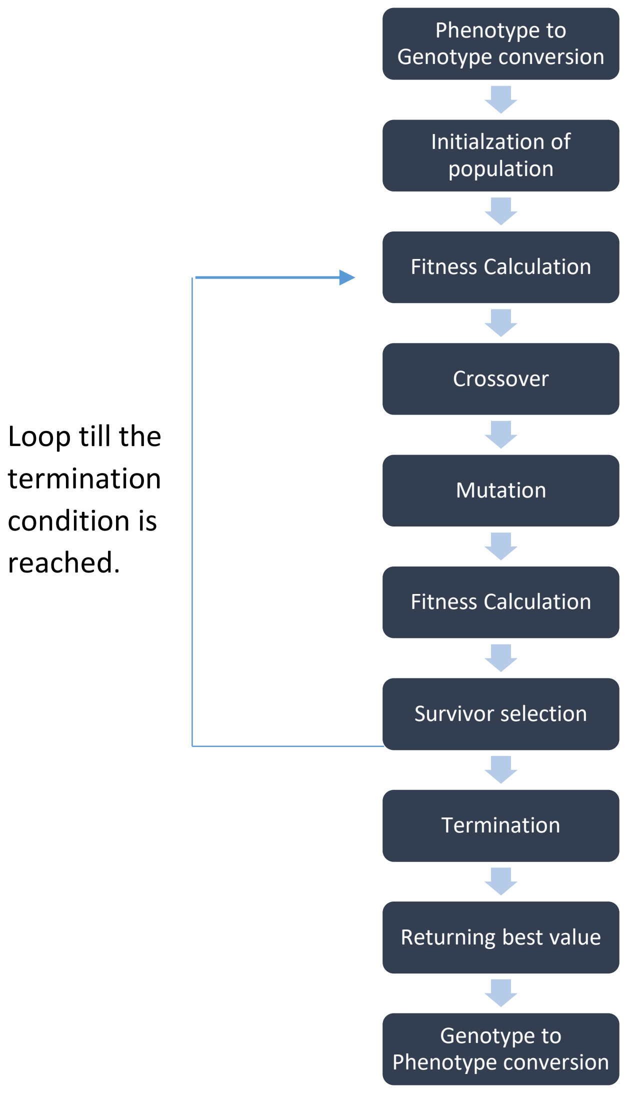
Phenotype: Range of each variable is of the order of $10^5 to\ 10^8$

Genotype: Binary representation of each variable

Chromosome: 30 bytes long character string (10 bytes for each variable)

Allele: Character '0' or '1'

The process of genetic algorithms is explained below in the diagram –


Phenotype to Genotype conversion
Initialzation of population
Fitness Calculation
Crossover
Mutation
Fitness Calculation
Survivor selection
Termination
Returning best value
Genotype to Phenotype conversion
Loop till the termination condition is reached.

The Genetic Algorithm Process Flow Diagram

3.2 Crossover Functions:

Crossover functions are analogous to mating in natural evolution. They allow chromosomes with higher fitness function values to mate and produce children which have a higher chance of giving a higher fitness value. Crossover takes place between two chromosome of the same population. When the genotype is binary representation, there are multiple ways of implementing the crossover function like,

- One-point crossover: The binary string is broken down at a fixed point and the one half of a chromosome is swapped with the corresponding half portion of the second chromosome.
- Two-point crossover: The binary string is divided at 2 points in the string and the portions at the end are swapped with corresponding portions of the second string
- Uniform crossover: Each child is formed gene by gene by taking the corresponding gene from one of the parents. The probability for a particular gene to be taken from a parent is 50%.

3.3 Mutation Functions:

In natural evolution, mutation takes place with low probability in a generation. Its purpose is to introduce diversity in the population with the hope of that diversity leading to a better breed with higher survival chances. Mutation in genetic algorithms works in the same way and serves the same purpose i.e. introducing diversity. The probability of mutation happening in a generation is usually kept low at around 1-5% so as to not steer the population away from the optimum solution's path. When the genotype is binary, mutation can be implemented as-

- Bit flip operation: Randomly flip the bits of random chromosomes in a population
- Bit swap: Swap the bits on random positions in a chromosome

3.4 Survival Selection:

This step involves choosing the parts of the population to promote to the next generation and the parts to discard entirely. This step has a major effect on the performance of the genetic algorithm. Selection of chromosomes can be done in two ways: Age-based selection or Fitness-based selection methods. In age-based selection method, the number of generations for which a chromosome has been in the population is the deciding factor. If it is greater than a fixed number, then it discarded. In fitness based selection, higher the fitness value higher is the chance of being promoted to the next generation. Fitness based selection methods consist of probabilistic methods like Roulette-wheel selection method or tournament selection method.

3.5 Termination Condition:

Termination condition can be when a fixed value of the objective is reached or a fixed number of generations have passed or the current population size is lower than a certain number or there is no or little improvement in the solutions generated over the past few generations.

Choosing the appropriate function to use for crossover, mutation, survival selection, etc. is all based on the specific details of the problem. There is a lot of flexibility in implementing the functions mentioned above. The functions should be tailored to the problem. The more problem-specific knowledge is fed into the algorithm, the better it performs and sooner it converges to the optimum solution. Effective implementation of genetic algorithms is all about balancing the diversity in the population with problem-specific knowledge. If the diversity is kept low, the algorithm has a high chance of converging at the nearest local optimum. Crossover and mutation help in maintaining diversity in the population.

For this particular problem of optimizing sensitivity, I am using two-point crossover function applied for each variable individually. The mutation function is a bit flip function which has a 5-10% probability of occurring in a generation. For the survival selection function, I tried many methods like roulette wheel selection method and tournament selection method. The lack of problem specific knowledge in this case, led to failure of these methods to converge at a point. Therefore, I choose to select top 40% of the population with highest fitness value and bottom 10% of the population (with lowest fitness value) for the next generation and discarded the 50% of the population left. The bottom 10% was kept to maintain diversity. Thus, the population is reduced to half its number after every generation. This led to a steady increase in the average fitness value of each generation and the algorithm converged to a point. The termination condition was kept at when the population size was reduced to less than 10.

Chapter 4 – Results and Discussion

The code is written in C++ language. The C++ library 'Eigen' has been used for calculation of the eigenvalues and eigenvectors of the Hamiltonian. The code is given and explained in the Appendix section of the report. The GCC compiler (version 4.4 or higher) is required for compilation. The command used for compilation is

```
g++ -std=c++11 main.cpp -o output_filename
```

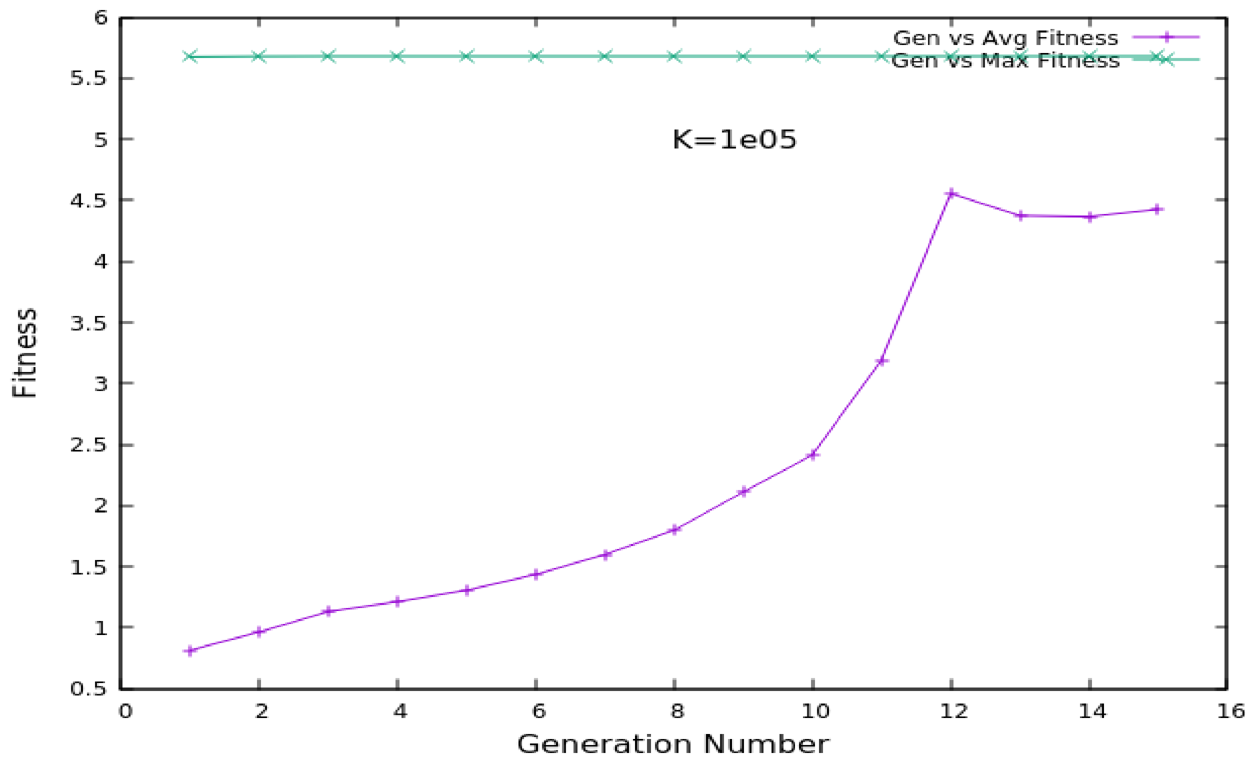
The results are stored in 'output.txt' in the same folder as the main.cpp file. To plot the Sensitivity vs. Theta plot compile and run the 'phi_vs_theta.cpp' file in the same manner as above. Enter the values of (Ax, Ay, Az) obtained in output.txt as input. The output of this program is stored in 'phiS.txt'.

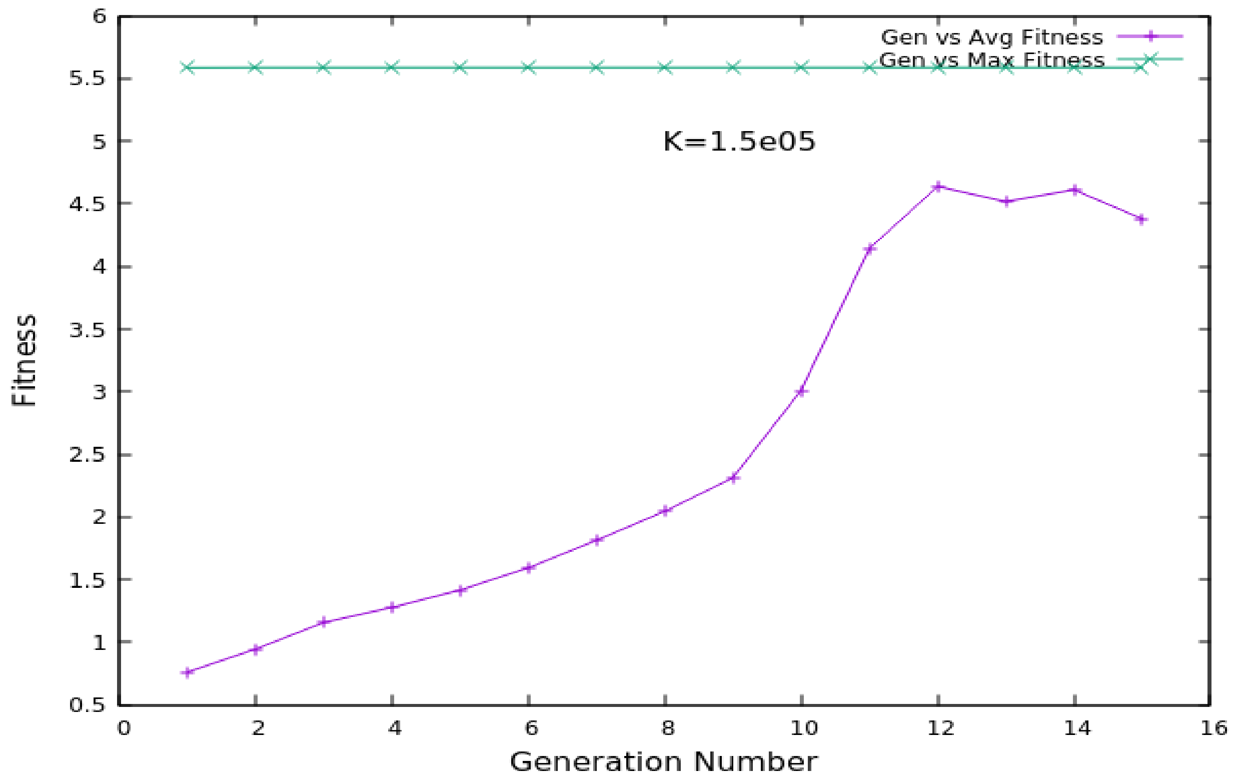
'gnuplot' software has been used to plot all the graphs. The file 'printGraphs.p' has the code to plot the two kind graphs. To plot the graphs, open gnuplot, navigate to the correct directory and simply run the command,

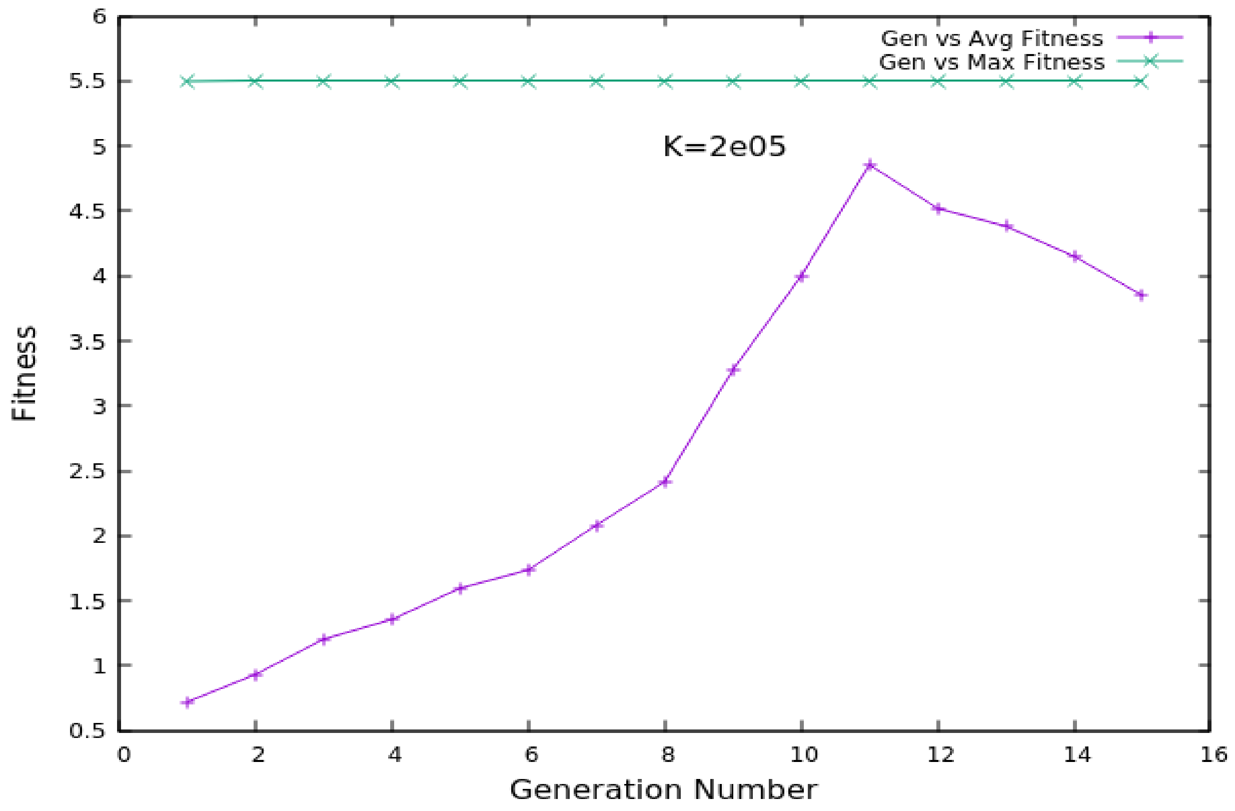
```
load 'printGraphs.p'
```

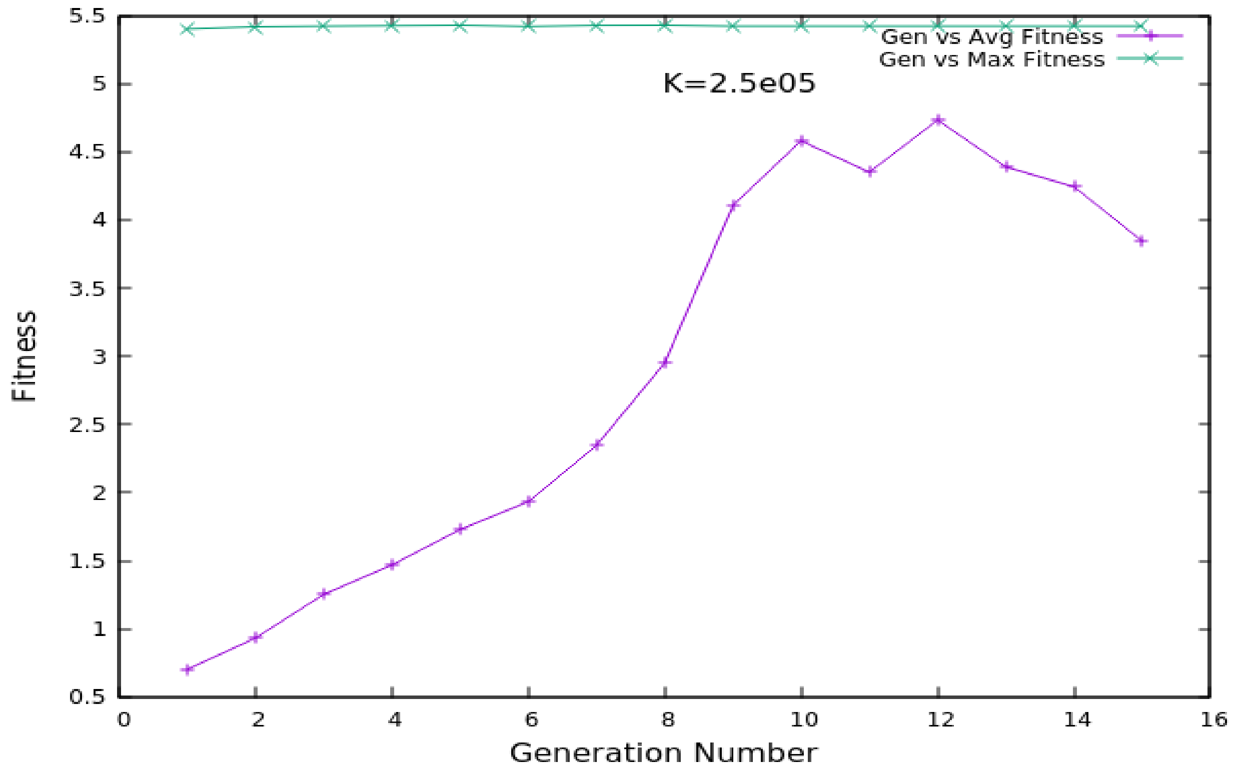
The following Fitness vs. Generation number plots for different values of K show the average fitness moving towards the maximum fitness in each generation.

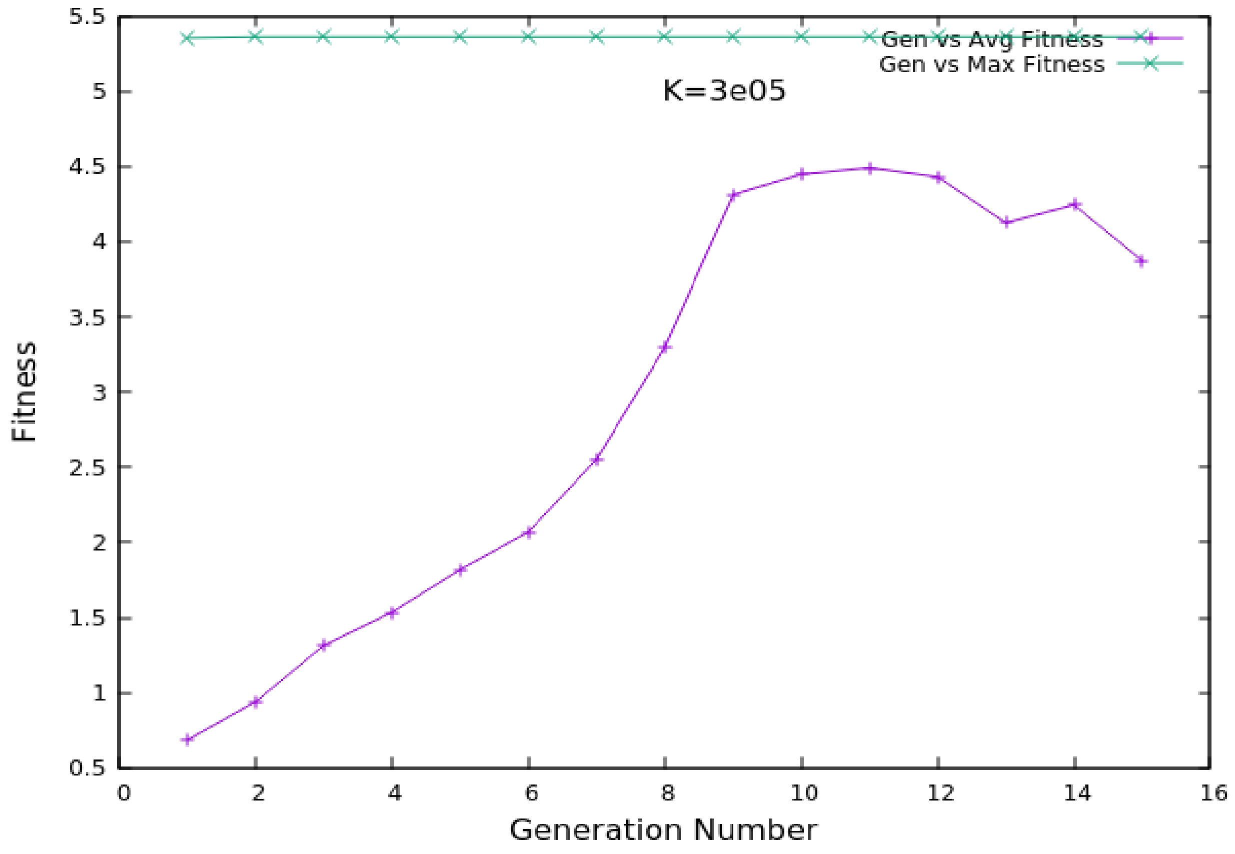
In more than 25 tests, the maximum fitness was reached was about ~ 5.6.

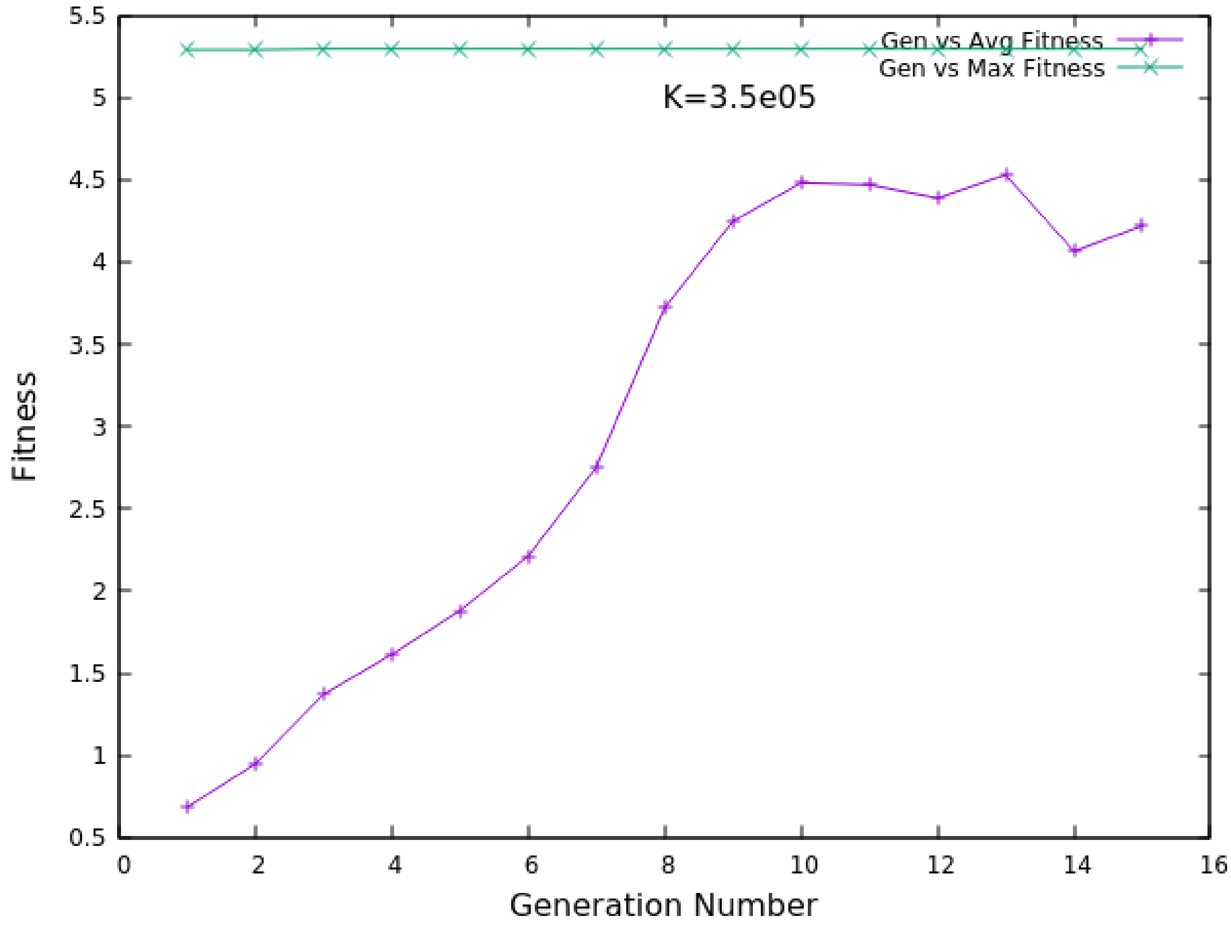



Gen vs Avg Fitness
Gen vs Max Fitness
K=1.5e05
Fitness
Generation Number
0
2
4
6
8
10
12
14
16
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6


Gen vs Avg Fitness
Gen vs Max Fitness
K=2e05
Fitness
Generation Number
0
2
4
6
8
10
12
14
16
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6


Gen vs Avg Fitness
Gen vs Max Fitness
K=2.5e05
Fitness
Generation Number
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
2
4
6
8
10
12
14
16

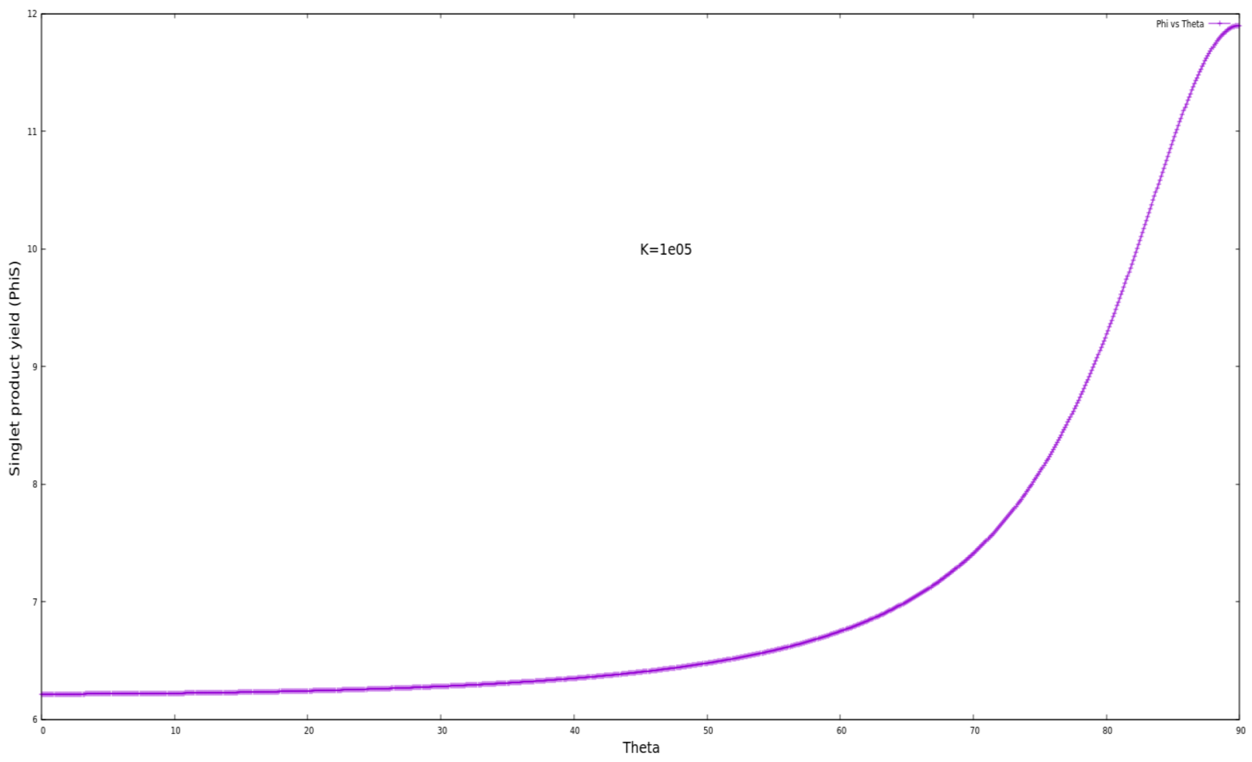

Gen vs Avg Fitness
Gen vs Max Fitness
K=3e05
Fitness
Generation Number
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
2
4
6
8
10
12
14
16

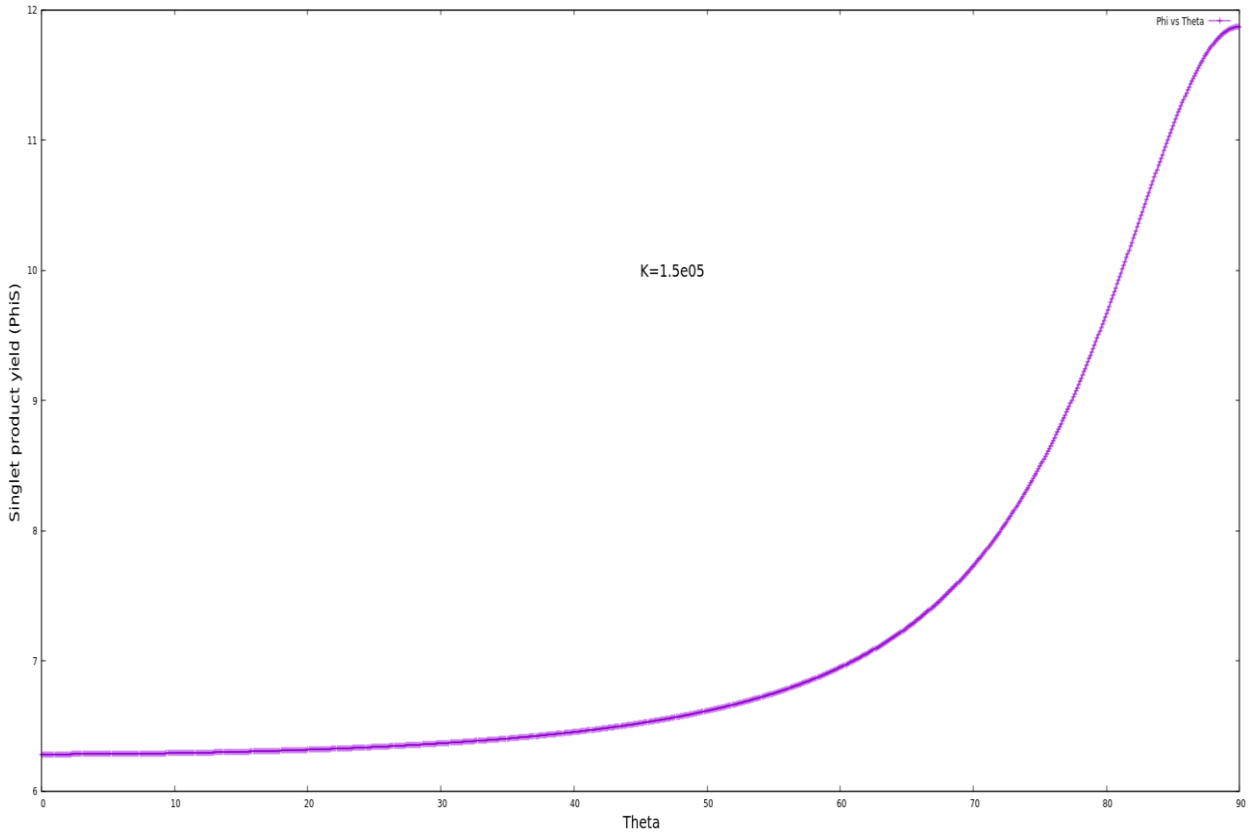


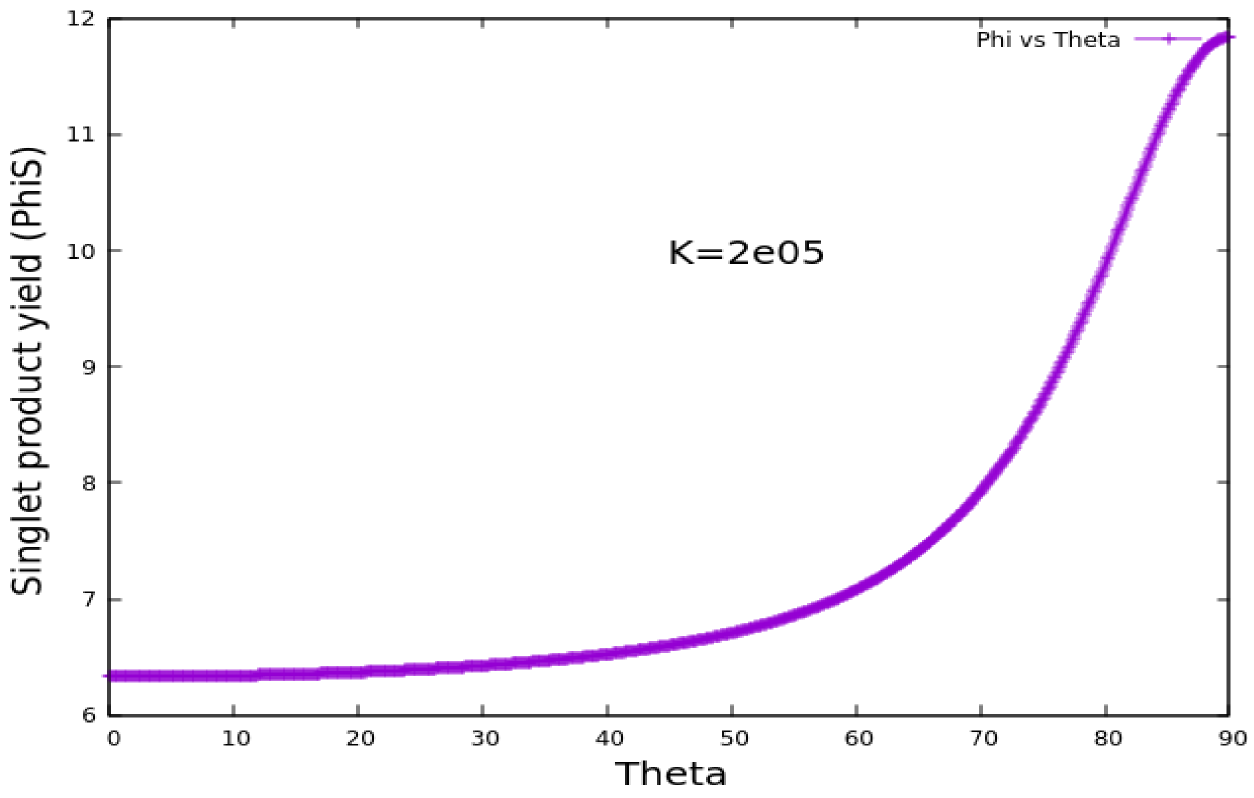
Multiple combination of values of (Ax, Ay, Az) give fitness in the range of 5.2-5.7. Some of them are – (1.2e06, 1.2e06, 510000), (5.72e06, 5.72e06, 540000), (9.88e06, 9.88e06, 26000), (10.04e06, 10.04e06, 26000), etc. The common pattern in all the solutions is that of,

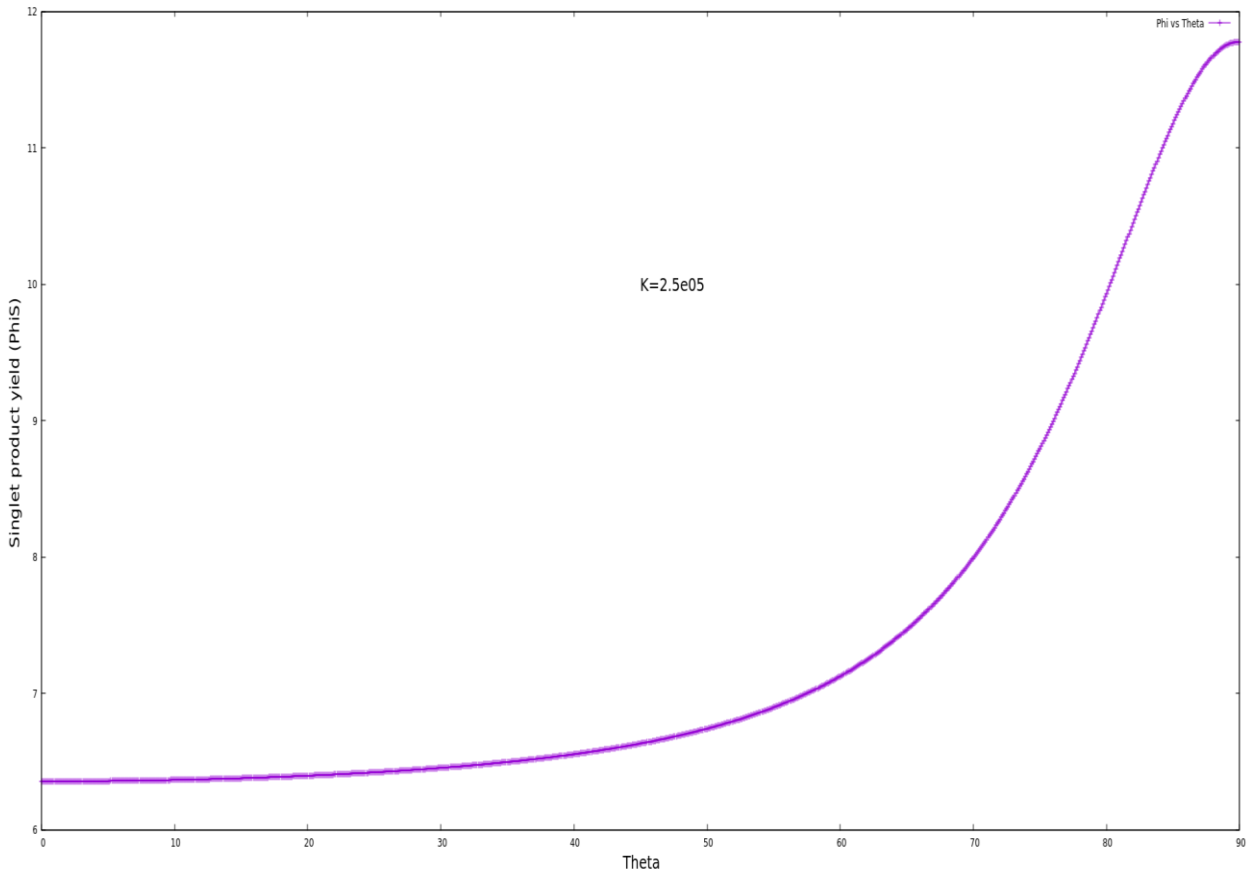
(Ax=Ay >Az). This falls into the disk-shaped hyperfine parameter regime.

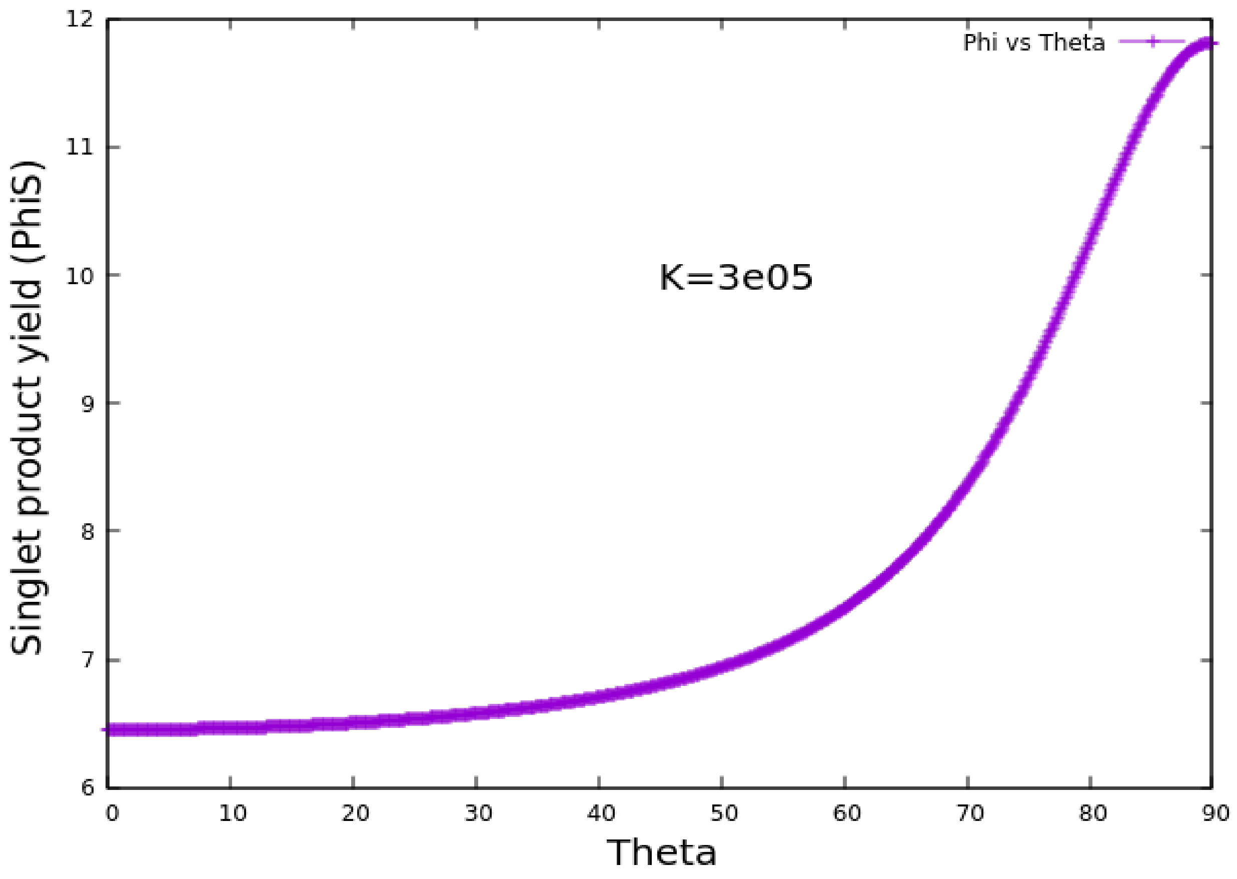
The following plots are for Singlet product yield vs. Theta. They show a steadily increasing function for different values of K.

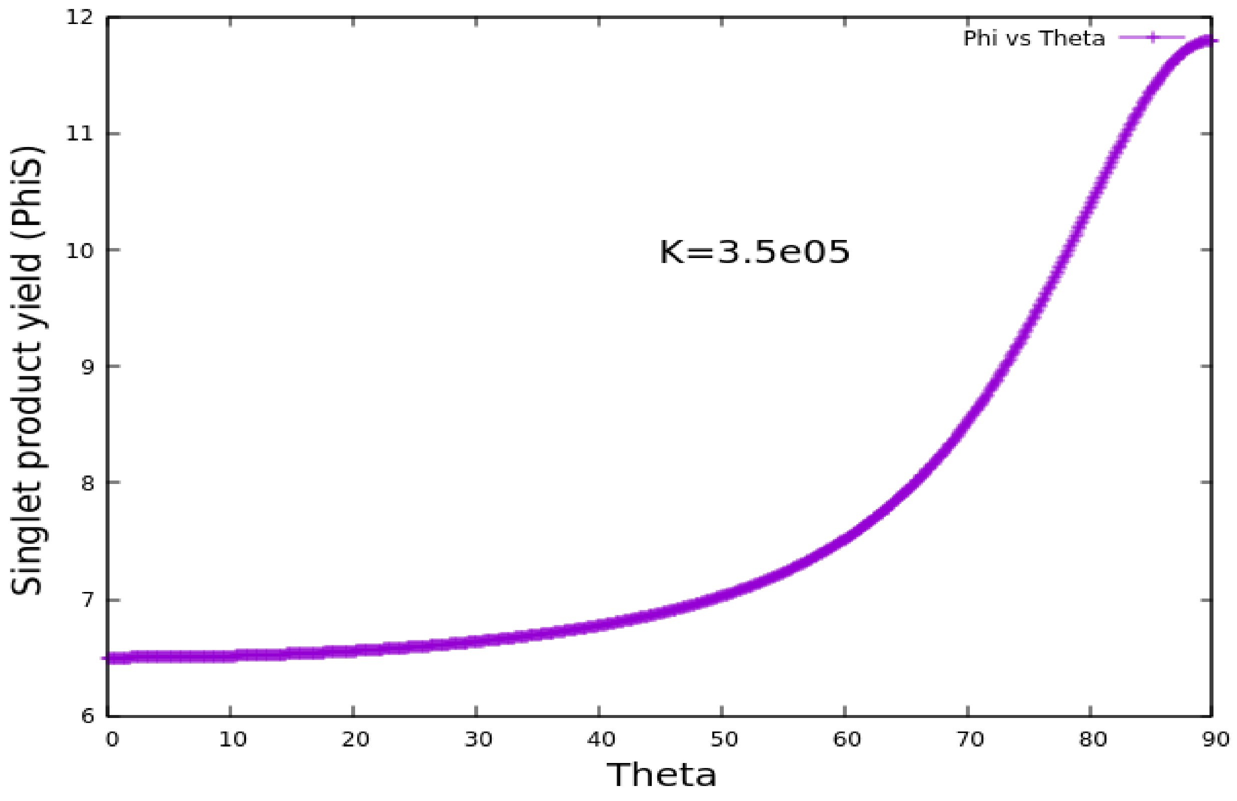

Phi vs Theta
K=1e05
Singlet product yield (PhiS)
Theta
6
7
8
9
10
11
12
0
10
20
30
40
50
60
70
80
90


Phi vs Theta
K=1.5e05
Singlet product yield (PhiS)
Theta
6
7
8
9
10
11
12
0
10
20
30
40
50
60
70
80
90


Singlet product yield (PhiS)
Theta
Phi vs Theta
K=2e05
12
11
10
9
8
7
6
0
10
20
30
40
50
60
70
80
90


Singlet product yield (PhiS)
Theta
Phi vs Theta
K=2.5e05
12
11
10
9
8
7
6
0
10
20
30
40
50
60
70
80
90


Phi vs Theta
K=3e05
Singlet product yield (PhiS)
Theta
12
11
10
9
8
7
6
0
10
20
30
40
50
60
70
80
90


Phi vs Theta
K=3.5e05
Singlet product yield (PhiS)
Theta
12
11
10
9
8
7
6
0
10
20
30
40
50
60
70
80
90

Chapter 5 – Conclusion and Future Scope

The values of the variables (Ax, Ay, Az) discovered for the maximum sensitivity help in describing the characteristics of the nuclear part of the molecules which form radical pairs. Nature improves the biological mechanisms generation after generation, by estimating the optimal sensitivity and hyperfine parameters we get closer to the actual working of the avian compass. Also, research is ongoing to create a geomagnetic sensor which can detect its location on Earth based on the local geomagnetic field using similar mechanism. The optimized values calculated can be used for maximizing the sensitivity of the artificial compass. Future scope of work includes the study of the effects of recombination rate and the initial state on the sensitivity of the avian compass.

Appendix A

A.1 The main.cpp file (driver file of the program)- "main.cpp"

```
#include <iostream>
#include <vector>
#include <ctime>
#include <fstream>
#include "fitness.h"
#include "crossover_mutation.h"
#include "survivor_selection.h"
using namespace std;

vector<double> CalcFitness(char** population);
void Crossover(char** population,vector<int> mating_pool);
void Mutation(char** population);

double total_fitness(vector<double>& fitness,int curr_pop_size){
	double t=0;
	for(int i=0;i<curr_pop_size;i++)
		t+=fitness[i];
	return t;
}
int MaxFitness(char** population,vector<double>& fitness,int curr_pop_size){
	double max=0;
	int imax=0;
	for(int i=0;i<curr_pop_size;i++){
		if(fitness[i]>max){
			max=fitness[i];
			imax=i;
		}
	}
	return imax;
}
```

```
double avg_fitness(vector<double>& fitness,int curr_pop_size){
        return total_fitness(fitness,curr_pop_size)/(double)curr_pop_size;
}

void PrintCurrentPop(char** population,int curr_pop_size){
        for(int i=0;i<curr_pop_size;i++)
                cout<<population[i]<<endl;
}

int main(){
        //Allocate memory for the population at its initialize size MAX_POPULATION
        int MAX_POPULATION;
        cin>>MAX_POPULATION;
        int p=0,m=MAX_POPULATION;
        while(m>10){
                m/=2;
                p++;
        }
        char** population[p+1];
        population[0]=(char**)malloc((MAX_POPULATION)*sizeof(char*));
        for(int i=0;i<MAX_POPULATION;i++)
                population[0][i]=(char*)malloc(31*sizeof(char));
        int curr_pop_size=MAX_POPULATION;

        //Create a filetream for storing output
        fstream ofile{"output.txt",ios::out};
        ofile<<"Initial population size="<<curr_pop_size<<endl<<"Gen   "<<"Avg F    "<<"Max F      "<<"Ax        Ay        Az
        A"<<endl;;

//Seed the random number generator to the system time
srand(clock());
//Initialize the population space with 1s and 0s randomly
for(int i=0;i<curr_pop_size;i++){
        for(int j=0;j<30;j++){
                population[0][i][j]=(char)(rand()%2+48);
        }
        population[0][i][30]='\0';
}
vector<double> fitness;
fitness=CalcFitness(population[0],curr_pop_size);
double curr_fitness=avg_fitness(fitness,curr_pop_size);
int count=0, gen=1;
cout<<"Generation 1 complete. Avg fitness="<<avg_fitness(fitness,curr_pop_size)<<endl<<endl;
int max=MaxFitness(population[0],fitness,curr_pop_size);
double A = 2*pi/pow((  pow(Ax(population[gen-1][max]),2)  +  pow(Ay(population[gen-1][max]),2)  +  pow(Az(population[gen-
1][max]),2) ),0.5);
ofile<<gen<<"   "<<curr_fitness<<"   "<<fitness[max]<<"    "<<Ax(population[gen-1][max])<<"   "<<Ay(population[gen-
1][max])<<"   "<<Az(population[gen-1][max])<<"   "<<A<<endl;
//Termination condition is when the size of the population falls to less than 10
while(curr_pop_size>10) {
                curr_pop_size/=2;
                cout<<"curr_pop_size="<<curr_pop_size<<endl;
                population[gen]=(char**)malloc((curr_pop_size)*sizeof(char*));
                for(int i=0;i<curr_pop_size;i++)
                        population[gen][i]=(char*)malloc(31*sizeof(char));
                for(int i=0;i<curr_pop_size;i++){
                        for(int j=0;j<30;j++){
                                population[gen][i][j]=(char)(rand()%2+48);
                        }
                        population[gen][i][30]='\0';
                }

                SurvivorSelection(population[gen-1],population[gen],fitness,curr_pop_size);
                Crossover(population[gen],curr_pop_size);
                Mutation(population[gen],curr_pop_size);
                fitness=CalcFitness(population[gen],curr_pop_size);
                curr_fitness=avg_fitness(fitness,curr_pop_size);
                gen++;

                cout<<"Generation "<<gen<<" complete. Avg Fitness="<<curr_fitness<<endl;

                max=MaxFitness(population[0],fitness,curr_pop_size);

                cout<<”Max Fitness=”<<max<<endl<<endl;

                A = 2*pi/pow((  pow(Ax(population[gen-1][max]),2)  +  pow(Ay(population[gen-1][max]),2)  +
                pow(Az(population[gen-1][max]),2) ),0.5);

                ofile<<gen<<"   "<<curr_fitness<<"   "<<fitness[max]<<"    "<<Ax(population[gen-1][max])<<"
                "<<Ay(population[gen-1][max])<<"   "<<Az(population[gen-1][max])<<"   "<<A<<endl;

        }


        //Output the A corresponding to the last generation maximum fitness value and the constant K taken.

        fstream Afile{"AvsK.txt",ios::out|ios::app};

        Afile<<A<<"    "<<K<<endl;

        //Print the last population

        PrintCurrentPop(population[gen-1],curr_pop_size);

        MaxFitness(population[gen-1],fitness,curr_pop_size);

        return 0;

}
```

A.2 The header file to calculate fitness of the population- “fitness.h”

```
#include <eigen3/Eigen/Eigenvalues>

#include <eigen3/Eigen/Core>

#define pi 3.14159265

#define K 1e5

using namespace std;


typedef Eigen::Matrix<double,8,8> Matrix8d;

typedef Eigen::Matrix<double,8,1> Vector8d;

typedef Eigen::Matrix<double,1,8> Vector8dR;


double Ax(char* chromosome){

        char tmp[10];

        for(int i=0;i<10;i++)

                tmp[i]=chromosome[i];

        double ans=0;

        for(int i=0;i<10;i++)

                ans+=(tmp[i]-'0')*pow(2,i);

        return ans*1e06;

}

double Ay(char*chromosome){

        char tmp[10];

        for(int i=10;i<20;i++)

                tmp[i-10]=chromosome[i];

        double ans=0;

        for(int i=0;i<10;i++)

                ans+=(tmp[i]-'0')*pow(2,i);

        return ans*1e06;

}


double Az(char* chromosome){

        char tmp[10];

        for(int i=20;i<30;i++)

                tmp[i-20]=chromosome[i];
        double ans=0;
        for(int i=0;i<10;i++)
                ans+=(tmp[i]-'0')*pow(2,i);
        return ans*1e06;
}

double f(double tmp){
        return K*K/(K*K + tmp*tmp);
}

double phiS(double* evalue, Vector8d* evector,Matrix8d* Ps){
        double ans=0;
        for(int i=0;i<8;i++){
                for(int j=0;j<8;j++){
                        Vector8dR temp;
                        double p=0;
                        temp=(*Ps)*evector[j];
                        p=temp.dot(evector[i]);
                        ans+=p*p+f(evalue[i]-evalue[j]);
                }
        }
        return ans/2;
}

vector<double> CalcFitness(char** population,int curr_pop_size){
        vector<double> fitness;
        for(int i=0;i<curr_pop_size;i++){
        double gamma_constant=4.132e06;
//Y*B Joules is in the form of energy. Since we are using freq units, we have to take Y*B as (Y*B*2*pi/h) per seconds. Since Y=0.5*u*g and g=2 (free electron), Y=u. Here h= Planck's constant= 6.626e-34 J-s, u=Bohr Magneton= 9.274e-24 J/T and B=local magnetic field = 47e-06 T
        int theta1=0,theta2=90;
//Construct the Hamiltonians. We construct 2 Hamiltonians - one for theta = 0 deg and one for theta = 90 deg
                Matrix8d H0;
                Matrix8d H90;
```

```
                    H0(0,3)=H0(0,4)=H0(0,5)=H0(0,7)=0;

                    H0(1,2)=H0(1,4)=H0(1,5)=H0(1,6)=0;

                    H0(2,1)=H0(2,5)=H0(2,6)=H0(2,7)=0;

                    H0(3,0)=H0(3,4)=H0(3,6)=H0(3,7)=0;

                    H0(4,0)=H0(4,1)=H0(4,3)=H0(4,7)=0;

                    H0(5,0)=H0(5,1)=H0(5,2)=H0(5,6)=0;

                    H0(6,1)=H0(6,2)=H0(6,3)=H0(6,5)=0;

                    H0(7,0)=H0(7,2)=H0(7,3)=H0(7,4)=0;


          H0(0,1)=H0(0,2)=H0(1,0)=H0(1,3)=H0(2,0)=H0(2,3)=H0(3,1)=H0(3,2)=H0(4,5)=H0(4,6)=H0(5,4)=H0(5,7)=H0(6,4)=H0(6,
7)=H0(7,5)=H0(7,6)=gamma_constant*sin(theta1*pi/180);

                    H0(0,6)=H0(1,7)=H0(6,0)=H0(7,1)=Ax(population[i])-Ay(population[i]);

                    H0(1,1)=H0(6,6)=Az(population[i]);

                    H0(2,2)=H0(5,5)=-Az(population[i]);

                    H0(0,0)=Az(population[i])+2*gamma_constant*cos(theta1*pi/180);

                    H0(3,3)=-Az(population[i])-2*gamma_constant*cos(theta1*pi/180);

                    H0(4,4)=-Az(population[i])+2*gamma_constant*cos(theta1*pi/180);

                    H0(7,7)=Az(population[i])-2*gamma_constant*cos(theta1*pi/180);


                    H90(0,3)=H90(0,4)=H90(0,5)=H90(0,7)=0;

                    H90(1,2)=H90(1,4)=H90(1,5)=H90(1,6)=0;

                    H90(2,1)=H90(2,5)=H90(2,6)=H90(2,7)=0;

                    H90(3,0)=H90(3,4)=H90(3,6)=H90(3,7)=0;

                    H90(4,0)=H90(4,1)=H90(4,3)=H90(4,7)=0;

                    H90(5,0)=H90(5,1)=H90(5,2)=H90(5,6)=0;

                    H90(6,1)=H90(6,2)=H90(6,3)=H90(6,5)=0;

                    H90(7,0)=H90(7,2)=H90(7,3)=H90(7,4)=0;


          H90(0,1)=H90(0,2)=H90(1,0)=H90(1,3)=H90(2,0)=H90(2,3)=H90(3,1)=H90(3,2)=H90(4,5)=H90(4,6)=H90(5,4)=H90(5,7)
=H90(6,4)=H90(6,7)=H90(7,5)=H90(7,6)=gamma_constant*sin(theta2*pi/180);

                    H90(0,6)=H90(1,7)=H90(6,0)=H90(7,1)=Ax(population[i])-Ay(population[i]);

                    H90(1,1)=H90(6,6)=Az(population[i]);

                    H90(2,2)=H90(5,5)=-Az(population[i]);

                    H90(0,0)=Az(population[i])+2*gamma_constant*cos(theta2*pi/180);

                    H90(3,3)=-Az(population[i])-2*gamma_constant*cos(theta2*pi/180);

                    H90(4,4)=-Az(population[i])+2*gamma_constant*cos(theta2*pi/180);
```

```
H90(7,7)=Az(population[i])-2*gamma_constant*cos(theta2*pi/180);

//Calculate and store the eigenvalues and eigenvectors of each Hamiltonianian
Vector8d evector0[8];
Vector8d evector90[8];
double evalue0[8]={0};
double evalue90[8]={0};

Eigen::EigenSolver<Matrix8d> sol0(H0);
for(int j=0;j<8;j++){
        evalue0[j]=sol0.eigenvalues()[j].real();
        evector0[j]=sol0.eigenvectors().col(j).real();
        //cout<<evector0[j]<<"  "<<endl;
}
Eigen::EigenSolver<Matrix8d> sol90(H90);
for(int j=0;j<8;j++){
        evalue90[j]=sol90.eigenvalues()[j].real();
        evector90[j]=sol90.eigenvectors().col(j).real();
}

//Construct Singlet Projection Operator
Matrix8d Ps;
for(int j=0;j<8;j++){
        Ps(0,j)=0;
        Ps(3,j)=0;
        Ps(4,j)=0;
        Ps(7,j)=0;
}
Ps(1,0)=Ps(1,3)=Ps(2,0)=Ps(2,3)=0;
Ps(1,1)=Ps(2,2)=1/4;
Ps(1,2)=Ps(2,1)=-1/4;
Ps(5,4)=Ps(5,7)=Ps(6,4)=Ps(6,7)=0;
Ps(5,5)=Ps(6,6)=1/4;
Ps(5,6)=Ps(6,5)=-1/4;
```

```
        for(int j=4;j<8;j++){
            for(int k=1;k<3;k++)
                Ps(k,j)=0;
        }
        for(int j=0;j<4;j++){
            for(int k=5;k<7;k++)
                Ps(k,j)=0;
        }
        double temp=phiS(evalue0,evector0,&Ps)-phiS(evalue90,evector90,&Ps);
        if(temp<0)      //We convert negative values to positive since it's the difference we are concerned with.
            fitness.push_back(-temp);
        else
            fitness.push_back(temp);
    }
    return fitness;
}
```

A.3 Header file containing the functions to carry out Crossover and mutation- “crossover_mutation.h”

```
void Crossover(char** population,int curr_pop_size)

//Two-point crossover. Since 3 variables of 10-bytes each are involved, the cross-over points are at 4 bytes and 6 bytes for each variable.

{
        int k=0,count=0;
        cout<<"Crossover()"<<endl;
        while(k<curr_pop_size-1){
                char tmp[7];
                for(int i=0;i<4;i++){
                        tmp[i]=population[k][i];
                        population[k][i]=population[k+1][i];
                        population[k+1][i]=tmp[i];
                }
                for(int i=7;i<10;i++){
                        tmp[i-3]=population[k][i];
                        population[k][i]=population[k+1][i];
                        population[k+1][i]=tmp[i-3];
                }
                for(int i=0;i<4;i++){
                        tmp[i]=population[k][i+10];
                        population[k][i+10]=population[k+1][i+10];
                        population[k+1][i+10]=tmp[i];
                }
                for(int i=7;i<10;i++){
                        tmp[i-3]=population[k][i+10];
                        population[k][i+10]=population[k+1][i+10];
                        population[k+1][i+10]=tmp[i-3];
                }
                for(int i=0;i<4;i++){
                        tmp[i]=population[k][i+20];
                        population[k][i+20]=population[k+1][i+20];
                        population[k+1][i+20]=tmp[i];
                }
                for(int i=7;i<10;i++){
                        tmp[i-3]=population[k][i+20];
                        population[k][i+20]=population[k+1][i+20];
                        population[k+1][i+20]=tmp[i-3];
                }
                k+=2;
        }
}
//Mutation has 5% probability of happening for each generation
void Mutation(char** population,int curr_pop_size) {
        srand(clock());
        if(rand()%100<=5){
                cout<<"Mutation()"<<endl;
                for(int i=0;i<curr_pop_size/50;i++){
                        if(population[rand()%curr_pop_size][rand()%30]=='0')
                                population[rand()%curr_pop_size][rand()%30]='1';
                        else
                                population[rand()%curr_pop_size][rand()%30]='0';
                }
        }
}
```

A.4 Header file to filter out the top 40% and bottom 10% of the population- “survivor_selection.h”

```
int findMax(vector<double>& fitness,int curr_pop_size){

        int imax=0;

        double max=-999999999;

        for(int i=0;i<curr_pop_size;i++){

                if(fitness[i]>max){

                        max=fitness[i];

                        imax=i;

                }

        }

        fitness[imax]=-99999999.00;

        return imax;

}


int findMin(vector<double>& fitness,int curr_pop_size){

        int imin=0,min=999999999;

        for(int i=0;i<curr_pop_size;i++){

                if(fitness[i]<min && fitness[i]>0){

                        min=fitness[i];

                        imin=i;

                }

        }

        fitness[imin]=999999999;

        return imin;

}


void SurvivorSelection(char** old_population, char** new_population, vector<double> fitness,int curr_pop_size){

        vector<int> mating_pool;

        int count=0;

        while(count<2*curr_pop_size*4/10){

                mating_pool.push_back(findMax(fitness,2*curr_pop_size));

                count++;
        }
        count=0;
        while(count<2*curr_pop_size/10){
                mating_pool.push_back(findMin(fitness,2*curr_pop_size));
                count++;
        }
        count=0;
        while(count<mating_pool.size()){
                for(int i=0;i<30;i++){
                        new_population[count][i]=old_population[mating_pool[count]][i];
                }
                count++;
        }
}
```

REFERENCES

1. Jayendra N. Bandyopadhyay, Tomasz Paterek and Dagomir Kaszlikowski, *'Quantum Coherence and Sensitivity of Avian Magnetoreception'*, Physical Review Letters, Sept 2012
2. C.R. Timmel, U. Till, B. Brocklehurst, K. A. Mclauchlan and P.J. Hore, *'Effects of weak magnetic fields on free radical recombination reactions'*, Molecular Physics, Vol. 95, 1998
3. Vishvendra Singh Poonia, Kiran Kondabagil, Dipankar Saha and Swaroop Ganguly, *'On the functional window of the avian compass'*, Dept. of Electrical Engg, IIT Bombay, Feb 2017.
4. Erik M. Gauger, Elisabeth Rieper, John J. L. Morton, Simon C. Benjamin and Vlatko Vedral, 'Sustained Quantum Coherence and Entanglement in the Avian Compass' , Physical Review Letters, Jan 2011
5. The Eigen 3.3 template library for Linear Algebra http://eigen.tuxfamily.org/index.php?title=Main_Page
6. Genetic Algorithms Tutorial https://www.tutorialspoint.com/genetic_algorithms/